Solución del Problema de Flowshop Distribuido y Permutado con Etapa de Ensamble considerando tiempos de alistamiento dependientes de la secuencia (DAPFSP-SDST) y fábricas heterogéneas a través de un algoritmo basado en VND.

Miguel Eduardo Correa González

Daniela Fernanda Ortíz Delgado

Trabajo de grado para optar el título de Ingeniero Industrial

Director:

Edwin Alberto Garavito Hernández

M.Sc Ingeniería Industrial

Universidad Industrial de Santander

Facultad de Ingenierías Físico-Mecánicas

Escuela de Estudios Industriales y Empresariales

Bucaramanga

2017

DEDICATORIA

Dedico el presente trabajo a mis padres por su apoyo incondicional durante el estudio de mi carrera, por apoyar mis proyectos como si fueran propios y permitirme vivir experiencias que fortalecieron mi formación profesional y personal. Agradezco los sacrificios que han tenido que hacer por mí y ofrezco este trabajo como la culminación de todos nuestros esfuerzos colectivos.

También lo dedico a mi novia y compañera de tesis por tomar parte en esta travesía y por un logro más en nuestro crecimiento profesional.

AGRADECIMIENTOS

Agradezco a mi familia por el apoyo recibido durante mi carrera, en especial a Viviana Correa y Omar Jaimes quienes fueron un ejemplo a seguir y una promesa de que el trabajo duro tiene sus recompensas.

Agradezco a los miembros del grupo OPALO por su apoyo, asesoramiento y sobre todo su amistad. Especialmente a Daniel Martínez y Leonardo Talero quienes fueron un apoyo incondicional durante la realización del proyecto y que sin duda alguna hicieron mucho más ameno este proceso de formación.

Finalmente quiero agradecer al Profesor Carlos Díaz y a Patricia Almeida por darme la oportunidad de pertenecer a la familia de posgrados de la Escuela de Estudios Industriales y Empresariales de la cual me llevo muchos recuerdos memorables tanto de experiencias como de personas espectaculares que conocí durante mi instancia allí.

Miguel Eduardo Correa González

DEDICATORIA

A mis ojitos de capulí,
porque sus bendiciones y enseñanzas perduran en mí y
su amor en mi corazón me mantiene fuerte y me hace seguir adelante.

A mi madre,
por ser una mujer luchadora, un gran ejemplo y por confiar en mí,
espero que esté conmigo mucho tiempo más para recibir su amor.

A Nicole,
porque mi esfuerzo por salir adelante también es por ella, quiero apoyarle y
animarla a ser feliz.

A mi Padre y Abuelito,
porque los imagino allí arriba sonriéndome y bendiciéndome.
Los amo.

AGRADECIMIENTOS

A Miguel, por acompañarme con amor y paciencia.

A Jessi, Sebas, Luisa, Angie, Jean y Camilo por su valiosa amistad, los quiero mucho.

A Paty y Mary González porque las sentí como mis madres, gracias por cuidar de mí y apoyarme.

Al profesor Edwin y Carlos Díaz, por su guía en la realización de este proyecto.

A Daniel Martínez y Leonardo Talero, por su amistad y disposición a ayudarnos y compartir su conocimiento.

Al grupo de Investigación OPALO, por todo su apoyo. Los extrañaré, serán grandes.

A mi familia, por su apoyo en las diferentes etapas de mi vida.

Daniela Ortíz Delgado

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Lista de Apéndices

Los siguientes apéndices se encuentran en la carpeta adjunta:

Apéndice A. Análisis bibliométrico.

Apéndice B. Gms del Modelo MILP para DAPFSP_SDST.

Apéndice C. Scripts en Matlab usados para la ejecución del modelo MILP de Gams.

Apéndice D. Descripción de funciones utilizadas en Gams y Matlab para ejecución de Modelo MILP.

Apéndice E. Resultados del modelo MILP para DAPFSP-SDST con fábricas homogéneas.

Apéndice F. Resultados del modelo MILP para DAPFSP-SDST con fábricas heterogéneas.

Apéndice G. Rutinas en Matlab utilizadas en la implementación del Algoritmo.

Apéndice H. Makespan y tiempo para corridas de calibración.

Apéndice I. Pruebas de supuestos y análisis de varianza para la variable TIME.

