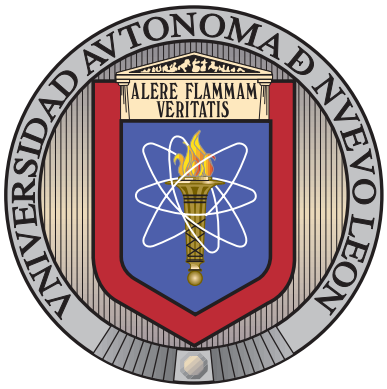
Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección de Estudios de Posgrado



Secuenciación de Trabajos en Sistemas de Producción Flexibles

por

César Arturo Sáenz Alanís

como requisito para obtener el grado de

Maestro en Ciencias

en Ingeniería de Sistemas

Julio 2015

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección de Estudios de Posgrado

Secuenciación de Trabajos en Sistemas de Producción Flexibles

por

César Arturo Sáenz Alanís

como requisito para obtener el grado de

Maestro en Ciencias

en Ingeniería de Sistemas

Julio 2015

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección de Estudios de Posgrado

Los miembros del Comité de Tesis recomendamos que la Tesis «Secuenciación de Trabajos en Sistemas de Producción Flexibles», realizada por el alumno César Arturo Sáenz Alanís, con número de matrícula 1398095, sea aceptada para su defensa como requisito para obtener el grado de Maestro en Ciencias en Ingeniería de Sistemas.

El Comité de Tesis

Dra. María Angelica Salazar Aguilar

Asesor

Dr. Vincent André Lionel Boyer

Revisor

Dr. Juan Carlos Cabada Amaya

Revisor

Vo. Bo.

Dr. Simón Martínez Martínez

Subdirector de Estudios de Posgrado

San Nicolás de los Garza, Nuevo León Julio 2015

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Primeramente agradezco a la coordinación del PISIS por haberme dado la oportunidad de estar en el programa; a la Universidad Autónoma de Nuevo León y a la Facultad de Ingeniería Mecánica y Eléctrica por haberme apoyado con becas y viáticos que me sirvieron para realizar mi estancia en Canadá.

Agradezco también al Consejo Nacional de Ciencia y Tecnología por la beca de manutención que me otorgó a lo largo de mi maestría.

Gracias a la Dra. María Angélica Salazar Aguilar, mi asesora de tesis, que estuvo apoyándome en todo momento, ayudándome a resolver mis dudas y guiándome a cada paso y constantemente. También gracias a los miembros del comité Vincent Boyer, por proporcionarme ideas a desarrollar y Juan Carlos Cabada Amaya, que me dió la oportunidad de estudiar y dar una solución un caso real en la industria cervecera. Agradezco al Dr. Leandro Callegari Coelho por haberme recibido y asesorado durante mi estancia en la Universidad Laval en Québec, Canadá.

Gracias también a mi familia, por todo el apoyo incondicional que me han brindado en cada etapa de mi vida, a mis amigos que me dieron consejos siempre que necesitaba.

A mis profesores y compañeros de PISIS, a todos, gracias.

Resumen

César Arturo Sáenz Alanís.

Candidato para el grado de Maestro en Ciencias
en Ingeniería de Sistemas.

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

Título del estudio:

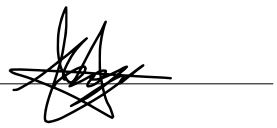
Secuenciación de Trabajos en Sistemas de Producción Flexibles

Número de páginas: 54.

Objetivos y método de estudio: Uno de los objetivos de este trabajo es el de estudiar un problema complejo de secuenciación de trabajos en sistemas flexibles, conocido en la literatura como *Flexible Job Shop Scheduling Problem* (FJSP); y proponer un algoritmo de optimización para resolverlo. El algoritmo se basa en un esquema tipo ALNS (*Adaptive Large Neighborhood Search*) híbrido, el cual, en ciertas iteraciones, hace llamadas al *branch-and-bound* de CPLEX para resolver el FJSP, usando un modelo propuesto por Vahid Roshanaei[47]. El segundo de los objetivos es el estudio de un problema de secuenciación de trabajos presente en una empresa cervecera de la localidad. Puesto que las características y restricciones del problema de producción de cerveza difieren con las del clásico FJSP, se desarrolla un algoritmo

de optimización de tipo GRASP (*Greedy Randomized Adaptive Search Procedure*), el cual será el punto de inicio para una futura implementación en la empresa.

Contribuciones y conclusiones: El ALNS propuesto para la solución del FJSP probó ser eficiente para un gran número de instancias tomadas de la literatura, alcanzando soluciones óptimas para más de la mitad de las instancias en las que el óptimo ha sido reportado en la literatura. Para mejorar la calidad de las soluciones (no óptimas) generadas por el algoritmo propuesto, se propone variar los parámetros del algoritmo, o bien, añadirle otro tipo de reactividad para que ajuste de manera automática los parámetros. En cuanto al caso estudiado de la compañía cervecera, nos proporcionaron dos instancias reales junto con la solución implementada en la planta, ésta se comparó con la solución reportada por el GRASP propuesto y se observó que la solución reportada por GRASP permite un ahorro de hasta el 28 % (6 días) con respecto al tiempo de producción requerido por la solución implementada por la compañía.

Firma del asesor: 

Dra. María Angélica Salazar Aguilar

Capítulo 1

Introducción

En este trabajo se aborda el Problema de secuenciación de trabajos en sistemas flexibles, conocido por sus siglas en inglés como FJSP (*Flexible Job Shop Scheduling Problem*), el cual consiste en generar una programación de producción de cierta cantidad de trabajos para satisfacer una demanda dada; cada trabajo está formado por un conjunto de operaciones que tienen que ser procesadas en un orden específico, y cada operación puede ser procesada en una máquina dentro de un conjunto de máquinas capaces de realizar dicha operación. Este problema está presente en el ámbito industrial en donde las empresas, con el fin de maximizar su capacidad de producción, manejan líneas de producción en paralelo. Las líneas de producción pueden contar con características idénticas o no.

Revisando la literatura existente en el FJSP, se observó un área de oportunidad para el desarrollo de una heurística eficiente que permita encontrar soluciones de buena calidad en un tiempo razonable, de ahí que en esta tesis se propone un algoritmo híbrido basado en la metaheurística ALNS (*Adaptive Large Neighborhood Search*) y un procedimiento de *Branch-and-Bound* que es llamado en algunas iteraciones del algoritmo. El desempeño del algoritmo es evaluado sobre varias clases de instancias tomadas de la literatura.

Además del problema mencionado anteriormente, se estudia un caso real de un empresa cervecera de la localidad, en donde no cuentan con un sistema automatizado para programar la secuencia de producción. Dicha empresa requiere mejorar la forma en que se planea la producción de cerveza, con el fin de maximizar la pro-

ducción y cubrir la demanda total de los diferentes productos que maneja. Debido a la gran cantidad de demanda que se tiene que cubrir constantemente y a la gran cantidad de variables y restricciones requeridas para la planeación de la producción, la programación se realiza básicamente en función de la experiencia del tomador de decisiones y decisiones heurísticas. Es por ello que es necesario desarrollar e implementar herramientas de optimización que permitan generar soluciones de mejor calidad en el menor tiempo posible. En esta tesis se propone un algoritmo metaheurístico tipo GRASP (*Greedy Randomized Adaptive Search Procedure*) con el fin de obtener planes de producción que maximicen la capacidad de producción de la planta.

1.1 Objetivo

Desarrollar e implementar un algoritmo de optimización para resolver el FJSP de manera eficiente en un tiempo de cómputo competitivo. Además, estudiar el proceso de planeación de producción que enfrenta una empresa cervecera local. Finalmente, proponer un algoritmo de optimización que resuelva el problema de planeación de producción de cerveza de manera eficiente y que sirva como soporte para el tomador de decisiones.

1.2 Justificación

Existe una gran cantidad de empresas con alto nivel de utilización, que tienen grandes volúmenes de demanda que deben satisfacer en un tiempo determinado, para eso, es una estrategia común utilizar líneas de producción que trabajan en paralelo para maximizar la producción. Sin embargo, un gran número de empresas carecen de herramientas para optimizar la secuencia de producción de sus múltiples productos, por lo que recurren a la experiencia del tomador de decisones, limitándose a encontrar cualquier solución factible basado en la experiencia, sin revisar la existencia de otras

soluciones que pudieran resultar más rentables.

1.3 Estructura de la tesis

En el Capítulo 2 se muestran los antecedentes de los distintos problemas de secuenciación de tareas relacionados con el presente trabajo.

En el Capítulo 3 se describe de manera más detallada el FJSP, y se describe el algoritmo propuesto para la solución del mismo. Se incluyen además los resultados computacionales obtenidos sobre varios conjuntos de instancias tomadas de la literatura.

En el Capítulo 4 se describe un caso de estudio proveniente de una compañía cervecera. Se presenta un algoritmo tipo GRASP, cuyo desempeño es evaluado en instancias reales e instancias artificiales creadas a partir de la situación real.

Finalmente, en el Capítulo 5 se muestran las conclusiones del trabajo realizado, así como algunas recomendaciones para mejorar el desempeño de los algoritmos propuestos en esta tesis.

Capítulo 2

Antecedentes

En este capítulo se describen algunos trabajos que estudian el problema de secuenciación de trabajos en máquinas paralelas. Existen muchas variantes de este tipo de problemas: i) Problemas en los que la secuenciación de trabajos se hace por lotes, un lote es un conjunto de trabajos que se procesan, comúnmente, simultáneamente y en una misma máquina; y los trabajos que componen el lote pueden ser idénticos o no idénticos. ii) Problemas que consideran tiempos de preparación los cuales pueden ser dependientes o independientes de la secuencia. iii) Problemas que contemplan familias de trabajos, y en algunos casos éstos se procesan por lotes, siempre y cuando los trabajos dentro de un mismo lote sean de la misma familia; este tipo de problemas pueden presentar tiempos de preparación que dependen de la secuencia de lotes. En esta investigación se realiza el estudio de un problema de secuenciación de trabajos en sistemas flexibles presente en la literatura y además, se estudia un caso práctico de un problema de secuenciación de trabajos por lotes, con familias de trabajos y tiempos de preparación que dependen de la secuencia. A continuación se presenta una revisión de la literatura relacionada con los problemas de estudio.

2.0.1 Problema de secuenciación de trabajos en sistemas flexibles

El problema de secuenciación de trabajos en sistemas flexibles hace referencia al *Flexible Job shop Scheduling Problem* (FJSP, por sus siglas en inglés).

El FJSP es una extensión del JSP (*Job Shop Scheduling Problem*), el cual es considerado como un problema NP-duro [17]. En el JSP se tienen n trabajos, cada trabajo j consta de cierto número de operaciones (n_j) que tienen que realizarse en un orden establecido. Se cuenta también con un conjunto de máquinas que trabajan en paralelo, cada una de las operaciones de cada trabajo o_j se tiene que llevar a cabo en una máquina en específico. El objetivo es encontrar la programación en la que se realizarán las operaciones de manera que se minimize el makespan (C_{max}).

En el FJSP se agrega la parte de la flexibilidad al JSP, la cual se refiere a que una misma operación puede realizarse en más de una máquina que esté capacitada para hacer el trabajo.

En el FJSP se tienen m máquinas en el sistema y n trabajos (j_1,...,j_n) a ser procesados. Cada trabajo j está formado por n_j operaciones que tienen restricciones de precedencia. Cada operación $O_{j,l}$ tiene que procesarse en una de las máquinas y el tiempo de procesamiento depende de las características de la máquina.

En este problema se puede presentar flexibilidad parcial o flexibilidad total, según Kacem et al.[23], la flexibilidad parcial se refiere a que existen algunas máquinas multipropósito no idénticas, lo que permite que cada operación pueda ser asignada únicamente en una de las máquinas capaces de procesarla; mientras que en el caso de flexibilidad total, todas las máquinas pueden procesar todas las operaciones. En Brandimarte [1] se expresa que hay dos decisiones a tomar en el FJSP, la asignación de operaciones a las máquinas, y la secuencia de operaciones dentro de las máquinas. Esto implica que el espacio de soluciones factibles se incremente en gran medida, lo que quiere decir que se vuelve aún más complicado de resolver, a optimalidad, que

el JSP.

Por su parte, Fattahi et al.[14] proponen un algoritmo basado en recocido simulado para resolver el FJSP con superposición en las operaciones así como un modelo matemático de programación lineal entera mixta para la solución de instancias pequeñas.

Fattahi et al.[15] proponen un modelo matemático lineal entero mixto para la solución de instancias pequeñas del FJSP, dicho modelo se basa en las posiciones en las que es posible insertar operaciones dentro de cada máquina. Otros ejemplos de modelos lineales enteros son los que se presentan en Ozguven et al.[42]. Estos modelos son los que se toman como base para la formulación de nuevos y mejores modelos de programación lineal entera en Vahid Roshanaei[47]. En éste último se presenta el modelo llamado $MILP - 6$ (ver sección 3.1.1), el cual se basa en los tiempo de terminación de cada operación para comenzar a procesar la operación siguiente dentro de un mismo trabajo. Dicho modelo es utilizado en esta tesis para mejorar las soluciones generadas por el Algoritmo 1, mostrado más adelante en la Sección 3.2.1.

2.1 Programación de trabajos por lotes

En esta sección se mencionan los estudios realizados para la programación de trabajos por lotes, esto es de utilidad ya que la programación que se realiza en el caso de estudio es también de este tipo, con lo que es posible comparar las características que se han abordado con las características específicas de nuestro problema.

Robert C. Dorsey et al.[13] consideran el problema de producir n productos diferentes en m máquinas idénticas en paralelo. Se asume que la producción ocurre en un horizonte de planeación finito que se compone de h periodos de producción. Durante cualquier periodo se puede asignar como máximo un producto en cada máquina. Cuando un producto i es asignado a una máquina se asume que se pro-

ducirán exactamente p_i unidades del mismo. El costo de producción depende del producto y del periodo en el cual fue asignado. Los productos terminados se guardan en el inventario y las demandas se satisfacen con los productos que están en el inventario. El objetivo del problema es determinar una programación que minimice la suma de los costos de producción e inventario a lo largo del horizonte de planeación. Para ello formulan un modelo de programación entera, el cual transforman a un problema de flujo de costo mínimo.

Jay B. Ghosh[18] hace referencia a versiones de una sola máquina y máquinas múltiples para el problema de programación por lotes que surgen cuando existen familias de trabajos y tiempos de preparación entre dichas familias. El objetivo es minimizar la terminación total de los trabajos. En su trabajo hacen observaciones con respecto a la complejidad del problema y las variantes presentadas en él.

T.C.E. Cheng y Z.L. Chen[7] abordaron un problema de programación de varios lotes de trabajos en dos máquinas en paralelo con el objetivo de minimizar el total del tiempo de terminación de los trabajos. Consideran un tiempo de preparación entre el procesamiento de trabajos que pertenecen a lotes distintos. En este trabajo muestran que el problema es NP-duro incluso cuando los tiempos de preparación son independientes de la secuencia y todos los tiempos de procesamiento son iguales.

Kim et al.[25] estudian la programación de transferencia de lotes de trabajos no idénticos en dos etapas con máquinas idénticas en paralelo en cada etapa.

Cheng et al.[6] atacaron un problema de programación de lotes que aparece comúnmente en un ambiente de manufactura flexible. En este caso, diferentes trabajos tienen que ser procesados en una línea de producción de dos etapas. Estos trabajos son procesados en lotes en la primera máquina, y el tiempo de terminación de un trabajo se define como el tiempo de terminación del lote que lo contiene. Cuando el procesamiento de todos los trabajos de un lote se completa, todo el lote se pasa intacto a la segunda máquina. Existe un tiempo de preparación que se agrega cada vez que se forma un lote en cualquier máquina. El objetivo es encontrar la

composición de los lotes y el orden en el que serán procesados con el fin de minimizar el makespan. Para ello, propusieron algoritmos de optimización heurística en cada fase del problema con el fin de tener mayor diversidad en las soluciones y evitar estancamientos en óptimos locales.