Apéndice J. Resultados del Algoritmo para DAPFSP-SDST con fábricas heterogéneas.

Apéndice K. Resultados del Algoritmo para DAPFSP-SDST con fábricas homogéneas.

RESUMEN

TÍTULO: "SOLUCIÓN DEL PROBLEMA DE FLOWSHOP DISTRIBUIDO Y PERMUTADO CON ETAPA DE ENSAMBLE CONSIDERANDO TIEMPOS DE ALISTAMIENTO DEPENDIENTES DE LA SECUENCIA (DAPFSP-SDST) Y FÁBRICAS HETEROGÉNEAS A TRAVÉS DE UN ALGORITMO BASADO EN VND"[1]

AUTORES:
CORREA GONZÁLEZ, Miguel Eduardo
ORTIZ DELGADO, Daniela Fernanda[2]

PALABRAS CLAVES:
Flowshop distribuido, fábricas heterogéneas, ensamble, MILP, algoritmo, metaheurística, búsqueda descendente, VND, programación.

DESCRIPCIÓN:
En esta investigación se estudia el problema de *Flowshop* Distribuido y Permutado con etapa de ensamble (DAPSP) considerando tiempos de alistamiento dependientes de la secuencia y fábricas heterogéneas con el objetivo de minimizar el *makespan*. Para su solución, se diseñó un modelo MILP y un algoritmo basado en VND compuesto por una etapa de generación de población inicial de secuencias de ensamble, otra de mejoramiento de la secuencia de procesamiento a través de estructuras de vecindario y finalmente una de exploración usando el operador genético de cruce entre secuencias de ensamble.

El Modelo MILP se implementó en el software GAMS® y se validó con 900 instancias pequeñas encontradas en la literatura para fábricas homogéneas, además se creó un diseño experimental para determinar los factores que inciden en el gap relativo.

El algoritmo se implementó en Matlab® y sus parámetros fueron calibrados a través de un diseño experimental; su desempeño fue evaluado mediante la comparación con los resultados del modelo MILP para instancias pequeñas con fábricas heterogéneas y con la *best solution* de la literatura para instancias grandes con fábricas homogéneas.

A través del diseño experimental se concluye que los trabajos, productos, fábricas y máquinas tienen un efecto significativo en el gap relativo dado por Gams. De la calibración del algoritmo se infiere que los mejores resultados de *makespan* se obtienen cuando el número de mejoras es grande y el intervalo de salto es bajo. Así mismo se encontró que el algoritmo es mejor que el modelo MILP para instancias pequeñas pues logra encontrar soluciones de calidad similar en tiempos computacionales significativamente más bajos; y en comparación con los métodos encontrados en la literatura para fábricas homogéneas, el algoritmo presenta un buen desempeño pues logra encontrar mejores soluciones o muy cercanas a las mejores encontradas en la literatura.

---

[1] Proyecto de Grado
[2] Facultad de Ingenierías Físico-mecánicas. Escuela de Estudios Industriales y Empresariales. Programa de Ingeniería Industrial. Director: M.Sc Edwin Alberto Garavito Hernández.

ABSTRACT

TITLE: "A SOLUTION TO THE DISTRIBUTED ASSEMBLY PERMUTATION FLOWSHOP SCHEDULING PROBLEM WITH SEQUENCE DEPENDENT SETUP TIMES (DAPFSP-SDST) AND HETEROGENEOUS FACTORIES THROUGH A VND-BASED ALGORITHM"[3]

AUTHORS:
CORREA GONZÁLEZ, Miguel Eduardo
ORTIZ DELGADO, Daniela Fernanda[4]

KEYWORDS*:* Distributed assembly flowshop, heterogeneous factories, MILP, algorithm, metaheuristic, descendent search, VND, scheduling.

DESCRIPTION:

In this Research, the Distributed Assembly Permutation Flowshop Scheduling Problem is studied considering sequence dependent setup times and heterogeneous with the objective of minimizing the makespan. For solve this problem, a MILP model and a VND-based algorithm were designed. The algorithm is composed by one stage to generate an initial set of good assembly sequences, other stage to improve the processing sequence through neighborhood structures, and finally, by a exploration stage where assembly sequences from the initial set are crossed.