Una metodología sistemática de dos pasos es propuesta por C.A. Méndez et al.[34] para resolver el problema de programación de lotes para satisfacer múltiples órdenes de productos con diferentes tiempos de entrega. Además, Méndez y Cerdá[35], introducen un eficiente marco matemático de MILP para la programación reactiva por lotes en multietapas con restricciones de recursos.

Sung et al.[52] consideran un problema de programación para máquinas procesadoras de lotes multietapas donde cada máquina puede procesar un lote de trabajos simultáneamente tal que todos los trabajos contenidos en el lote son liberados al mismo tiempo después de su procesamiento para pasar a la siguiente máquina, esto puede relacionarse con las etapas de producción de cerveza, ya que un mismo lote tiene que pasar a través de las diferentes etapas del proceso cervecero.

Torabi et al.[54] abordan un problema de programación de lotes multiproducto en ciclos, en grupos flexibles determinísticos donde el horizonte de planeación es finito y fijo por la administración. El problema consiste de una parte combinatoria (subproblemas de asignación de máquinas y secuenciación) y una parte continua (subproblemas de dimensionamiento de lotes y programación).

Dastidar y Nagi[12] presentan modelos matemáticos y algoritmos para un problema de programación de producción con división por lotes de operaciones de ensamblado. Las operaciones a procesar se separan en lotes de tamaño adecuado y se asignan a las máquinas paralelas idénticas disponibles. La metodología que siguen se compone de un algoritmo para la separación de lotes seguido de un algoritmo para la programación de los lotes en las máquinas en paralelo con el objetivo de minimizar el makespan.

En Lin et al.[28] estudian un problema de programación de ensambles con tres

máquinas, el cual aparece en muchos procesos de manufactura. En este problema las primeras dos máquinas producen componentes individuales, y la tercera máquina es tratada como una segunda etapa en la que se ensamblan los componentes del lote de la primera etapa. Cada que un lote es formado en la segunda etapa se requiere un tiempo de preparación constante. El objetivo es minimizar el makespan.

En Kashan et al.[24] se investiga el problema de programación de procesamiento de lotes en máquinas idénticas en paralelo, en el cual cada máquina puede procesar un grupo de trabajos simultáneamente como un lote. Cada trabajo se caracteriza por su tamaño y tiempo de procesamiento. El tiempo de procesamiento de un lote está determinado por el tiempo de procesamiento más grande de los trabajos dentro del lote. Ellos proponen un algoritmo genético híbrido con el objetivo de minimizar el makespan.

Leung et al.[26] consideran el problema de programación de n trabajos en m máquinas idénticas en paralelo, donde los trabajos son procesados en lotes y el tiempo de procesamiento de cada trabajo depende de su tiempo de espera, es decir, el tiempo al que inció el trabajo menos el tiempo en el que inició el lote. Si el trabajo comienza antes de que se cumpla un tiempo de espera límite dado, requiere de un tiempo de procesamiento básico, si el tiempo de espera sobrepasa el tiempo de espera límite, se requiere un tiempo de procesamiento extendido para llevar a cabo el trabajo. El objetivo es minimizar la suma de los tiempos de terminación de los trabajos.

En Manjeshwar et al.[31] se trabaja con un problema de programación en el cual los tiempo de procesamiento y los tamaños de los trabajos se conocen y no son idénticos. Se cuenta con dos máquinas. Cada máquina puede procesar un lote mientras no se exceda su capacidad. El tiempo de procesamiento del lote es el mayor tiempo de procesamiento de los trabajos que contiene. El objetivo es el de minimizar el makespan.

Tang y Liu[53] estudiaron el problema de programación en dos máquinas con lotes y tiempo de liberación. El objetivo es minimizar el makespan. Formulan el pro-

blema con un modelo de programación entera mixta y muestran que es un problema NP-duro. También desarrollan un algoritmo heurístico basado en programación dinámica para resolver el problema.

Zhang y Gu[59] proponen un método para modelar problemas de programación incluyendo lotes y tiempos de preparación dependientes de la secuencia anticipados mediante redes de Petri.

Wang y Chou[56] propusieron varias metaheurísticas para la solución del problema de programación de procesamiento de lotes en máquinas en paralelo en el cual cada máquina procesa simultáneamente varios trabajos en un lote, siempre y cuando el tamaño de todos los trabajos en el lote no exceda la capacidad de la máquina. El tiempo de procesamiento del lote es el tiempo de procesamiento del trabajo más largo dentro del lote.

Bayi Cheng et al.[5] consideran el problema de programación de lotes en máquinas paralelas con trabajos de tamaño arbitrario. Las máquinas tienen la misma capacidad de tamaño y velocidad de procesamiento. Los trabajos son procesados en lotes dado que el tamaño de los trabajos en el lote no exceda la capacidad de las máquinas. Una vez que el lote comienza a procesarse, no se puede interrumpir hasta que todos los trabajos terminen de procesarse. En este trabajo se muestra un modelo de programación entera mixta, seguido de la complejidad computacional del problema. Proponen un método de colonia de hormigas donde se usa el criterio de Metrópolis[55] para superar la convergencia inmadura.

Wang et al.[57] desarrollan un modelo de optimización integrado para la generación y producción de lotes sin violar las restricciones de procesos de producción en una empresa de energía intensiva. Utilizan técnicas de linealización para formular un modelo entero de programación lineal (MILP), mismo que se usa para dar solución al problema.

Noroozi et al.[41] presentan un problema de programación donde un grupo de trabajos puede ser procesado en una máquina simultáneamente. Desarrollaron tres

algoritmos efectivos para resolver el problema, incluyendo un algoritmo genético, y un recocido simulado híbridos.

Lozano y Madaglia[29] trabajaron en un problema de programación que consiste en programar trabajos en cierto número de máquinas en paralelo que procesan lotes, asignando los trabajos a los lotes y después se hace la secuenciación en las máquinas. Ellos proponen una heurística de dos fases que combina métodos exactos con búsquedas heurísticas con el objetivo de maximizar el uso de las máquinas y minimizar el retraso en la terminación de cada trabajo.

Jia y Leung[22] abordan el problema de programación de n trabajos de tamaño arbitrario en un conjunto de m máquinas idénticas en paralelo con el fin de minimizar el makespan. Proponen un algoritmo metaheurístico basado en sistema de hormiga máx-min para resolver el problema.

En esta sección se presentaron los trabajos más relevantes del problema de secuenciación de trabajos en lotes, se observó que en éstos no se permite la partición de lotes. Es por ello, que los modelos y métodos propuestos en la literatura no se ajustan al caso de estudio que se estudia en esta tesis. En el caso de estudio, los trabajos que conforman un mismo lote pueden ser procesados en diferentes máquinas, y el tiempo de terminación del lote es el tiempo de terminación del último trabajo procesado.

2.1.1 Programación de trabajos idénticos en lotes

Se dice que se tiene un lote de trabajos idénticos cuando éstos tienen el mismo tiempo de procesamiento. Esta es una característica específica del caso que se presenta en la planta cervecera. En Peter Brucker et al.[2] se estudia un problema de programación donde se establecen g grupos de trabajos en m máquinas que trabajan en paralelo. Cada grupo contiene trabajos idénticos. Se requiere particionar cada grupo en lotes, y asignar los lotes a las máquinas. Un lote puede ser procesado sólo en una máquina. Existe un tiempo de preparación independiente de la secuencia

antes de que un lote de un grupo sea procesado. Se asocia un tiempo límite a cada grupo de trabajos. El objetivo es encontrar una programación factible que cumpla con los tiempos límites de cada grupo de trabajos.

En el trabajo de Alessandro Condotta et al.[9] se estudian los problemas de programación de lotes en paralelo con capacidades de lote y trabajos de misma duración en un ambiente de máquinas paralelas. El objetivo es minimizar el máximo tiempo de espera(lateness). En este trabajo se proponen varios algoritmos heurísticos que probaron ser mejores que los algoritmos existentes en los últimos 20 años.

A. Costa[10] estudió el problema de programación de lotes para la industria farmacéutica desarrollando un algoritmo genético híbrido equipado con dos etapas de codificación. Cada lote se compone de trabajos idénticos. El tamaño del lote depende del tipo de trabajo y el tiempo de procesamiento del lote es igual al tiempo de procesamiento del tipo de trabajo que contiene. Tiempos de preparación dependientes de la secuencia y tiempos de remoción son requeridos cada vez que un lote va a ser procesado. Se tienen ventanas de tiempo fijas en las cuales no se pueden realizar preparaciones ni remociones. Existen varias restricciones en relación a las ventanas de tiempo, tiempos de preparación y remociones que tienen que respetarse.

Byung-Cheon y Kangbok Lee[8] estudiaron un flexible job shop scheduling problem de dos etapas, la primera etapa con una máquina y la segunda etapa con m máquinas idénticas en paralelo, donde los tiempos de procesamiento de cada trabajo en cada etapa es el mismo. El objetivo del problema es el de minimizar el makespan. En este trabajo muestran que la complejidad del problema es NP-duro cuando se fija m en la segunda etapa del proceso.

En Mehta y Uzsoy[33] se estudió el problema de minimizar la latencia total en una máquina procesadora de lotes con familias de trabajos incompatibles, donde todos los trabajos de la misma familia tienen tiempos de procesamiento idénticos y trabajos de diferentes familias no se pueden procesar juntos. Se presenta un algoritmo de programación dinámica el cual tiene complejidad de tiempo polinomial cuando el

número de familias de trabajos y la capacidad de lotes en las máquinas son fijos.

Mosheiov y Oron[40] estudian el problema de programación de lotes en una máquina para minimizar el tiempo total de flujo. Los tiempos de procesamiento se asumen idénticos para todos los trabajos. El tamaño de los lotes pueden estar acotados. Una solución óptima consiste en el número y tamaño de lotes a procesar.

Mosheiov y Oron[39] también estudiaron el problema de programación de lotes en m-máquinas para minimizar el makespan y el tiempo total de flujo. Se asumen tiempos de procesamiento de trabajos idénticos, tiempos de preparación independientes de la secuencia, y disponibilidad de lotes. Proponen un algoritmo heurístico para la minimización del tiempo de flujo.

2.2 Programación de familias de trabajos

En Shen y Buscher[48] se aborda el problema de programación de lotes en serie para minimizar el makespan. Se toman en cuenta tiempos de preparación dependientes de la secuencia de familias de trabajos y disponibilidad de trabajos. Proponen un algoritmo de búsqueda Tabú con varios vecindarios y múltiples listas tabú.

En Udo Buscher y Liji Shen[3] se aborda el problema de programación en máquinas paralelas, involucrando tiempos de preparación para las familias de trabajos. Se propone una búsqueda Tabú con varios vecindarios con el objetivo de minimizar el makespan. El tiempo de preparación se aplica cuando se cambia la familia de los trabajos a procesar en las máquinas. Después, Udo Buscher y Liji Shen[4] proponen heurísticas basadas en MIP para balancear la calidad de la solución con el tiempo de cómputo.

En H.A.J. Crauwels[11] se aborda un problema de programación en máquinas paralelas para la planeación de materiales requeridos por periodo (cada semana por ejemplo), donde los trabajos se particionan en familias y se requiere un tiempo de preparación asociado para cada familia al inicio de cada periodo y de cada lote.

Cada lote es un máximo conjunto de trabajos de la misma familia, que son procesados continuamente. Se utiliza una formulación de programación entera para minimizar el número de periodos sobrecargados y el tiempo de sobrecarga. Después, se aplica una heurísticas para aligerar la carga de los periodos sobrecargados. Un problema similar es abordado en Li y Yuan[27] en el que se estudia la programación de n familias de trabajos con fechas de salida en m máquinas idénticas en paralelo. Cada máquina tiene un límite de trabajos a procesar al mismo tiempo como lote. Trabajos de diferentes familias no se pueden procesar en el mismo lote. El objetivo es minimizar el makespan.

Minimizar la máxima latencia en el problema de programación de lotes en máquinas idénticas en paralelo con llegadas de trabajos dinámicas fue el tema que se estudió en Malve y Uzsoy[30]. En su trabajo proponen una familia de mejoras heurísticas iterativas combinadas con un algoritmo genético, en este trabajo se maneja incompatibilidad de trabajos de diferentes familias. Un trabajo similar es el de Mathiraja et al.[32], en el que se estudia el problema de programación con máquinas no idénticas en paralelo con la presencia de llegadas de trabajos dinámicas, familias de trabajos incompatibles y tamaños de trabajo no idénticos. El objetivo es maximizar el uso de las máquinas.

2.2.1 Programación de familias de trabajos idénticos

En esta sección se presentan los problemas que más similitudes tienen con el problema estudiado en la compañía cervecera, que son los lotes de trabajos de la misma familia, siendo que cada trabajo de la misma familia tiene tiempos de procesamiento idénticos. Un ejemplo es el estudio realizado por Shen et al.[51], en donde estudian el problema de programación de familias de trabajos en máquinas en paralelo para minimizar el tiempo ponderado total de terminación. Existen tiempos de preparación cuando se procesan trabajos de diferentes familias y dependientes de la secuencia. En su trabajo se enfoncan en el caso especial donde todos los trabajos de la misma familia tienen tiempos de procesamiento idénticos y se pueden formar

lotes de trabajos de la misma familia.

En Shen et al.[50] se considera un problema de programación en máquinas en paralelo para minimizar el tiempo total de terminación. Cada trabajo pertenece a una familia. Existen tiempos de preparación dependientes de la secuencia y se asume disponibilidad de lotes y lotes en serie. Otro trabajo similar es Shen et al.[49] en el que se trata el problema de programación y formación de lotes donde se presentan tiempos de preparación dependientes de la secuencia de familias. El objetivo es minimizar el makespan. Ellos dividen las familias de productos en lotes.

La programación de la producción se puede definir como la distribución de los recursos de producción a lo largo del tiempo para satisfacer algún criterio de la mejor manera tal y como lo describe Stephen C. Graves[20]. Con el fin de facilitar el entendimiento de los problemas de programación de trabajos, Graham et al.[19] proponen la nomenclatura común con la que se hace referencia a los problemas de programación con máquinas en paralelo. Mientras que en Méndez et al.[36] se hace una clasificación de los problemas de programación de lotes y los modelos de optimización correspondientes. Comparan la efectividad y la eficiencia de los modelos y se discuten problemas reales que no encajan tan fácilmente en los métodos existentes.

En cuanto a la complejidad de este tipo de problemas M.R. Garey et al.[17] muestran que el flow shop scheduling problem con m-máquinas es NP-duro para todo $m \geq 2$. En Rinnooy Kan[45] se puede encontrar una explicación más a fondo en cuanto a la complejidad de los problemas más comunes de programación de trabajos.

La complejidad de los problemas de programación de lotes con tiempo de preparación se presenta en Monma y Potts[37]. Demostrando que el problema es NP-duro incluso para el caso donde se tienen dos máquinas idénticas y tiempos de preparación independientes de la secuencia.

Para mayor información acerca de los algoritmos básicos propuestos para la solución de problemas de programación por lotes se sugiere ver el trabajo realizado

por Potts y Kovalyov[43].