The MILP model was implemented in GAMS® and validated with small instances from literature for DAPFSP-SDST with heterogeneous factories; besides, an experimental design was created to stablish incident factors on relative gap.

Moreover, the Algorithm was implemented in Matlab® and the more influent parameters were calibrated through an experimental design. Furthermore, the algorithm performance was evaluated by a comparison with MILP results in small instances with heterogeneous factories and with the best solution in the literature for big instances with homogeneous factories.

Through the experimental design it's possible conclude that jobs, products, factories and machines have a significative effect on relative gap given by gams. From the algorithm calibration, we can infer the best results for makespan are achieved with a high level of improvements and a low level of reinsertion interval. Also, we found that proposed algorithm performs better than MILP in small instances for finding similar solutions in significantly slower PC times. Finally, we found the algorithm has a similar performance than methods in literature for homogeneous factories because it is able to find better solutions in some instances, and a RPD of 0,79% in the others.

[3] Graduation Project.
[4] Faculty of physicomechanical Engineering, School f Industrial and Business Studies. Industrial Engineering. Directed by M.Sc Edwin Alberto Garavito Hernández

Introducción

En el presente proyecto se estudia el problema de Flow Shop Distribuido y Permutado con etapa de Ensamble (DAPFSP), considerando tiempos de alistamiento dependientes de la secuencia y fabricas heterogéneas. Este problema está compuesto por dos etapas, una de producción y una de ensamble. En la primera de ellas existe un número de trabajos que deben ser procesados; para esto, la etapa cuenta con dos o más fabricas paralelas que constan de dos o más maquinas dispuestas en serie y en las cuales se puede procesar cualquiera de los trabajos. Cada trabajo pertenece a una lista de requerimientos para la elaboración de alguno de los productos que se ensamblan en la segunda etapa, por lo cual sólo se puede llevar a cabo el ensamble de un producto hasta que todos los trabajos que pertenecen a éste se hayan procesado en la etapa anterior.

Las extensiones del problema consideradas en este proyecto ofrecen un acercamiento a la realidad de los sistemas productivos, como es el caso de las fábricas distribuidas, donde es común que las empresas tengan dos o más fábricas de producción en diferentes lugares con el objetivo de disminuir costos o riesgos de operación (Kahn, Castellion, & Griffin, 2004)

La consideración de tiempos de alistamiento responde a las necesidades de las organizaciones de planear sus operaciones teniendo en cuenta los tiempos no productivos en los cuales se llevan a cabo actividades de limpieza, ajuste, calibración, entre otros; en este caso se asumen como dependientes de la secuencia pues los trabajos son diferentes entre sí y cada uno podría tener su propia lista de requerimientos y necesidades.

Por último, la consideración de fábricas heterogéneas obedece a las recomendaciones hechas en la literatura por autores como Hatami, Ruiz & Romano. (2015), en las cuales manifiestan la importancia de tomar en cuenta las fabricas heterogéneas dentro de la etapa de procesamiento para hacer el problema más realista y en escenarios más complejos.

Como métodos de solución se presenta un modelo MILP y un algoritmo basado en búsqueda local descendente para el DAPFSP-SDST con fábricas heterogéneas. Para validar el modelo MILP, éste se compara con 900 instancias pequeñas de la literatura para fábricas homogéneas y para determinar los factores influyentes se utilizan instancias pequeñas para fábricas heterogéneas. Así mismo, el algoritmo se calibra con cuatro instancias a través de diseños experimentales y su desempeño se evalúa comparándolo con los resultados del MILP en instancias pequeñas y con resultados de literatura en instancias grandes para fábricas homogéneas.

En el capítulo 4 se lleva a cabo la revisión de literatura para las diferentes ramas que componen el DAPFSP-SDST desde un enfoque evolutivo. En el capítulo 6 se detallan las características del problema, así como los supuestos bajo los que se estudia. Luego, en el capítulo 7 se presenta el diseño, implementación, validación y análisis de resultados para el modelo MILP. En el capítulo 8 se encuentra el diseño e implementación del algoritmo, mientras que en los capítulos 9,10 y 11 se describe su calibración y análisis de resultados. Por último, en el capítulo 12 y 13 se presentan las conclusiones del proyecto y recomendaciones para futuras investigaciones.