En el caso de estudio que se presenta en esta tesis existe una característica que no se encontró en la literatura, y es que cada trabajo produce una cantidad distinta de producto dependiendo de la máquina en la que fue procesado. En otras palabras, la cantidad de trabajos a procesar no es fija, y para una misma instancia puede variar la cantidad de trabajos procesados para cubrir la demanda, ésto depende directamente de la secuencia y la repartición de trabajos en las máquinas. El algoritmo propuesto en esta tesis toma en cuenta esta características y minimiza el tiempo para cubrir la demanda de cada familia de trabajos, por lo que es uno de los principales aportes de este trabajo.

Capítulo 3

Problema de secuenciación de trabajos en sistemas flexibles

En este capítulo se presenta una descripción formal del FJSP (*Flexible Job Shop Scheduling Problem*) así como una formulación matemática propuesta por Vahid Roshanaei[47]. También se describe de manera detallada el algoritmo de optimización propuesto para resolverlo, el cual está basado en un esquema de tipo ALNS que interactúa con el *Branch-and-Bound* de CPLEX. Se muestran los resultados obtenidos y se analiza el desempeño del algoritmo sobre varios conjuntos de instancias tomadas de la literatura.

3.1 Descripción del Problema

En el FJSP se tiene un conjunto J de trabajos, cada trabajo $j \in J$ está formado por un conjunto de operaciones N_j, la operación $l \in N_j$ del trabajo $j \in J$ se representa por $O_{j,l}$. Se cuenta con un conjunto de máquinas M en paralelo. Cada operación $O_{j,l}$ puede ser procesada en alguna máquina dentro de un conjunto de máquinas $R_{j,l}$ que son capaces de procesarla. El tiempo de procesamiento de la operación $O_{j,l}$ en la máquina $i \in R_{j,l}$ se representa por $p_{j,l,i}$. El tiempo de terminación de operación $O_{j,l}$ está dado por $C_{j,l}$. El FJSP puede estudiarse como un problema divido en dos subproblemas: el problema de asignar cada trabajo a alguna máquina capaz de procesarlo; y el problema de ordenar los trabajos dentro de las máquinas

a las que se asignaron, tal como se describe en la sección 2.0.1, en donde se da una introducción al problema, el cual se presenta en gran medida en el mundo industrial hoy en día.

3.1.1 Modelo matemático

El modelo matemático utilizado para resolver parcialmente el FJSP está basado en los tiempos de terminación de trabajos y tiene por objetivo minimizar el makespan, fue propuesto por Vahid Roshanaei[47]. La variable de decisión $Y_{j,l,i}$, toma valor de 1 cuando se asigna la operación $l \in N_j$ del trabajo $j \in J$ a la máquina $i \in R_{l,j}$, toma valor de 0 de otra manera. La variable X_{jlhz} toma valor de 1 cuando la operación O_{jl} es procesada despues de la operación O_{hz}, toma valor de 0 de otra manera.

$$\text{Min} \quad z = C_{max} \tag{3.1}$$

sujeto a:

$$\sum_{i \in R_{jl}} Y_{jli} = 1 \qquad \forall j \in J, l \in N_j \tag{3.2}$$

$$C_{jl} \geq C_{j,l-1} + \sum_{i \in R_{jl}} Y_{jli} p_{jli} \qquad \forall j, l \tag{3.3}$$

$$C_{jl} \geq C_{hz} + p_{jli} - M\left(3 - X_{jlhz} - Y_{jli} - Y_{hzi}\right) \qquad \forall j < n, l; h > j, z; \; i \in R_{jl} \cap R_{hz} \tag{3.4}$$

$$C_{hz} \geq C_{jl} + p_{hzi} - M\left(X_{jlhz} + 2 - Y_{jli} - Y_{hzi}\right) \qquad \forall j < n, l; h > j, z; \; i \in R_{jl} \cap R_{hz} \tag{3.5}$$

$$C_{max} \geq C_{jn_l} \qquad \forall j \in J \tag{3.6}$$

$$C_{jl} \geq 0 \qquad \forall j \in J, l \in N_j \tag{3.7}$$

$$C_{j0} = 0 \qquad \forall j \in J \tag{3.8}$$

La restricción (3.2) asegura que cada operación de cada trabajo se asigne a una sola máquina, siendo la variable Y_{jli} igual a 1 cuando la operación l del trabajo j (O_{jl}) es asignada a la máquina i dentro de un conjunto de máquinas R_{jl} capaces de procesar la operación, Y_{jli} toma valor 0 de otra forma. El tiempo en que termina la operación O_{jl} se representa por la variable C_{jl}. La restricción (3.3) restringe el valor de C_{jl} tomando en cuenta el tiempo de terminación de la operación previa ($C_{j,l-1}$) y el tiempo de procesamiento de la operación O_{jl} en la máquina i a la que se asignó (p_{jli}). La secuenciación de las operaciones dentro de las máquinas está determinada por las restricciones

(3.4) y (3.5), en donde se utiliza un número grande M, y una variable binaria X_{jlhz} que toma valor de 1 cuando la operación O_{jl} es procesada después de la O_{hz}, activando las restricciones (3.5), X_{jlhz} toma valor de 0 cuando la operación O_{jl} no es procesada después de la operación O_{jl}, activando las restricciones (3.4). Las restricciones (3.6) se usan para relacionar el tiempo de terminación de la última operación procesada con la función objetivo. Los tiempos de terminación de las operaciones están restringidos a valores positivos por las restricciones (3.7). La inicialización de los tiempos de terminación de la operación 0 se realiza en las restricciones (3.8).

3.2 Metodología de solución

Debido a que actualmente no existen métodos heurísticos eficientes para la solución del FJSP, en esta tesis se propone un algoritmo metaheurístico tipo ALNS (Adaptive Large Neighborhood Search), usando el *Branch-and-Bound* de Cplex entre iteraciones del ALNS para resolver parcialmente el problema y de esta manera aproximarnos a la solución óptima. El ALNS permite una fácil manipulación de las soluciones ya que cuenta con un número de sub-heurísticas, las cuales son usadas con una frecuencia que depende del desempeño que éstan han tenido durante las iteraciones previas del ALNS (Ropke and Pisinger[46]).

La creación de soluciones iniciales se realiza de manera aleatoria, es decir, se selecciona un trabajo a la vez y se asigna cada una de las operaciones del trabajo, en la secuencia correspondiente, en alguna máquina que sea capaz de procesarla, la cual es seleccionada de manera aleatoria. Posteriormente, estas soluciones son mejoradas mediante una serie de operadores de destrucción y reparación (ver Algoritmo 1).

3.2.1 Algoritmo ALNS

Algoritmo 1 Algoritmo principal ALNS

Inicializar todos los pesos de destrucción y reparación igual a 1 y sus puntuaciones igual a 0.
$s* \leftarrow s \leftarrow$ solución inicial
$\tau \leftarrow \tau_{inicio}$
mientras No se alcance el número de iteraciones máximas **hacer**
    $s' \leftarrow s$
    Seleccionar un operador de destrucción basado en los pesos actuales
    Seleccionar un operador de reparación basado en los pesos actuales
    Aplicar operadores a s' y actualizar el numero de veces utilizados
    si Se cumple criterio para compactar solución **entonces**
        $Compactar(s')$
    fin si
    si $z(s) < z(s')$ **entonces**
        $s' \leftarrow s$
        si $z(s) < z(s*)$ **entonces**
            $s* \leftarrow s$
            Recompensar operadores con puntaje alto
            Tomar $s*$ como solución inicial y resolver el problema con el Branch and Bound de Cplex durante 100 segundos
            si Solución de Cplex es mejor a $s*$ **entonces**
                $s* \leftarrow SolucionCplex$
            fin si
        si no
            Recompensar operadores con puntaje medio
        fin si
    si no, **si** s' es aceptada por el criterio de recocido simulado **entonces**
        $s \leftarrow s'$
        Recompensar operadores con puntaje bajo
    fin si
    si Número de iteraciones es múltiplo de φ **entonces**
        Tomar $s*$ como solución inicial y resolver el problema con el Branch and Bound de Cplex durante 100 segundos
        si Solucion de Cplex es mejor que $s*$ **entonces**
            $s* \leftarrow SolucionCplex$
        fin si
        Actualizar los pesos de operadores y reiniciar puntuaciones
    fin si
    $\tau = \tau * (0.01/\tau)^{1/(niter)}$
fin mientras

Regresar $s*$

El algoritmo se inicia usando una temperatura inicial $\tau_{inicio} = 20000$ cuya tasa de enfriamiento depende del número de iteraciones que se quiera correr el algoritmo $niter$. Los pesos de los movimientos de destrucción y reparación se inicializan en 1. En esta implementación el valor de φ fue de 100, y se introduce un factor de reacción de 0.7, es decir, los nuevos pesos calculados se compondrán del 70 % de su desempeño en las 100 últimas iteraciones y del 30 % de su peso anterior. Las puntuaciones se inicializan en 0 y se actualizan con valores de recompensa de 10, 5 y 2 para el puntaje alto, medio y bajo respectivamente. La función $z(s)$ devuelve el valor del makespan de la solución s.

Debido a que en algunos casos, al realizar los operadores de destrucción y reparación, se acomodan dos operaciones consecutivas del mismo trabajo de manera que existe un tiempo muerto entre éstas, se creó función $Compactar(s)$ que acelera el tiempo de inicio de las operaciones siempre y cuando sea posible, con el fin de reducir el tiempo muerto de las máquinas.

A continuación se explican los movimientos implementados para el algoritmo.

3.2.2 Operadores de destrucción

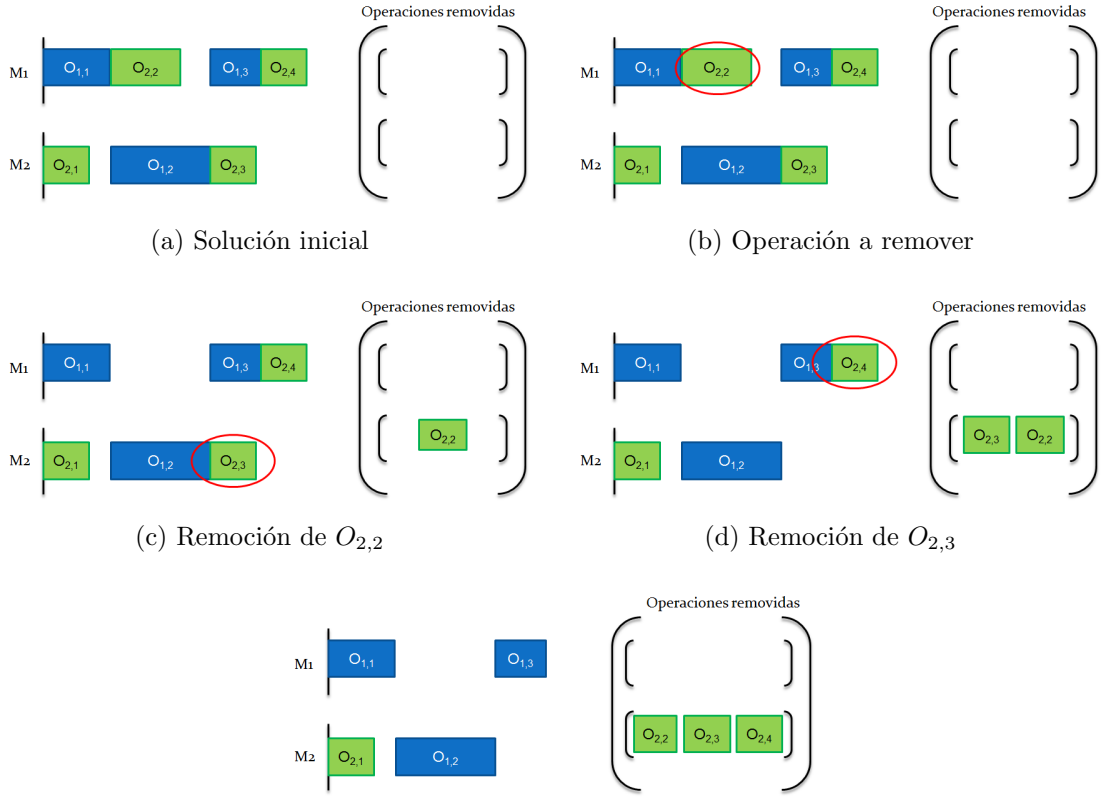
Para la solución del FJSP se propusieron un total de 8 operadores de destrucción que serán descritos en esta sección, y los cuales fueron implementados en el Algoritmo 1. Algunos usan una lista restringida de candidatos para procurar que exista cierta diversidad en esta etapa del algoritmo, para determinar el tamaño de la lista se utiliza un factor δ que determina el porcentaje de máquinas totales que entrarán a la lista. Las operaciones removidas se van guardando en un conjunto de subconjuntos de manera que cada subconjunto contiene únicamente operaciones de un mismo trabajo, las operaciones además se guardan en orden ascendente, lo que le facilitará que la reparación evite caer en infactibilidad por secuencia de operaciones de un mismo trabajo.

Operadores de destrucción masiva

Este tipo de operadores seleccionan una operación O_{jl} asignada a una máquina y la remueven junto con todas las operaciones del mismo trabajo que dependan (sean antecedidas) por dicha operación, esto con el fin de evitar caer en infactibilidad.

Para un mejor entendimiento de este tipo de operadores se muestra en la Figura 3.1a un ejemplo de una solución inicial factible donde se tienen dos trabajos y dos máquinas en las que están distribuidas las operaciones de cada trabajo, se muestra también el conjunto de subconjuntos

de operaciones removidas vacío. Se selecciona la operación $O_{2,2}$ a remover de acuerdo al criterio del movimiento de destrucción masiva utilizado, tal como se muestra en la Figura 3.1b. En la Figura 3.1c se muestra la solución después de la remoción de la operación seleccionada, la operación removida entra al subconjunto de operaciones removidas (en este caso, se tiene la operación 2 del trabajo 2, $O_{2,2}$); como puede observarse, la operación $O_{2,3}$ depende de la operación removida ($O_{2,2}$), por lo que se selecciona para su remoción. La Figura 3.1d muestra que al remover la operación $O_{2,3}$ es necesario seleccionar la operación $O_{2,4}$, ya que esta es precedida por la operación removida $O_{2,3}$. Por último se muestra en la Figura 3.1e la solución al remover la última operación que dependía de las operaciones removidas.



(a) Solución inicial

(b) Operación a remover

(c) Remoción de $O_{2,2}$

(d) Remoción de $O_{2,3}$

(e) Remoción de $O_{2,4}$

Figura 3.1: Movimiento de destrucción masiva

Remoción de operación aleatoria de máquina aleatoria: Se selecciona una máquina al azar del conjunto de máquinas que tienen por lo menos una operación asignada, posteriormente se selecciona al azar una operación que esté actualmente asignada a dicha máquina y se remueve junto con todas las operaciones del mismo trabajo que dependen de la operación removida.

Remoción de operación aleatoria en una de las máquinas con mayor makespan: Se selecciona de manera aleatoria una de las máquinas dentro de un conjunto de máquinas con mayor makespan, conjunto cuyo tamaño está dado por δ. De la máquina seleccionada se remueve una operación de manera aleatoria junto con todas las operaciones del mismo trabajo que dependen de la operación removida.

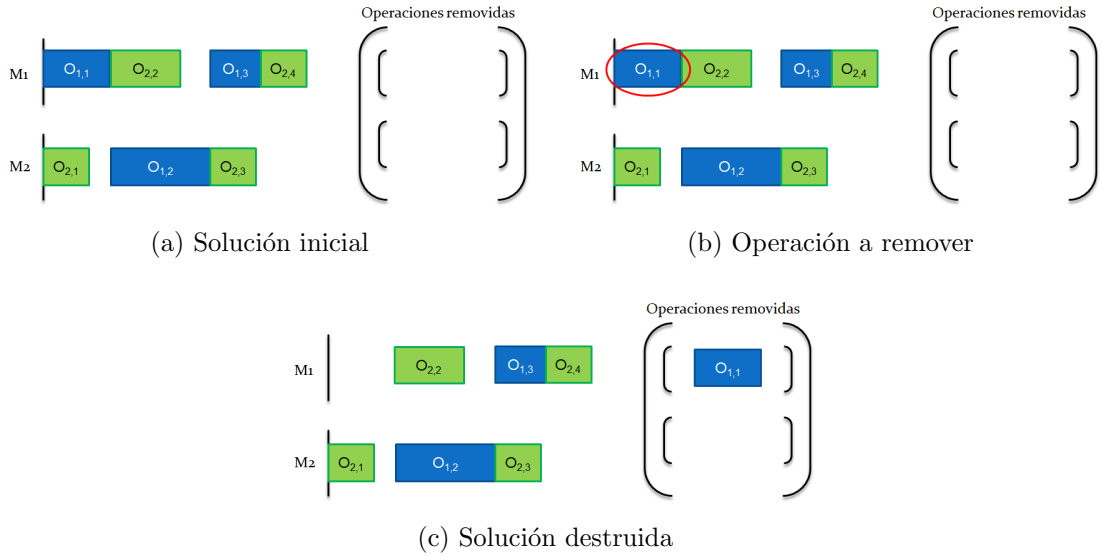
Remoción de operación aleatoria en una de las máquinas con mayor tiempo de ocupación: El tiempo de ocupación de una máquina se define como la suma del tiempo de procesamiento de las operaciones asignadas a la máquina. Se selecciona de manera aleatoria una de las máquinas dentro de un conjunto de máquinas con mayor tiempo de ocupación, conjunto cuyo tamaño está dado por δ. De la máquina seleccionada se remueve una operación de manera aleatoria junto con todas las operaciones del mismo trabajo que dependen de la operación removida.

Remoción de operación aleatoria en una de las máquinas con mayor tiempo de ocio: El tiempo de ocio de una máquina se define como la suma de los tiempos muertos entre las operaciones asignadas a la máquina. Se selecciona de manera aleatoria una de las máquinas dentro de un conjunto de máquinas con mayor tiempo de ocio, conjunto cuyo tamaño está dado por δ. De la máquina seleccionada se remueve una operación de manera aleatoria junto con todas las operaciones del mismo trabajo que dependen de la operación removida.

Operadores de destrucción simple

Este tipo de operadores de destrucción remueven únicamente la operación seleccionada, pero al momento de reparar la solución el algoritmo verifica la factibilidad de la siguiente operación del mismo trabajo, y en caso de que la operación siguiente se volviese infactible, ésta es removida de la solución para su posterior reparación.

En la Figura 3.2a se presenta un ejemplo de una solución inicial factible a la cual se le aplicará un movimiento de destrucción simple. Como puede observarse, se tienen dos trabajos y dos máquinas en las que están distribuidas las operaciones de cada trabajo y se muestra también el conjunto de subconjuntos de operaciones removidas vacío. En la Figura 3.2b se selecciona la operación $O_{1,1}$ a remover de acuerdo al criterio del movimiento de destrucción simple utilizado. La solución parcialmente destruida se muestra en la Figura 3.2c. Esta solución será posteriormente reconstruida por alguno de los operadores de reparación.



(a) Solución inicial

(b) Operación a remover

(c) Solución destruida

Figura 3.2: Movimiento de destrucción simple

Remoción de operación aleatoria de máquina aleatoria: Se selecciona una máquina al azar del conjunto de máquinas que tienen por lo menos una operación asignada, selecciona una operación que esté actualmente asignada a dicha máquina y la remueve.

Remoción de operación aleatoria en una de las máquinas con mayor makespan: Se selecciona de manera aleatoria una de las máquinas dentro de un conjunto de máquinas con mayor makespan, conjunto cuyo tamaño está dado por δ. De la máquina seleccionada se remueve una operación de manera aleatoria.

Remoción de operación aleatoria en una de las máquinas con mayor tiempo de ocupación: Se selecciona de manera aleatoria una de las máquinas dentro de un conjunto de máquinas con mayor tiempo de ocupación, conjunto cuyo tamaño está dado por δ. De la máquina seleccionada se remueve una operación de manera aleatoria.

Remoción de operación aleatoria en una de las máquinas con mayor tiempo de ocio: Se selecciona de manera aleatoria una de las máquinas dentro de un conjunto de máquinas con mayor tiempo de ocio, conjunto cuyo tamaño está dado por δ. De la máquina seleccionada se remueve una operación de manera aleatoria.

3.2.3 Operadores de reparación

Para la implementación del Algoritmo 1 se utilizaron 8 operadores de reparación, algunos usan una lista restringida de candidatos para procurar que exista cierta diversidad en esta etapa del algoritmo. Para determinar el tamaño de la lista se utiliza el mismo factor δ usado en los operadores de destrucción, éste determina el porcentaje de máquinas capaces de procesar la operación a insertar. Ya que las operaciones removidas están separadas en conjuntos ordenados, se facilita la reparación de soluciones factibles.

Inserción aleatoria: Se selecciona un conjunto de operaciones removidas de manera aleatoria y se elige la primera operación de dicho conjunto. Del conjunto de máquinas capaces de procesar la operación, se determina al azar la máquina a la cual se asignará la operación. Se inserta la operación en la máquina seleccionada en una posición tomada aleatoriamente de un conjunto de posibles posiciones en las que puede ser insertada.

Inserción en máquinas con mayor tiempo de ocio, primera posición: Se selecciona un conjunto de operaciones removidas de manera aleatoria y se elige la primera operación de dicho conjunto. Del conjunto de máquinas capaces de procesar la operación, se hace una lista restringida de máquinas que pueden realizar la operación y que tengan más tiempo de ocio, se elige una máquina de la lista al azar, a la cual se asignará la operación. Se inserta la operación en la máquina seleccionada en la primera posición posible.

Inserción en máquinas con menor tiempo de terminación, primera posición: Se selecciona un conjunto de operaciones removidas de manera aleatoria y se elige la primera operación de dicho conjunto. Del conjunto de máquinas capaces de procesar la operación, se hace una lista restringida de máquinas que pueden realizar la operación y que tengan menor tiempo de terminación, se elige una máquina de la lista al azar, y se inserta la operación en la primera posición posible en la máquina seleccionada.

Inserción en máquinas con menor tiempo de procesamiento, primera posición: Se selecciona un conjunto de operaciones removidas de manera aleatoria y se elige la primera operación de dicho conjunto. Del conjunto de máquinas capaces de procesar la operación, se hace una lista restringida de máquinas que pueden realizar la operación y que tengan menor tiempo de procesamiento para la operación, se elige una máquina de la lista al azar, a la cual se asignará la operación. Se inserta la operación en la máquina seleccionada en la primera posición posible.

Inserción en máquina aleatoria, primera posición: Se selecciona un conjunto de operaciones removidas de manera aleatoria y se elige la primera operación de dicho conjunto. Del conjunto de máquinas capaces de procesar la operación, se determina al azar la máquina a la cual se asignará la operación. Se inserta la operación en la máquina seleccionada en la posición más temprana posible.

Inserción en máquinas con mayor tiempo de ocio, posición aleatoria: Se selecciona un conjunto de operaciones removidas de manera aleatoria y se elige la primera operación de dicho conjunto. Del conjunto de máquinas capaces de procesar la operación, se hace una lista restringida de máquinas que pueden realizar la operación y que tengan más tiempo de ocio, se elige una máquina de la lista al azar, a la cual se asignará la operación. Se inserta la operación en la máquina seleccionada en una posición tomada aleatoriamente de un conjunto de posibles posiciones en las que puede ser insertada.

Inserción en máquinas con menor tiempo de terminación, posición aleatoria: Se selecciona un conjunto de operaciones removidas de manera aleatoria y se elige la primera operación de dicho conjunto. Del conjunto de máquinas capaces de procesar la operación, se hace una lista restringida de máquinas que pueden realizar la operación y que tengan menor tiempo de terminación, se elige una máquina de la lista al azar, a la cual se asignará la operación. Se inserta la operación en la máquina seleccionada en una posición tomada aleatoriamente de un conjunto de posibles posiciones en las que puede ser insertada.

Inserción en máquinas con menor tiempo de procesamiento, posición aleatoria: Se selecciona un conjunto de operaciones removidas de manera aleatoria y se elige la primera operación de dicho conjunto. Del conjunto de máquinas capaces de procesar la operación, se hace una lista restringida de máquinas que pueden realizar la operación y que tengan menor tiempo de procesamiento para la operación, se elige una máquina de la lista al azar, a la cual se asignará la operación. Se inserta la operación en la máquina seleccionada en una posición tomada aleatoriamente de un conjunto de posibles posiciones en las que puede ser insertada.

3.3 Experimentación computacional y resultados

La experimentación computacional se realizó en una Workstation HP Z620 con procesador Intel Xeon(R) CPU E5-2620 v2 a 2.10 GHz y con memoria RAM de 64 Gb, bajo el sistema operativo Ubuntu 14.04 LTS. El ALNS propuesto se codificó en c++ haciendo uso de las librerías de CPLEX. Se utilizó el Branch-and-Bound de CPLEX 12.6 para resolver parcialmente el FJSP en algunas iteraciones del ALNS.

Las instancias con las que se probó el Algoritmo 1 fueron tomadas de Quintiq[44], están divididas en grupos, dentro de cada grupo, las instancias varían en cuanto al número de trabajos n, número de máquinas m, tiempos de procesamiento, y capacidad de máquinas para realizar las operaciones de los trabajos. Se probaron todos los grupos de instancias con los parámetros descritos en la Sección 3.2 con un número de iteraciones de 1000, dejando que Cplex trabajase durante 100 segundos, la Tabla 3.1 muestra los resultados de manera condensada, donde se muestra a grandes rasgos la cantidad de soluciones óptimas conocidas en la literatura, por grupo de instancias, y la cantidad de soluciones óptimas que se encontraron con el ALNS propuesto. Como puede observarse, el ALNS híbrido fue capaz de alcanzar más de la mitad de las soluciones óptimas conocidas.

Tabla 3.1: Comparación de cantidad de soluciones óptimas encontradas por el ALNS para cada grupo de instancias.

Clase de instancia	Número de instancias	Soluciones óptimas conocidas	Óptimos alcanzados
Barnes	21	21	18
Brandimarte	10	9	4
Dauzere	18	6	0
Hurink_edata	66	64	45
Hurink_rdata	66	52	20
Hurink_sdata	66	65	51
Hurink_vdata	66	63	22

En la Tabla 3.2 se muestra el porcentaje de gap promedio para cada grupo de instancias, tomando en cuenta que para las instancias que no han sido resueltas a optimalidad, el gap se calcula respecto a la mejor cota superior conocida actualmente. Las mejores cotas encontradas hasta la fecha están reportadas en Quintiq[44].

Tabla 3.2: Gap (%) promedio con respecto a las mejores soluciones conocidas.

Clase de instancia	Gap (%)
Barnes	0.1
Brandimarte	15.5
Dauzere	29.4
Hurink_edata	1.8
Hurink_rdata	3.8
Hurink_sdata	0.7
Hurink_vdata	16.1

3.3.1 Análisis de resultados

En los resultados mostrados en la Tabla 3.1 es posible notar la cantidad de óptimos alcanzados por el algoritmo propuesto(Algoritmo 1). Cabe resaltar que el ALNS propuesto fue capaz de resolver a optimalidad más de la mitad de las instancias en las que se conoce el óptimo en la literatura, lo que indica el buen desempeño del algoritmo propuesto.

En la Tabla 3.2 resalta el hecho de que en las instancias del grupo Dauzere se tiene un gap promedio del 29.4 %, y en las instancias de los grupos Brandimarte y Hurink$_v$data se tienen gaps promedio de 15.5 % y 16.1 %, son gaps promedio un poco grandes, sin embargo en las instancias de los grupos Barnes, Hurink$_e$data, Hurink$_r$data, y Hurink$_s$data se tienen gaps promedio muy bajos. Esto último demuestra que el algoritmo en sí es bueno para la resolución del FJSP, y una posible razón para que los resultados no fueran los esperados en todas las instancias es que los parámetros utilizados para la experimentación presentada parecen favorables para unos grupos de instancias y no tan favorables para otros grupos.

Capítulo 4

Caso de estudio

En este capítulo se presenta un problema real de planeación de producción en una empresa cervecera de la localidad. Se describe un procedimiento tipo GRASP para dar solución al problema y el desempeño del mismo se evalúa sobre instancias reales y aleatorias que han sido generadas a partir de las reales. Los resultados muestran la efectividad del algoritmo propuesto, se observa un ahorro de hasta 28 % con respecto al tiempo requerido por la planeación realizada por el tomador de decisiones en la empresa. Cabe señalar que los tiempos de cómputo del algoritmo son despreciables en comparación con el tiempo que toma el tomador de decisiones en la empresa.

4.1 Descripción del caso de estudio

El proceso de elaboración de cerveza consta de 3 etapas principales: cocimiento, fermentación y reposo.

En la etapa de cocimiento se preparan las mezclas de granos, dichas mezclas son llamadas mostos, las cuales se envían posteriormente a la etapa de fermentación. Se tiene un conjunto de W de mostos. Cada mosto $w \in W$ puede pertenecer a una familia $f_w \in F$. En el área de cocimiento se cuenta con 2 líneas de cocimiento las cuales trabajan en paralelo. Se considera cada casa de cocimiento como una máquina dentro de un conjunto de máquinas H. Cada mosto w puede ser asignado a alguna de las máquinas dentro de un conjunto H_w de máquinas capaces de procesarlo. Se tiene un conjunto D de demandas a cubrir por tipo de mosto. La demanda a cubrir de mosto w está dada por d_w. Las casas de cocimiento pueden trabajar a 2 ritmos de producción distintos, denominados ritmo normal, y ritmo lento. El ritmo al que se trabaje dependerá de la demanda a cubrir y las condiciones de la planta, el ritmo determina el tiempo que existe entre la entrada de un mosto y la entrada del siguiente mosto a una casa de cocimiento. La unidad de producción en la etapa de cocimiento, se le conoce como cocimiento, es decir, producir un lote de mosto w se produce

una cantidad x de cocimientos, que conformarán el lote a procesar. La cantidad de cocimientos del lote está determinada por la capacidad del tanque de fermentación al que es asignado. Se tiene un arreglo A_h donde se guardan los tiempos de disponibilidad de cada cada h. El tiempo al que la casa $h \in H$ está disponible para asignar otro cocimiento se denota por $a_h \in A_h$.

En cada línea, cada vez que se cambia el tipo de mosto $w \in W$ a mosto $g \in W \setminus w$ (tipo de mezcla de materias primas) se realiza un proceso de lavado $s_{w,g}$, cuya duración depende de el tipo de mosto que se estaba procesando y el tipo de mosto que está por procesarse, y también depende de las características de cada casa de cocimiento. Además, cada que se alimentan 4 cocimientos, aunque el siguiente cocimiento sea del mismo tipo de mosto, se efectúa un lavado corto, ésto con el fin de asegurar la efectividad de la transferencia de calor en la olla. El efecto que tienen éstos lavados en la producción es el retraso en la entrada del siguiente cocimiento en la línea, y éste a su vez afecta el tiempo de terminación de la producción total requerida.

Aproximadamente cada semana se hace un paro para dar mantenimiento y lavar equipos, dura por lo general 12 horas en cada línea, aunque hay ocasiones en que puede llegar a tardar hasta 36 horas. Al momento de hacer la planeación de la producción se programan estos mantenimientos con una duración de 18 horas en cada tren de cocimiento. La línea 1 de cocimiento cuenta con dos trenes de cocimiento, lo que físicamente quiere decir que tiene dos cocedores (la etapa de cocimiento requiere cocedores, maceradores, y otras subetapas, pero sólo se cuenta con un tanque para cada etapa, a excepción de los cocedores), éstos trabajan intercalados (se carga un cocimiento mediante un tren de carga y el siguiente cocimiento con el otro tren y así sucesivamente), esta característica hace que los lavados cortos no afecten el ritmo de producción en esta línea, ya que mientras se lava un tren de carga, se utiliza el otro para procesar el cocimiento. La línea 2 cuenta únicamente con un tren de cocimiento. Se tiene un conjunto K de levaduras, las cuales se usan para que comiencen la etapa de fermentación, y la levadura usada depende del tipo de mosto a procesar. Se asume que se usa una unidad de levadura por lote.

En la etapa de fermentación se cuenta con un conjunto T de tanques y un arreglo A_t que guarda los tiempos de disponibilidad de cada tanque.Todo tanque $t \in T$ cuenta con una capacidad Q_t, que es la cantidad de cocimientos que puede fermentar. Un lote comienza su fermentación una vez que todos los cocimientos que lo componen están en el tanque de fermentación asignado. Una vez que un lote cumple su edad de fermentación está listo para pasar a la etapa de reposo. Para fines prácticos suponemos una capacidad ilimitada en el área de reposo. El tanque de fermentación t estará disponible para la asignación de un lote una vez esté completamente vacío, este tiempo de disponibilidad de tanque t se le denota como $a_t \in A_t$. Se cuenta con una línea única de comunicación entre estas dos etapas, lo que quiere decir que sólo se puede transferir un lote a la vez de fermentación a reposo.

El problema estudiado en la empresa tiene características similares a las de un FJSP, ya que una forma de ver los cocimientos de cada lote es tratarlos como trabajos idénticos (por ser del mismo tipo de mosto) que pueden ser procesados en alguna máquina (casa de cocimiento) que sea capaz de procesarla, ya que no todas las casas pueden procesar todos los tipos de mostos. El tamaño de los lotes lo determina la capacidad del tanque al que se asigna el trabajo. Sin embargo, la cantidad de lotes a producir para satisfacer la demanda depende de la secuencia en que se procesen los lotes. Esto debido a que cada trabajo produce una cantidad distinta de producto dependiendo de la máquina a la que es asignado, y la repartición de trabajos en las máquinas depende de la disponibilidad de las máquinas. La programación de la producción es un proceso que se hace tomando en cuenta las proyecciones de la demanda para las próximas 4 semanas, haciendo un corte cada jueves para monitorear la cantidad de cerveza que ya ha sido procesada y se encuentra en el inventario, la cantidad de cerveza que estará lista para su salida de la planta para cuando se requiera y la cantidad que se debe producir al final de las 4 semanas proyectadas. Una diferencia notable entre el caso de estudio y los FJSP comunes es que en este caso no se cuenta con una cantidad de trabajos fija a procesar para cubrir la demanda, sino que cada trabajo produce una cantidad específica de producto final que depende de la máquina a la que fue asignado. Es debido a las características tan específicas del proceso de elaboración de cerveza, que se optó desarrollar e implementar una heurística que permita incluir de manera sencilla todas las restricciones que se encuentran en este problema.

4.2 Metodología de solución

La metodología propuesta está basada en la metaheurística conocida como GRASP (Greedy Randomized Adaptive Search Procedure). El GRASP es una meta-heurística multi-arranque, que consta principalmente de dos fases, una fase constructiva en donde se genera una solución usando una función voraz aleatorizada; la segunda fase emplea un procedimiento de búsqueda local a la solución generada esperando incrementar la calidad de la solución, tal como lo mencionan Resende y Feo [16].

La fase constructiva del algoritmo propuesto crea una solución inicial en base a la demanda a curbir, y la fase de mejora, implementa intercambios de lotes con el fin de reducir los tiempos de preparación en la solución (ver Algoritmo 2).

4.2.1 Fase de construcción

Esta fase consiste en construir una solución inicial, la cual es una programación de secuencia de lotes, que pasan a través de la etapa de cocimiento y fermentación, esto para satisfacer la demanda de cada tipo de mosto. Cada lote se compone de varios cocimientos (operaciones) que pueden ser procesados en una o más casas de cocimiento (máquinas paralelas). La cantidad de cocimientos en cada lote es determinado por el tamaño del tanque de fermentación al cual el lote es asignado. En cada iteración i, se crea un lote S_i y el tipo demosto a ser producido en ese lote se selecciona de una lista restringida de candidatos (LRC), la cual se genera con los dos tipos de mostos que tienen la mayor demanda tomando en cuenta la disponibilidad de levadura. La manera en que se cuida que no se agote la levadura disponible es llevando un conteo del uso de la misma, siendo que para procesar un mosto w se requiere usar la levadura $k_w \in K$, y restringido que la levadura k_w no pueda ser utilizada $\bar{y}_{kw}$ número de veces de manera consecutiva. El lote S_i se asigna al tanque de fermentación $t^* \in T$ que será el primero en estar disponible, y el tamaño de S_i se determina por la capacidad de t^*. Después, se lleva a cabo la asignación de cada cocimiento $j \in S_i$ a las casas de cocimiento (máquinas). Cada cocimiento es asignado a la casa que esté más próxima a estar disponible. Este procedimiento se repite hasta que la demanda de cada mosto se satisfice (ver Algoritmo 3). Se programan paros generales en las casas de cocimiento de acuerdo a las políticas de la empresa. Nótese que la empresa no permite empezar la producción de ningún lote en la etapa de cocimiento si no hay taques de fermentación disponibles en ese momento. Por lo tanto, se agrega un tiempo en la etapa de cocimiento en caso de ser necesario para satisfacer esta restricción. La solución S que arroja la fase de construcción es una secuencia de lotes, que será alimentada posteriormente a la fase de mejora. La fase constructiva se corre una cantidad $iter_{max}$ de veces.

4.2.2 Fase de mejora

En esta fase se toma la solución S que se genera en la fase de construcción, se toma un lote S_i al azar y se hace una lista de posibles lotes con los cuáles puede intercambiar su posición, en esta lista entrarán sólo los lotes que sean del mismo tamaño y usen la misma levadura que S_i. De esta lista se selecciona un lote S_j de forma aleatoria. Si el intercambio llevado entre los lotes S_i y S_j produce una disminución en cuanto al tiempo de procesamiento requerido en la nueva secuencia, se realiza el intercambio de lotes y se actualiza el reparto de cocimientos en las casas de cocimiento y el makespan (ver Algoritmo 4). La fase de mejora se corre una cantidad $niter$ de veces.

Algoritmo 2 GRASP propuesto

Entrada: W: conjunto de mostos

H_w: conjunto de casas de cocimiento (máquinas) que pueden procesar mosto $w \in W$

d_w: demanda de mosto $w \in W$

k_w: levadura que usa mosto $w \in W$

A_h: arreglo de tiempos de disponibilidad de casas de cocimiento (máquinas)

A_t: arreglo de tiempos de disponibilidad de tanques de fermentación

$\bar{y}_{kw}$: límite de usos consecutivos de levadura k_w

Salida: S^b mejor solución encontrada.

1: $j \leftarrow 0$
2: **mientras** $j < iter_{max}$ **hacer**
3: $S \leftarrow \text{ConstructorSol}(W, H_w, d_w, k_w, A_h, A_t, \bar{y}_{kw})$
4: $S' \leftarrow \text{IntercambioLotes}(S, n_{iter})$
5: **si** $(f(S') < f(S))$ **entonces**
6: $S \leftarrow S''$
7: **fin si**
8: **si** $(f(S) < f(S^b))$ **entonces**
9: $S^b \leftarrow S$
10: **fin si**
11: $j = j + 1$
12: **fin mientras**
13: **Regresar** S^b

Algoritmo 3 ConstructorSol($W, H_w, d_w, k_w, A_h, A_t, \bar{y}_{kw}$)

Entrada: W: conjunto de mostos

H_w: conjunto de casas de cocimiento (máquinas) que pueden procesar mosto $w \in W$

d_w: demanda de mosto $w \in W$

k_w: levadura que usa mosto $w \in W$

A_h: arreglo de tiempos de disponibilidad de casas de cocimiento (máquinas)

A_t: arreglo de tiempos de disponibilidad de tanques de fermentación

$\bar{y}_{k\,w}$: límite de usos consecutivos de levadura k_w

Salida: $S \leftarrow \{S_1, S_2, ..., S_n\}$: Solución, secuencia de lotes a ser producidos

Conjunto $S \leftarrow \emptyset$

$i \leftarrow 0$ *// contador de lotes*

$y_f \leftarrow 0$, $f \in F$ *// contadores de levadura*

$stop_{prev} \leftarrow 0$ *//tiempo de paro general previo*

mientras $\exists w \in W : d_w > 0$ **hacer**

  Conjunto $i \leftarrow i + 1$

  Sea $t^* \leftarrow \arg\min_{t \in T}\{a_t\}$ el tanque que estará disponible primero

  Conjunto $S_i \leftarrow \emptyset$ *// crea el $i-$ésimo lote*

  Conjunto $|S_i| \leftarrow Q_{t^*}$ *// el tamaño del lote está dado por el tanque t^**

  Sea $l \leftarrow \arg\max_{w \in W}\{d_w\} : y_{f_w} < \bar{y}_{f_w}, f_w \in F\}$

  Sea $k \leftarrow \arg\max_{w \in W \setminus l}\{d_w\} : y_{f_w} < \bar{y}_{f_w}, f_w \in F\}$

  $LRC \leftarrow \{l, k\}$

  Elegir $\bar{w} \in LRC$ aleatoriamente *// determina el tipo de mosto a producir en lote S_i*

  para $j = 1, ..., |S_i|$ **hacer**

    Sea $h^* \leftarrow \arg\min_{h \in H_{\bar{w}}}\{a_h\}$ la casa que estará disponible primero

    si $j = 1$ **entonces**

      si $((a_{h^*} - stop_{prev}) \geq 10)||(((a_{h^*} - stop_{prev}) \geq 7)\&(a_{h^*} < a_{t^*}))$ **entonces**

        Insertar paro general en las líneas de producción, actualizar $a_h, h \in H$

        $stop_{prev} = a_{h^*}$

      si no, **si** $(a_{h^*} < a_{t^*})$ **entonces**

        Insertar tiempo muerto en las líneas de producción, actualizar $a_h, h \in H$

      fin si

    fin si

    Conjunto $st_i \leftarrow a_{h^*}$ *// Tiempo de inicio de producción de lote S_i*

    Asignar cocimiento $j \in S_i$ a casa h^*

    Actualizar a_{h^*}

  fin para

  $y_{f_{\bar{w}}} \leftarrow y_{f_{\bar{w}}} + 1$, $y_{f_w} \leftarrow 0, \forall f_w \in F \setminus f_{\bar{w}}$ *// actualizar los contadores de levadura*

  actualizar a_{t^*} considerando el tiempo de fermentación $fa_{\bar{w}}$ *// nuevo tiempo de disponibilidad de tanque t^**

  actualizar $d_{\bar{w}} \leftarrow d_{\bar{w}} - Q_{t^*}$ *// la demanda de mosto w*

  $S \leftarrow S \bigcup \{S_i\}$

fin mientras

Regresar S

Algoritmo 4 IntercambioLotes(S, n_{iter})

Entrada: $S \leftarrow \{S_1, S_2, ..., S_n\}$: Solución inicial (secuencia ordenada de lotes según el tiempo inicial de producción)

$iter_{max}$: número máximo de iteraciones

Salida: $S^* \leftarrow \{S_1, S_2, ..., S_n\}$: Mejor solución encontrada

1: $k \leftarrow 0, S^* \leftarrow S$
2: **mientras** $k < n_{iter}$ **hacer**
3: **para** cada $S_i \in S$, $i = 1, ..., n$ **hacer**
4: Sea N el conjunto de lotes en S que usan el mismo tipo de levadura y tienen el mismo tamaño que S_i
5: Selección aleatoria de $S_j \in N$
6: Sea S' la solución obtenida después del intercambio entre S_i y S_j en S
7: **si** S' tiene menor tiempo de procesamiento que la secuencia en S^* **entonces**
8: $S^* \leftarrow S'$
9: **fin si**
10: **fin para**
11: $k \leftarrow k + 1, S \leftarrow S^*$
12: **fin mientras**
13: **Regresar** S^*

4.3 Experimentación computacional y resultados

El algoritmo fue codificado en Java y la experimentación computacional se realizó en el mismo entorno que se describe en la sección 3.3. En esta sección se muestra la comparación entre los resultados obtenidos por el algoritmo propuesto y los resultados de dos instancias que nos fueron otorgadas por la empresa. La Tabla 4.1 muestra esa comparación. Cabe mencionar que la programación de producción que se hace en la empresa puede tomar una jornada laboral completa para generar una solución factible, mientras que el algoritmo propuesto tarda menos de 10 segundos para generar una solución de buena calidad.

Tabla 4.1: Comparación de programación de la empresa con la programación del algoritmo propuesto.

Instancia	Solución cervecería (días)	Solución GRASP (días)	Tiempo(s)	Mejora (%)
Inst0desv0.0	22	16.486	9.491	25.063
Inst1desv0.0	23	16.486	8.732	28.321

Además se probaron instancias ficticias que fueron creadas en base a información proporcionada por la empresa. En cada instancia se fija la cantidad de casas de cocimiento (máquinas), cantidad de mostos que se producen (familias de trabajo), tipos de levadura, y tanques de fermentación. En cada instancia se varía la demanda total, esto en un intento de emular las temporadas de alta demanda que se presentan en la compañía. Los resultados obtenidos se muestran en las Tablas 4.2, 4.3, y 4.4.

Tabla 4.2: Resultado con desviación baja en la demanda.

Instancia	Makespan mejor solución (días)	Tiempo de ejecución (s)
Inst0desv0.2	14.82	11.262
Inst1desv0.2	14.90	9.902
Inst2desv0.2	14.82	10.298
Inst3desv0.2	14.82	10.724
Inst4desv0.2	14.82	9.736
Inst5desv0.2	14.82	10.629
Inst6desv0.2	14.82	10.080
Inst7desv0.2	14.82	9.963
Inst8desv0.2	14.82	9.113
Inst9desv0.2	14.40	8.745
Inst10desv0.2	14.82	10.176
Inst11desv0.2	14.82	10.225
Inst12desv0.2	14.40	9.784
Inst13desv0.2	14.82	9.324
Inst14desv0.2	14.82	9.244
Inst15desv0.2	14.82	9.246
Inst16desv0.2	14.40	8.890
Inst17desv0.2	14.40	9.177
Inst18desv0.2	14.82	9.519
Inst19desv0.2	14.90	10.535

Tabla 4.3: Resultado con desviación media en la demanda.

Instancia	Makespan mejor solución (días)	Tiempo de ejecución (s)
Inst0desv0.4	15.90	12.998
Inst1desv0.4	15.90	13.215
Inst2desv0.4	15.90	14.612
Inst3desv0.4	15.57	14.511
Inst4desv0.4	15.49	13.129
Inst5desv0.4	16.07	15.424
Inst6desv0.4	15.65	12.590
Inst7desv0.4	15.57	13.041
Inst8desv0.4	15.49	12.359
Inst9desv0.4	15.57	12.868
Inst10desv0.4	15.90	13.731
Inst11desv0.4	15.49	12.472
Inst12desv0.4	15.57	13.060
Inst13desv0.4	15.90	14.192
Inst14desv0.4	15.90	13.711
Inst15desv0.4	16.50	14.334
Inst16desv0.4	15.49	12.848
Inst17desv0.4	15.49	13.539
Inst18desv0.4	15.65	13.164
Inst19desv0.4	15.49	12.693

Tabla 4.4: Resultado con desviación alta en la demanda.

Instancia	Makespan mejor solución (días)	Tiempo de ejecución (s)
Inst0desv0.6	17.57	16.964
Inst1desv0.6	17.15	16.009
Inst2desv0.6	16.65	15.688
Inst3desv0.6	16.65	15.533
Inst4desv0.6	17.24	16.464
Inst5desv0.6	17.15	16.340
Inst6desv0.6	16.82	16.314
Inst7desv0.6	17.15	16.036
Inst8desv0.6	17.24	16.457
Inst9desv0.6	17.15	18.490
Inst10desv0.6	17.24	17.276
Inst11desv0.6	16.82	16.120
Inst12desv0.6	16.74	16.111
Inst13desv0.6	16.65	16.184
Inst14desv0.6	16.65	17.024
Inst15desv0.6	17.15	16.403
Inst16desv0.6	16.49	15.483
Inst17desv0.6	17.15	15.926
Inst18desv0.6	16.74	15.972
Inst19desv0.6	17.15	17.102

4.3.1 Análisis de resultados

En la Tabla 4.1 se observa que la diferencia entre los tiempos en que se satisface la programación de producción generada por el algoritmo propuesto permite cumplir con los requerimientos de demanda en un tiempo menor (6 a 7 días) con respecto a la programación realizada por el tomador de decisiones en la empresa. Una programación eficiente significa un incremento en la capacidad

de producción. Para una empresa que está produciendo continuamente y cuyos productos son tan demandados por la sociedad, esto es una ventaja considerable, ya que entre más rápido se cubra una demanda proyectada, más pueden ampliar su horizonte de planeación, y se cuenta con más tiempo para adaptarse a cualquier contratiempo que pueda surgir por causas externas a la empresa.

En cuanto a las Tablas 4.2, 4.3, y 4.4, su objetivo es mostrar el impacto que tiene el incremento de la demanda manteniendo las condiciones de la planta, con el fin de tener un conocimiento más profundo acerca de los alcances de la misma. Nótese que estos cambios en la demanda no impactan el desempeño computacional del algoritmo, como puede observarse en las tablas, el tiempo promedio permanece similar en los tres casos analizados.

Capítulo 5

Conclusiones

En este trabajo se desarrolló e implementó un algoritmo ALNS híbrido, para la resolución del FJSP. El método probó ser muy eficiente con ciertos grupos de instancias tomadas de la literatura, alcanzando los valores óptimos para más de la mitad de las instancias en las que se conoce su solución óptima, pero no muy eficiente con otros grupos de instancias. Esto puede deberse a que algunos de los parámetros del ALNS se fijaron de antemano y permanecen fijos a lo largo de todo el procedimiento y experimentación.

Una manera en al que se pueda intentar mejorar los resultados sería implementar técnicas que nos permitan ajustar los parametros para cada instancia, al permitirle reactividad al algoritmo es posible encontrar una mejor combinación de parámetros que permitan generar mejores resultados.

Por otra parte, en esta investigación se presentó un caso de estudio de una planta cervecera de la localidad con características que no habían sido estudiadas en la literatura: partición de lotes y número de trabajos variables. Además, la cantidad de producto elaborado depende de la máquina en que se procesen los trabajos que integran los lotes. Esta característica no se presenta en ninguno de los trabajos de la literatura, por lo que se propuso un algoritmo tipo GRASP que pudiera determinar la cantidad de trabajos a procesar de manera dinámica. El GRASP propuesto para el problema de la programación de la secuencia de producción de cerveza probó ser más eficiente que los métodos de planificación que se utilizan en la empresa actualmente, tanto en la calidad de la solución como el tiempo en que se tarda en obtenerla. Esto resulta ser muy práctico para los planificadores de la empresa, además de que les da la ventaja de que pueden verificar y rectificar cualquier detalle externo a la empresa que pueda surgir de imprevisto.

En cuanto las instancias ficticias, creadas para simular el incremento de demanda que se da en algunas temporadas, se puede observar que el tiempo de producción varía entre 15, 16 y 17 días en promedio cuando se incrementa la demanda en un 20 %, 40 % y hasta 60 %, respectivamente, lo que implica un incremento no muy grande respecto a las instancias reales que nos fueron proporcionadas.

El algoritmo y resultados del caso de estudio están basados en una de las plantas que posee la empresa cervecera, por lo que una posible extensión al algoritmo sería la generalización de los parámetros alimentados, como lo son el número de casas de cocimiento con sus respectivas capacidades y tiempos de producción, el tipo de levaduras utilizadas, el tipo de mostos producidos en la planta, cantidad y capacidad de tanques de fermentación.

Apéndice A

Apéndice

En este apartado se muestran las tablas de resultados desglosadas, en las cuales se aprecia el valor objetivo que se obtuvo para cada instancia del FJSP, así como el tiempo de ejecución del algoritmo y el gap (%) con respecto a la mejor solución reportada, hasta el momento, en la literatura. También se muestran las mejores cotas que se han reportado. En la primera columna de las tablas presentadas se muestra el nombre de la instancia, la segunda columna determina la cantidad de n trabajos y la cantidad m de máquinas. En las columnas 3 y 4 se reportan las cotas conocidas, mientras que en la columna 5 se muestra el valor objetivo obtenido en este trabajo. El tiempo de ejecución se observa en la columna 6. La columna 7 indica solamente si se ha reportado el óptimo en la literatura. La columna 8 muestra el gap entre el valor objetivo de la columna 5 y la mejor solución reportada en la literatura, es decir, con la mejor cota superior que se muestra para cada instancia en la literatura (columna 4).

Tabla A.1: Resultados instancias tipo Barnes

Instancia	nxm	cota inferior	cota superior	valor objetivo	tiempo (s)	¿óptimo conocido?	gap (%)
Barnes_mt10c1	10x11	927	927	927	42	si	0.000
Barnes_mt10cc	10x12	908	908	908	33	si	0.000
Barnes_mt10x	10x11	918	918	918	48	si	0.000
Barnes_mt10xx	10x12	918	918	918	90	si	0.000
Barnes_mt10xxx	10x13	918	918	918	84	si	0.000
Barnes_mt10xy	10x12	905	905	905	24	si	0.000
Barnes_mt10xyz	10x13	847	847	847	17	si	0.000
Barnes_setb4c9	15x11	914	914	914	133	si	0.000
Barnes_setb4cc	15x12	907	907	907	1103	si	0.000
Barnes_setb4x	15x11	925	925	925	260	si	0.000
Barnes_setb4xx	15x12	925	925	925	632	si	0.000
Barnes_setb4xxx	15x13	925	925	925	220	si	0.000
Barnes_setb4xy	15x12	910	910	910	137	si	0.000
Barnes_setb4xyz	15x13	902	902	902	103	si	0.000
Barnes_seti5c12	15x16	1169	1169	1169	1560	si	0.000
Barnes_seti5cc	15x17	1135	1135	1135	1306	si	0.000
Barnes_seti5x	15x16	1198	1198	1205	1505	si	0.584
Barnes_seti5xx	15x17	1194	1194	1201	1306	si	0.586
Barnes_seti5xxx	15x18	1194	1194	1204	1304	si	0.838
Barnes_seti5xy	15x17	1135	1135	1135	1807	si	0.000
Barnes_seti5xyz	15x18	1125	1125	1125	1907	si	0.000

Tabla A.2: Resultados instancias tipo Brandimarte

Instancia	nxm	cota inferior	cota superior	valor objetivo	tiempo (s)	¿óptimo conocido?	gap (%)
Brandimarte_Mk01	10x6	40	40	40	47	si	0.000
Brandimarte_Mk02	10x6	26	26	27	1101	si	3.846
Brandimarte_Mk03	15x8	204	204	204	1305	si	0.000
Brandimarte_Mk04	15x8	60	60	60	1402	si	0.000
Brandimarte_Mk05	15x4	172	172	180	1402	si	4.651
Brandimarte_Mk06	10x15	57	57	80	1305	si	40.351
Brandimarte_Mk07	20x5	139	139	156	1402	si	12.230
Brandimarte_Mk08	20x10	523	523	523	2210	si	0.000
Brandimarte_Mk09	20x10	307	307	389	2417	si	26.710
Brandimarte_Mk10	20x15	189	193	323	2121	no	67.358

Tabla A.3: Resultados instancias tipo Dauzere

Instancia	nxm	cota inferior	cota superior	valor objetivo	tiempo (s)	¿óptimo conocido?	gap (%)
Dauzere_01a	10x5	2505	2505	2752	1306	si	9.860
Dauzere_02a	10x5	2228	2230	2608	1408	no	16.951
Dauzere_03a	10x5	2228	2228	2481	1610	si	11.355
Dauzere_04a	10x5	2503	2503	2668	1606	si	6.592
Dauzere_05a	10x5	2192	2205	2460	1507	no	11.565
Dauzere_06a	10x5	2163	2174	2747	3916	no	26.357
Dauzere_07a	15x8	2216	2276	2601	1716	no	14.279
Dauzere_08a	15x8	2061	2062	2905	3039	no	40.883
Dauzere_09a	15x8	2061	2061	2901	2749	si	40.757
Dauzere_10a	15x8	2212	2263	2713	1815	no	19.885
Dauzere_11a	15x8	2018	2041	2934	3330	no	43.753
Dauzere_12a	15x8	1969	1994	2904	2139	no	45.637
Dauzere_13a	20x10	2197	2245	2600	1826	no	15.813
Dauzere_14a	20x10	2161	2161	3207	2257	si	48.404
Dauzere_15a	20x10	2161	2161	3273	1870	si	51.458
Dauzere_16a	20x10	2193	2232	2805	2328	no	25.672
Dauzere_17a	20x10	2088	2110	3097	1737	no	46.777
Dauzere_18a	20x10	2057	2078	3175	2182	no	52.791

Tabla A.4: Resultados instancias tipo Hurink$_e$*data*

Instancia	nxm	cota inferior	cota superior	valor objetivo	tiempo (s)	¿óptimo conocido?	gap (%)
Hurink_edata_abz5	10x10	1167	1167	1167	13	si	0.000
Hurink_edata_abz6	10x10	925	925	925	11	si	0.000
Hurink_edata_abz7	20x15	604	613	686	1611	no	11.909
Hurink_edata_abz8	20x15	625	636	722	1410	no	13.522
Hurink_edata_abz9	20x15	644	644	694	2417	si	7.764
Hurink_edata_car1	11x5	6176	6176	6176	43	si	0.000
Hurink_edata_car2	13x4	6327	6327	6455	1101	si	2.023
Hurink_edata_car3	12x5	6856	6856	6856	361	si	0.000
Hurink_edata_car4	14x4	7789	7789	7789	1101	si	0.000
Hurink_edata_car5	10x6	7229	7229	7229	10	si	0.000
Hurink_edata_car6	8x9	7990	7990	7990	10	si	0.000
Hurink_edata_car7	7x7	6123	6123	6123	1	si	0.000
Hurink_edata_car8	8x8	7689	7689	7689	7	si	0.000
Hurink_edata_la01	10x5	609	609	609	9	si	0.000
Hurink_edata_la02	10x5	655	655	655	9	si	0.000
Hurink_edata_la03	10x5	550	550	550	9	si	0.000
Hurink_edata_la04	10x5	568	568	568	8	si	0.000
Hurink_edata_la05	10x5	503	503	503	7	si	0.000
Hurink_edata_la06	15x5	833	833	833	1201	si	0.000
Hurink_edata_la07	15x5	762	762	762	1201	si	0.000
Hurink_edata_la08	15x5	845	845	845	1101	si	0.000
Hurink_edata_la09	15x5	878	878	878	1201	si	0.000
Hurink_edata_la10	15x5	866	866	866	1201	si	0.000
Hurink_edata_la11	20x5	1103	1103	1106	1301	si	0.272
Hurink_edata_la12	20x5	960	960	960	1202	si	0.000
Hurink_edata_la13	20x5	1053	1053	1053	1202	si	0.000
Hurink_edata_la14	20x5	1123	1123	1123	1202	si	0.000
Hurink_edata_la15	20x5	1111	1111	1125	1302	si	1.260
Hurink_edata_la16	10x10	892	892	892	13	si	0.000
Hurink_edata_la17	10x10	707	707	707	9	si	0.000
Hurink_edata_la18	10x10	842	842	842	12	si	0.000
Hurink_edata_la19	10x10	796	796	796	9	si	0.000
Hurink_edata_la20	10x10	857	857	857	10	si	0.000
Hurink_edata_la21	15x10	1009	1009	1059	1303	si	4.955
Hurink_edata_la22	15x10	880	880	880	561	si	0.000
Hurink_edata_la23	15x10	950	950	956	1503	si	0.632
Hurink_edata_la24	15x10	908	908	908	1403	si	0.000
Hurink_edata_la25	15x10	936	936	936	713	si	0.000
Hurink_edata_la26	20x10	1106	1106	1164	1706	si	5.244
Hurink_edata_la27	20x10	1181	1181	1294	1506	si	9.568
Hurink_edata_la28	20x10	1142	1142	1211	1605	si	6.042
Hurink_edata_la29	20x10	1107	1107	1160	1908	si	4.788
Hurink_edata_la30	20x10	1188	1188	1281	1405	si	7.828
Hurink_edata_la31	30x10	1532	1532	1678	2720	si	9.530
Hurink_edata_la32	30x10	1698	1698	1767	2316	si	4.064
Hurink_edata_la33	30x10	1547	1547	1713	1612	si	10.730
Hurink_edata_la34	30x10	1599	1599	1767	2012	si	10.507
Hurink_edata_la35	30x10	1736	1736	1829	1713	si	5.357
Hurink_edata_la36	15x15	1160	1160	1160	423	si	0.000
Hurink_edata_la37	15x15	1397	1397	1397	1506	si	0.000
Hurink_edata_la38	15x15	1141	1141	1142	1405	si	0.088
Hurink_edata_la39	15x15	1184	1184	1184	794	si	0.000
Hurink_edata_la40	15x15	1144	1144	1173	1406	si	2.535
Hurink_edata_mt06	6x6	55	55	55	2	si	0.000
Hurink_edata_mt10	10x10	871	871	871	27	si	0.000
Hurink_edata_mt20	20x5	1088	1088	1121	1402	si	3.033
Hurink_edata_orb10	10x10	933	933	933	21	si	0.000
Hurink_edata_orb1	10x10	977	977	977	31	si	0.000
Hurink_edata_orb2	10x10	865	865	865	17	si	0.000
Hurink_edata_orb3	10x10	951	951	951	120	si	0.000
Hurink_edata_orb4	10x10	984	984	984	59	si	0.000
Hurink_edata_orb5	10x10	842	842	842	14	si	0.000
Hurink_edata_orb6	10x10	958	958	958	50	si	0.000
Hurink_edata_orb7	10x10	389	389	389	10	si	0.000
Hurink_edata_orb8	10x10	894	894	894	32	si	0.000
Hurink_edata_orb9	10x10	933	933	933	50	si	0.000

Tabla A.5: Resultados instancias tipo Hurink$_r$*data*

Instancia	nxm	cota inferior	cota superior	valor objetivo	tiempo (s)	¿óptimo conocido?	gap (%)
Hurink_rdata_abz5	10 x 10	**954**	**954**	961	1102	si	0.734
Hurink_rdata_abz6	10 x 10	**807**	**807**	807	12	si	0.000
Hurink_rdata_abz7	20 x 15	493	530	623	1822	no	17.547
Hurink_rdata_abz8	20 x 15	507	541	616	2728	no	13.863
Hurink_rdata_abz9	20 x 15	517	537	636	2023	no	18.436
Hurink_rdata_car1	11 x 5	**5034**	**5034**	5089	1301	si	1.093
Hurink_rdata_car2	13 x 4	**5985**	**5985**	5997	1308	si	0.201
Hurink_rdata_car3	12 x 5	5622	5623	5660	1501	no	0.658
Hurink_rdata_car4	14 x 4	**6514**	**6514**	6563	1522	si	0.752
Hurink_rdata_car5	10 x 6	**5615**	**5615**	5698	1101	si	1.478
Hurink_rdata_car6	8 x 9	**6147**	**6147**	6147	20	si	0.000
Hurink_rdata_car7	7 x 7	**4425**	**4425**	4425	4	si	0.000
Hurink_rdata_car8	8 x 8	**5692**	**5692**	5692	19	si	0.000
Hurink_rdata_la01	10 x 5	**570**	**570**	571	1201	si	0.175
Hurink_rdata_la02	10 x 5	**529**	**529**	532	1201	si	0.567
Hurink_rdata_la03	10 x 5	**477**	**477**	480	1316	si	0.629
Hurink_rdata_la04	10 x 5	**502**	**502**	507	1312	si	0.996
Hurink_rdata_la05	10 x 5	**457**	**457**	459	1307	si	0.438
Hurink_rdata_la06	15 x 5	**799**	**799**	803	1102	si	0.501
Hurink_rdata_la07	15 x 5	**749**	**749**	761	1101	si	1.602
Hurink_rdata_la08	15 x 5	**765**	**765**	773	1102	si	1.046
Hurink_rdata_la09	15 x 5	**853**	**853**	858	1202	si	0.586
Hurink_rdata_la10	15 x 5	**804**	**804**	809	1201	si	0.622
Hurink_rdata_la11	20 x 5	**1071**	**1071**	1085	1102	si	1.307
Hurink_rdata_la12	20 x 5	**936**	**936**	942	1202	si	0.641
Hurink_rdata_la13	20 x 5	**1038**	**1038**	1040	1602	si	0.193
Hurink_rdata_la14	20 x 5	**1070**	**1070**	1083	1302	si	1.215
Hurink_rdata_la15	20 x 5	**1089**	**1089**	1110	1708	si	1.928
Hurink_rdata_la16	10 x 10	**717**	**717**	717	22	si	0.000
Hurink_rdata_la17	10 x 10	**646**	**646**	646	10	si	0.000
Hurink_rdata_la18	10 x 10	**666**	**666**	666	83	si	0.000
Hurink_rdata_la19	10 x 10	**700**	**700**	700	66	si	0.000
Hurink_rdata_la20	10 x 10	**756**	**756**	756	13	si	0.000
Hurink_rdata_la21	15 x 10	808	825	910	1906	no	10.303
Hurink_rdata_la22	15 x 10	741	755	804	1805	no	6.490
Hurink_rdata_la23	15 x 10	816	832	921	1504	no	10.697
Hurink_rdata_la24	15 x 10	775	800	845	2207	no	5.625
Hurink_rdata_la25	15 x 10	768	784	831	2206	no	5.995
Hurink_rdata_la26	20 x 10	1056	1058	1128	3014	no	6.616
Hurink_rdata_la27	20 x 10	1085	1087	1222	1809	no	12.420
Hurink_rdata_la28	20 x 10	1075	1076	1136	2311	no	5.576
Hurink_rdata_la29	20 x 10	993	994	1074	2011	no	8.048
Hurink_rdata_la30	20 x 10	1068	1072	1221	1207	no	13.899
Hurink_rdata_la31	30 x 10	**1520**	**1520**	1766	1717	si	16.184
Hurink_rdata_la32	30 x 10	**1657**	**1657**	1948	1819	si	17.562
Hurink_rdata_la33	30 x 10	**1497**	**1497**	1672	1818	si	11.690
Hurink_rdata_la34	30 x 10	**1535**	**1535**	1718	2528	si	11.922
Hurink_rdata_la35	30 x 10	**1549**	**1549**	1754	2219	si	13.234
Hurink_rdata_la36	15 x 15	**1023**	**1023**	1077	2415	si	5.279
Hurink_rdata_la37	15 x 15	**1062**	**1062**	1126	2213	si	6.026
Hurink_rdata_la38	15 x 15	**954**	**954**	976	2312	si	2.306
Hurink_rdata_la39	15 x 15	**1011**	**1011**	1083	1913	si	7.122
Hurink_rdata_la40	15 x 15	**955**	**955**	1010	1912	si	5.759
Hurink_rdata_mt06	6 x 6	**47**	**47**	47	1	si	0.000
Hurink_rdata_mt10	10 x 10	**686**	**686**	686	134	si	0.000
Hurink_rdata_mt20	20 x 5	**1022**	**1022**	1033	1602	si	1.076
Hurink_rdata_orb10	10 x 10	**742**	**742**	742	1202	si	0.000
Hurink_rdata_orb1	10 x 10	**746**	**746**	746	30	si	0.000
Hurink_rdata_orb2	10 x 10	**696**	**696**	696	1177	si	0.000
Hurink_rdata_orb3	10 x 10	**712**	**712**	712	1302	si	0.000
Hurink_rdata_orb4	10 x 10	**753**	**753**	753	40	si	0.000
Hurink_rdata_orb5	10 x 10	**639**	**639**	639	125	si	0.000
Hurink_rdata_orb6	10 x 10	**754**	**754**	754	1202	si	0.000
Hurink_rdata_orb7	10 x 10	**302**	**302**	302	1202	si	0.000
Hurink_rdata_orb8	10 x 10	**639**	**639**	640	1202	si	0.156
Hurink_rdata_orb9	10 x 10	**694**	**694**	694	27	si	0.000

Tabla A.6: Resultados instancias tipo Hurink$_s$*data*

Instancia	nxm	cota inferior	cota superior	valor objetivo	tiempo (s)	¿óptimo conocido?	gap (%)
Hurink_sdata_abz5	10 x 10	**1234**	**1234**	1234	26	si	0.000
Hurink_sdata_abz6	10 x 10	**943**	**943**	943	10	si	0.000
Hurink_sdata_abz7	20 x 15	**656**	**656**	696	1611	si	6.098
Hurink_sdata_abz8	20 x 15	653	667	706	1610	no	5.847
Hurink_sdata_abz9	20 x 15	**678**	**678**	726	2314	si	7.080
Hurink_sdata_car1	11 x 5	**7038**	**7038**	7038	70	si	0.000
Hurink_sdata_car2	13 x 4	**7166**	**7166**	7166	1100	si	0.000
Hurink_sdata_car3	12 x 5	**7312**	**7312**	7312	1200	si	0.000
Hurink_sdata_car4	14 x 4	**8003**	**8003**	8003	1101	si	0.000
Hurink_sdata_car5	10 x 6	**7702**	**7702**	7702	94	si	0.000
Hurink_sdata_car6	8 x 9	**8313**	**8313**	8313	7	si	0.000
Hurink_sdata_car7	7 x 7	**6558**	**6558**	6558	3	si	0.000
Hurink_sdata_car8	8 x 8	**8264**	**8264**	8264	8	si	0.000
Hurink_sdata_la01	10 x 5	**666**	**666**	666	8	si	0.000
Hurink_sdata_la02	10 x 5	**655**	**655**	655	6	si	0.000
Hurink_sdata_la03	10 x 5	**597**	**597**	597	2	si	0.000
Hurink_sdata_la04	10 x 5	**590**	**590**	590	2	si	0.000
Hurink_sdata_la05	10 x 5	**593**	**593**	593	21	si	0.000
Hurink_sdata_la06	15 x 5	**926**	**926**	926	1201	si	0.000
Hurink_sdata_la07	15 x 5	**890**	**890**	890	1101	si	0.000
Hurink_sdata_la08	15 x 5	**863**	**863**	863	1100	si	0.000
Hurink_sdata_la09	15 x 5	**951**	**951**	951	1201	si	0.000
Hurink_sdata_la10	15 x 5	**958**	**958**	958	1201	si	0.000
Hurink_sdata_la11	20 x 5	**1222**	**1222**	1222	1201	si	0.000
Hurink_sdata_la12	20 x 5	**1039**	**1039**	1039	1101	si	0.000
Hurink_sdata_la13	20 x 5	**1150**	**1150**	1150	1101	si	0.000
Hurink_sdata_la14	20 x 5	**1292**	**1292**	1292	1101	si	0.000
Hurink_sdata_la15	20 x 5	**1207**	**1207**	1207	1201	si	0.000
Hurink_sdata_la16	10 x 10	**945**	**945**	945	10	si	0.000
Hurink_sdata_la17	10 x 10	**784**	**784**	784	10	si	0.000
Hurink_sdata_la18	10 x 10	**848**	**848**	848	7	si	0.000
Hurink_sdata_la19	10 x 10	**842**	**842**	842	11	si	0.000
Hurink_sdata_la20	10 x 10	**902**	**902**	902	9	si	0.000
Hurink_sdata_la21	15 x 10	**1046**	**1046**	1055	1502	si	0.860
Hurink_sdata_la22	15 x 10	**927**	**927**	927	914	si	0.000
Hurink_sdata_la23	15 x 10	**1032**	**1032**	1032	1202	si	0.000
Hurink_sdata_la24	15 x 10	**935**	**935**	935	566	si	0.000
Hurink_sdata_la25	15 x 10	**977**	**977**	977	543	si	0.000
Hurink_sdata_la26	20 x 10	**1218**	**1218**	1250	1405	si	2.627
Hurink_sdata_la27	20 x 10	**1235**	**1235**	1279	1204	si	3.563
Hurink_sdata_la28	20 x 10	**1216**	**1216**	1238	1304	si	1.809
Hurink_sdata_la29	20 x 10	**1152**	**1152**	1218	2005	si	5.729
Hurink_sdata_la30	20 x 10	**1355**	**1355**	1355	1305	si	0.000
Hurink_sdata_la31	30 x 10	**1784**	**1784**	1784	1512	si	0.000
Hurink_sdata_la32	30 x 10	**1850**	**1850**	1850	1411	si	0.000
Hurink_sdata_la33	30 x 10	**1719**	**1719**	1719	1410	si	0.000
Hurink_sdata_la34	30 x 10	**1721**	**1721**	1799	2112	si	4.532
Hurink_sdata_la35	30 x 10	**1888**	**1888**	1888	2515	si	0.000
Hurink_sdata_la36	15 x 15	**1268**	**1268**	1274	1506	si	0.473
Hurink_sdata_la37	15 x 15	**1397**	**1397**	1401	2008	si	0.286
Hurink_sdata_la38	15 x 15	**1196**	**1196**	1210	1607	si	1.171
Hurink_sdata_la39	15 x 15	**1233**	**1233**	1243	1506	si	0.811
Hurink_sdata_la40	15 x 15	**1222**	**1222**	1235	1605	si	1.064
Hurink_sdata_mt06	6 x 6	**55**	**55**	55	1	si	0.000
Hurink_sdata_mt10	10 x 10	**930**	**930**	930	56	si	0.000
Hurink_sdata_mt20	20 x 5	**1165**	**1165**	1175	1602	si	0.858
Hurink_sdata_orb10	10 x 10	**944**	**944**	944	14	si	0.000
Hurink_sdata_orb1	10 x 10	**1059**	**1059**	1059	1087	si	0.000
Hurink_sdata_orb2	10 x 10	**888**	**888**	888	11	si	0.000
Hurink_sdata_orb3	10 x 10	**1005**	**1005**	1005	259	si	0.000
Hurink_sdata_orb4	10 x 10	**1005**	**1005**	1005	17	si	0.000
Hurink_sdata_orb5	10 x 10	**887**	**887**	887	16	si	0.000
Hurink_sdata_orb6	10 x 10	**1010**	**1010**	1010	221	si	0.000
Hurink_sdata_orb7	10 x 10	**397**	**397**	397	17	si	0.000
Hurink_sdata_orb8	10 x 10	**899**	**899**	899	47	si	0.000
Hurink_sdata_orb9	10 x 10	**934**	**934**	934	29	si	0.000

Tabla A.7: Resultados instancias tipo Hurink$_v$*data*

Instancia	nxm	cota inferior	cota superior	valor objetivo	tiempo (s)	¿óptimo conocido?	gap (%)
Hurink_vdata_abz5	10 x 10	**859**	**859**	859	400	si	0.000
Hurink_vdata_abz6	10 x 10	**742**	**742**	742	98	si	0.000
Hurink_vdata_abz7	20 x 15	**492**	**492**	845	2458	si	71.748
Hurink_vdata_abz8	20 x 15	506	507	849	2465	no	67.456
Hurink_vdata_abz9	20 x 15	497	498	886	1650	no	77.912
Hurink_vdata_car1	11 x 5	**5005**	**5005**	5019	1406	si	0.280
Hurink_vdata_car2	13 x 4	**5929**	**5929**	5940	1201	si	0.186
Hurink_vdata_car3	12 x 5	**5597**	**5597**	5667	1101	si	1.251
Hurink_vdata_car4	14 x 4	**6514**	**6514**	6523	1301	si	0.138
Hurink_vdata_car5	10 x 6	4909	4912	4965	1407	no	1.079
Hurink_vdata_car6	8 x 9	**5486**	**5486**	5486	18	si	0.000
Hurink_vdata_car7	7 x 7	**4281**	**4281**	4281	5	si	0.000
Hurink_vdata_car8	8 x 8	**4613**	**4613**	4613	30	si	0.000
Hurink_vdata_la01	10 x 5	**570**	**570**	573	1212	si	0.526
Hurink_vdata_la02	10 x 5	**529**	**529**	530	1428	si	0.189
Hurink_vdata_la03	10 x 5	**477**	**477**	479	1318	si	0.419
Hurink_vdata_la04	10 x 5	**502**	**502**	503	1207	si	0.199
Hurink_vdata_la05	10 x 5	**457**	**457**	458	1420	si	0.219
Hurink_vdata_la06	15 x 5	**799**	**799**	811	1101	si	1.502
Hurink_vdata_la07	15 x 5	**749**	**749**	759	1201	si	1.335
Hurink_vdata_la08	15 x 5	**765**	**765**	769	1402	si	0.523
Hurink_vdata_la09	15 x 5	**853**	**853**	860	1402	si	0.821
Hurink_vdata_la10	15 x 5	**804**	**804**	811	1302	si	0.871
Hurink_vdata_la11	20 x 5	**1071**	**1071**	1080	1202	si	0.840
Hurink_vdata_la12	20 x 5	**936**	**936**	950	1302	si	1.496
Hurink_vdata_la13	20 x 5	**1038**	**1038**	1043	1202	si	0.482
Hurink_vdata_la14	20 x 5	**1070**	**1070**	1077	1303	si	0.654
Hurink_vdata_la15	20 x 5	**1089**	**1089**	1103	1303	si	1.286
Hurink_vdata_la16	10 x 10	**717**	**717**	717	84	si	0.000
Hurink_vdata_la17	10 x 10	**646**	**646**	646	81	si	0.000
Hurink_vdata_la18	10 x 10	**663**	**663**	663	114	si	0.000
Hurink_vdata_la19	10 x 10	**617**	**617**	617	292	si	0.000
Hurink_vdata_la20	10 x 10	**756**	**756**	756	117	si	0.000
Hurink_vdata_la21	15 x 10	**800**	**800**	864	2312	si	8.000
Hurink_vdata_la22	15 x 10	**733**	**733**	818	1709	si	11.596
Hurink_vdata_la23	15 x 10	**809**	**809**	927	1407	si	14.586
Hurink_vdata_la24	15 x 10	**773**	**773**	859	1609	si	11.125
Hurink_vdata_la25	15 x 10	**751**	**751**	823	2012	si	9.587
Hurink_vdata_la26	20 x 10	**1052**	**1052**	1615	2035	si	53.517
Hurink_vdata_la27	20 x 10	**1084**	**1084**	1596	1416	si	47.232
Hurink_vdata_la28	20 x 10	**1069**	**1069**	1575	1514	si	47.334
Hurink_vdata_la29	20 x 10	**993**	**993**	1572	2022	si	58.308
Hurink_vdata_la30	20 x 10	**1068**	**1068**	1615	2015	si	51.217
Hurink_vdata_la31	30 x 10	**1520**	**1520**	2052	3061	si	35.000
Hurink_vdata_la32	30 x 10	**1657**	**1657**	2207	2142	si	33.193
Hurink_vdata_la33	30 x 10	**1497**	**1497**	2149	1426	si	43.554
Hurink_vdata_la34	30 x 10	**1535**	**1535**	2105	1327	si	37.134
Hurink_vdata_la35	30 x 10	**1549**	**1549**	2099	2142	si	35.507
Hurink_vdata_la36	15 x 15	**948**	**948**	1629	1932	si	71.835
Hurink_vdata_la37	15 x 15	**986**	**986**	1662	2543	si	68.560
Hurink_vdata_la38	15 x 15	**943**	**943**	1553	1932	si	64.687
Hurink_vdata_la39	15 x 15	**922**	**922**	1489	1944	si	61.497
Hurink_vdata_la40	15 x 15	**955**	**955**	1554	2341	si	62.723
Hurink_vdata_mt06	6 x 6	**47**	**47**	47	0	si	0.000
Hurink_vdata_mt10	10 x 10	**655**	**655**	655	76	si	0.000
Hurink_vdata_mt20	20 x 5	**1022**	**1022**	1038	1904	si	1.566
Hurink_vdata_orb10	10 x 10	**681**	**681**	681	134	si	0.000
Hurink_vdata_orb1	10 x 10	**695**	**695**	695	110	si	0.000
Hurink_vdata_orb2	10 x 10	**620**	**620**	620	284	si	0.000
Hurink_vdata_orb3	10 x 10	**648**	**648**	648	106	si	0.000
Hurink_vdata_orb4	10 x 10	**753**	**753**	753	77	si	0.000
Hurink_vdata_orb5	10 x 10	**584**	**584**	584	467	si	0.000
Hurink_vdata_orb6	10 x 10	**715**	**715**	715	118	si	0.000
Hurink_vdata_orb7	10 x 10	**275**	**275**	275	264	si	0.000
Hurink_vdata_orb8	10 x 10	**573**	**573**	573	80	si	0.000
Hurink_vdata_orb9	10 x 10	**659**	**659**	659	119	si	0.000

Bibliografía

[1] Brandimarte, P., «Routing and scheduling in a flexible job shop by tabu search», *Annals of Operations Research*, **41**(3), págs. 157–183, 1993.

[2] Brucker, P., M. Kovalyov, Y. Shafransky y F. Werner, «Batch scheduling with deadlines on parallel machines», *Annals of Operations Research*, **83**(0), págs. 23–40, 1998.

[3] Buscher, U. y L. Shen, «Solving the Batch Scheduling Problem with Family Setup Times», en B. Fleischmann, K.-H. Borgwardt, R. Klein y A. Tuma (editores), *Operations Research Proceedings 2008*, tomo 2008, Springer Berlin Heidelberg, págs. 117–122, 2009.

[4] Buscher, U. y L. Shen, «MIP formulations and heuristics for solving parallel batching problems», *Journal of Systems Science and Complexity*, **23**(5), págs. 884–895, 2010.

[5] Cheng, B., Q. Wang, S. Yang y X. Hu, «An improved ant colony optimization for scheduling identical parallel batching machines with arbitrary job sizes», *Applied Soft Computing*, **13**(2), págs. 765–772, 2013.

[6] Cheng, T., B. Lin y A. Toker, «Makespan minimization in the two-machine flowshop batch scheduling problem», *Naval Research Logistics (NRL)*, **47**(2), págs. 128–144, 2000.

[7] Cheng, T. C. E. y Z.-L. Chen, «Parallel Machine Scheduling with Batch Setup Times», *Operations Research*, **42**(6), págs. 1171–1174, 1994.

[8] Choi, B.-C. y K. Lee, «Two-stage proportionate flexible flow shop to minimize the makespan», *Journal of Combinatorial Optimization*, **25**(1), págs. 123–134, 2013.

[9] Condotta, A., S. Knust y N. V. Shakhlevich, «Parallel batch scheduling of equal-length jobs with release and due dates», *Journal of Scheduling*, **13**(5), págs. 463–477, 2010.

[10] Costa, A., «Hybrid genetic optimization for solving the batch-scheduling problem in a pharmaceutical industry», *Computers & Industrial Engineering*, **79**, págs. 130–147, 2015.

[11] Crauwels, H., P. Beullens y D. Van Oudheusden, «Parallel machine scheduling by family batching with sequence-independent set-up times», *International Journal of Operations Research*, **3**(2), págs. 144–154, 2006.

[12] DASTIDAR, S. G. y R. NAGI, «Batch splitting in an assembly scheduling environment», *International Journal of Production Economics*, **105**(2), págs. 372–384, 2007.

[13] DORSEY, R. C., T. J. HODGSON y H. D. RATLIFF, «Technical NoteâA Production-Scheduling Problem with Batch Processing», *Operations Research*, **22**(6), págs. 1271–1279, 1974.

[14] FATTAHI, P., F. JOLAI y J. ARKAT, «Flexible job shop scheduling with overlapping in operations», *Applied Mathematical Modelling*, **33**(7), págs. 3076–3087, 2009.

[15] FATTAHI, P., M. S. MEHRABAD y F. JOLAI, «Mathematical modeling and heuristic approaches to flexible job shop scheduling problems», *Journal of Intelligent Manufacturing*, **18**(3), págs. 331–342, 2007.

[16] FEO, T. A. y M. G. RESENDE, «Greedy randomized adaptive search procedures», *Journal of global optimization*, **6**(2), págs. 109–133, 1995.

[17] GAREY, M. R., D. S. JOHNSON y R. SETHI, «The Complexity of Flowshop and Jobshop Scheduling», *Mathematics of Operations Research*, **1**(2), págs. 117–129, 1976.

[18] GHOSH, J. B., «Batch scheduling to minimize total completion time», *Operations Research Letters*, **16**(5), págs. 271–275, 1994.

[19] GRAHAM, R. L., E. L. LAWLER, J. K. LENSTRA y A. R. KAN, «Optimization and approximation in deterministic sequencing and scheduling: a survey», *Annals of discrete mathematics*, **5**, págs. 287–326, 1979.

[20] GRAVES, S. C., «A review of production scheduling», *Operations Research*, **29**(4), págs. 646–675, 1981.

[21] HALL, N. G. y C. N. POTTS, «The coordination of scheduling and batch deliveries», *Annals of operations research*, **135**(1), págs. 41–64, 2005.

[22] JIA, Z.-H. y J. Y.-T. LEUNG, «A meta-heuristic to minimize makespan for parallel batch machines with arbitrary job sizes», *European Journal of Operational Research*, **240**(3), págs. 649–665, 2015.

[23] KACEM, I., S. HAMMADI y P. BORNE, «Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems», *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, **32**(1), págs. 1–13, 2002.

[24] KASHAN, A. H., B. KARIMI y M. JENABI, «A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes», *Computers & Operations Research*, **35**(4), págs. 1084–1098, 2008.

[25] KIM, J., S.-H. KANG y S.-M. LEE, «Transfer batch scheduling for a two-stage flowshop with identical parallel machines at each stage», *Omega*, **25**(5), págs. 547–555, 1997.

[26] LEUNG, J. Y.-T., C. NG y T. E. CHENG, «Minimizing sum of completion times for batch scheduling of jobs with deteriorating processing times», *European Journal of Operational Research*, **187**(3), págs. 1090–1099, 2008.

[27] LI, S. y J. YUAN, «Parallel-machine parallel-batching scheduling with family jobs and release dates to minimize makespan», *Journal of Combinatorial Optimization*, **19**(1), págs. 84–93, 2010.

[28] LIN, B., T. CHENG y A. CHOU, «Scheduling in an assembly-type production chain with batch transfer», *Omega*, **35**(2), págs. 143–151, 2007.

[29] LOZANO, A. J. y A. L. MEDAGLIA, «Scheduling of parallel machines with sequence-dependent batches and product incompatibilities in an automotive glass facility», *Journal of Scheduling*, **17**(6), págs. 521–540, 2014.

[30] MALVE, S. y R. UZSOY, «A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families», *Computers & Operations Research*, **34**(10), págs. 3016–3028, 2007.

[31] MANJESHWAR, P. K., P. DAMODARAN y K. SRIHARI, «Minimizing makespan in a flow shop with two batch-processing machines using simulated annealing», *Robotics and Computer-Integrated Manufacturing*, **25**(3), págs. 667–679, 2009.

[32] MATHIRAJAN, M., V. CHANDRU y A. SIVAKUMAR, «Heuristic algorithms for scheduling heat-treatment furnaces of steel casting industries», *Sadhana*, **32**(5), págs. 479–500, 2007.

[33] MEHTA, S. V. y R. UZSOY, «Minimizing total tardiness on a batch processing machine with incompatible job families», *IIE transactions*, **30**(2), págs. 165–178, 1998.

[34] MÉNDEZ, C., G. HENNING y J. CERDÁ, «Optimal scheduling of batch plants satisfying multiple product orders with different due-dates», *Computers & Chemical Engineering*, **24**(9), págs. 2223–2245, 2000.

[35] MÉNDEZ, C. A. y J. CERDÁ, «An MILP framework for batch reactive scheduling with limited discrete resources», *Computers & chemical engineering*, **28**(6), págs. 1059–1068, 2004.

[36] MÉNDEZ, C. A., J. CERDÁ, I. E. GROSSMANN, I. HARJUNKOSKI y M. FAHL, «State-of-the-art review of optimization methods for short-term scheduling of batch processes», *Computers & Chemical Engineering*, **30**(6), págs. 913–946, 2006.

[37] MONMA, C. L. y C. N. POTTS, «On the complexity of scheduling with batch setup times», *Operations Research*, **37**(5), págs. 798–804, 1989.

[38] MOON, S., S. PARK y W. K. LEE, «New MILP models for scheduling of multiproduct batch plants under zero-wait policy», *Industrial & engineering chemistry research*, **35**(10), págs. 3458–3469, 1996.

[39] MOSHEIOV, G. y D. ORON, «Open-shop batch scheduling with identical jobs», *European Journal of Operational Research*, **187**(3), págs. 1282–1292, 2008.

[40] MOSHEIOV, G. y D. ORON, «A single machine batch scheduling problem with bounded batch size», *European Journal of Operational Research*, **187**(3), págs. 1069–1079, 2008.

[41] NOROOZI, A., H. MOKHTARI y I. N. K. ABADI, «Research on computational intelligence algorithms with adaptive learning approach for scheduling problems with batch processing machines», *Neurocomputing*, **101**, págs. 190–203, 2013.

[42] ÖZGÜVEN, C., L. ÖZBAKIR y Y. YAVUZ, «Mathematical models for job-shop scheduling problems with routing and process plan flexibility», *Applied Mathematical Modelling*, **34**(6), págs. 1539–1548, 2010.

[43] POTTS, C. N. y M. Y. KOVALYOV, «Scheduling with batching: a review», *European journal of operational research*, **120**(2), págs. 228–249, 2000.

[44] QUINTIQ, «Flexible Job shop Scheduling Problem Results», url http://www.quintiq.com/optimization/flexible-job-shop-scheduling-problem-results.html.

[45] RINNOOY KAN, A., «Machine scheduling problems: classification, complexity and computations», *Martinus Nijhoff, The Hague*, 1976.

[46] ROPKE, S. y D. PISINGER, «An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows», *TRANSPORTATION SCIENCE*, **40**(4), págs. 455–472, 2006.

[47] ROSHANAEI, V., «Mathematical Modelling and Optimization of Flexible Job Shops Scheduling Problem», , 2012.

[48] SHEN, L. y U. BUSCHER, «Solving the serial batching problem in job shop manufacturing systems», *European Journal of Operational Research*, **221**(1), págs. 14–26, 2012.

[49] SHEN, L., J. N. GUPTA y U. BUSCHER, «Flow shop batching and scheduling with sequence-dependent setup times», *Journal of Scheduling*, **17**(4), págs. 353–370, 2014.

[50] SHEN, L., L. MÖNCH y U. BUSCHER, «An iterative approach for the serial batching problem with parallel machines and job families», *Annals of Operations Research*, **206**(1), págs. 425–448, 2013.

[51] SHEN, L., L. MÖNCH y U. BUSCHER, «A simultaneous and iterative approach for parallel machine scheduling with sequence-dependent family setups», *Journal of Scheduling*, **17**(5), págs. 471–487, 2014.

[52] SUNG, C. S., Y. H. KIM y S. H. YOON, «A problem reduction and decomposition approach for scheduling for a flowshop of batch processing machines», *European Journal of Operational Research*, **121**(1), págs. 179–192, 2000.

[53] TANG, L. y P. LIU, «Minimizing makespan in a two-machine flowshop scheduling with batching and release time», *Mathematical and Computer Modelling*, **49**(5), págs. 1071–1077, 2009.

[54] TORABI, S., B. KARIMI y S. F. GHOMI, «The common cycle economic lot scheduling in flexible job shops: The finite horizon case», *International Journal of Production Economics*, **97**(1), págs. 52–65, 2005.

[55] VAN LAARHOVEN, P. J. y E. H. AARTS, *Simulated annealing*, Springer, 1987.

[56] WANG, H.-M. y F.-D. CHOU, «Solving the parallel batch-processing machines with different release times, job sizes, and capacity limits by metaheuristics», *Expert Systems with Applications*, **37**(2), págs. 1510–1521, 2010.

[57] WANG, Z., F. GAO, Q. ZHAI, X. GUAN, K. LIU y D. ZHOU, «An integrated optimization model for generation and batch production load scheduling in energy intensive enterprise», en *Power and Energy Society General Meeting, 2012 IEEE*, IEEE, págs. 1–8, 2012.

[58] XU, D., Z. CHENG, Y. YIN y H. LI, «Makespan minimization for two parallel machines scheduling with a periodic availability constraint», *Computers & Operations Research*, **36**(6), págs. 1809–1812, 2009.

[59] ZHANG, H. y M. GU, «Modeling job shop scheduling with batches and setup times by timed Petri nets», *Mathematical and Computer Modelling*, **49**(1), págs. 286–294, 2009.

Ficha autobiográfica

César Arturo Sáenz Alanís

Candidato para el grado de Maestro en Ciencias
en Ingeniería de Sistemas

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Tesis:

Secuenciación de Trabajos en Sistemas de Producción Flexibles

Nací el 12 de marzo de 1990, en la Ciudad de Monterrey, Nuevo León. Siendo el segundo hijo de la Lic. Amparo Alanís Ríos y de César Sáenz Treviño. Asistí a la primaria Gral. Jesús María Garza y la secundaria Jesús Cantú Leal. En el 2005 ingresé a la preparatoria # 15 Florida. Me gradué de la Facultad de Ciencias Químicas de la Universidad Autónoma de Nuevo León como Ingeniero Químico en diciembre del 2012. Fui practicante en la empresa PYOSA por 9 meses en el año 2012. En el año 2013 inicié mis estudios de maestría en el Posgrado de Ingeniería de Sistemas en la Facultad de Ingeniería Mecánica y Eléctrica de la Universidad Autónoma de Nuevo León, gracias al apoyo de la beca de CONACYT. Hice una estancia en la Ciudad de Québec, Canadá, en la Universidad Laval, bajo la tutela del Doctor Leandro Callegari Coelho.