Tabla 1
Cumplimiento de objetivos del proyecto

Objetivos Específicos	Cumplimiento
• Identificar la estructura del DAPFSP a partir de una revisión de literatura de las variantes del problema de *flowshop* que lo componen	Capítulo 4
• Realizar la revisión bibliográfica del problema de DAPFSP-SDST para determinar si en estudios previos se ha abordado el problema con fábricas heterogéneas.	Capítulo 4 -vi
• Diseñar el modelo de programación matemática (MILP) para el problema propuesto, implementarlo en el software GAMS y validarlo utilizando las instancias encontradas en la literatura para el problema DAPFSP-SDST con fábricas iguales.	Capítulo 7
• Diseñar un método heurístico para solucionar el problema e implementarlo en el software MATLAB.	Capítulo 8
• Evaluar el desempeño del algoritmo mediante la comparación con instancias de la literatura para fábricas homogéneas o a través de un análisis estadístico de los resultados.	Capítulo 10
• Elaborar un artículo de carácter publicable con el análisis, resultados y conclusiones del tema abordado.	

1. Planteamiento del Problema

El Problema de Flowshop Distribuido y Permutado con etapa de ensamble considerando tiempos de alistamiento dependientes de la secuencia (DAPFSP-SDST) y fábricas heterogéneas comprende dos etapas: la etapa de procesamiento que está conformada por **f** fábricas, cada una de las cuales es un sistema *flowshop* con un mismo número de máquinas **m** dispuestas en serie en las que se

procesan **n** trabajos o componentes que se envían posteriormente a una etapa de ensamble, en la que una sola máquina ensambla los trabajos para obtener **t** productos.

La solución al problema involucra determinar qué trabajos se asignan a las diferentes fábricas y su secuencia de procesamiento en ellas, así como la secuencia de ensamble de los productos, con el objetivo de minimizar el máximo tiempo de completamiento de los productos (*makespan*). Por consiguiente, el estudio de este problema está enmarcado en el área de programación de la producción pues en él se busca la optimización de recursos como las máquinas, operarios y el tiempo en el proceso de fabricación de ciertos productos.

El presente trabajo estudia el problema DAPFSP-SDST con fábricas heterogéneas, lo cual involucra que los tiempos de procesamiento y de alistamiento en las máquinas dependan de la fábrica. Esta nueva consideración se toma ante la necesidad de acercar el problema a la realidad de la industria, dado que las condiciones de operación de las máquinas u operarios pueden cambiar de acuerdo al lugar en el que éstos se encuentren. Asimismo, el problema planteado puede extenderse a otras áreas del conocimiento como la informática, en la gestión de búsquedas en sistemas de bases de datos distribuidas (Query Processing in Distributed Database System).

La complejidad de este problema, clasificada como NP-Hard, no permite obtener soluciones óptimas a partir de métodos exactos en tiempos computacionales razonables, por lo que se hace uso de métodos que permitan llegar a soluciones cercanas a la óptima. El método aproximado que se presenta en este proyecto está basado en la Búsqueda Descendente en Vecindario Variable (VND).

Dentro de la pertinencia teórica, la consideración de fábricas heterogéneas responde a recomendaciones hechas en la literatura por autores con mayor trayectoria en el estudio del

DAPFSP (Ruiz, Hatami, & Romano, 2015). Además, el planteamiento de este problema sirve como punto de referencia para futuras investigaciones donde se aborde el problema a partir de nuevas consideraciones y/o diferentes métodos de solución o en las que se busque evaluar la influencia que tiene considerar fábricas heterogéneas en el desempeño de las heurísticas y metaheurísticas.

A nivel metodológico, dado que en la literatura no se encuentran instancias para el DAPFSP-SDST con fábricas heterogéneas, en el presente proyecto se proponen y evalúan ciertas instancias adaptando las que se encuentran en la literatura para el problema con fábricas homogéneas; esto permitirá la comparación y evaluación de futuros métodos de solución planteados.

2. Justificación del Proyecto

Los procesos de apertura de mercados y globalización han llevado a las empresas a crear nuevas estrategias de producción que les permitan competir y sostenerse. Con la internacionalización de los mercados las empresas han llegado a diferentes países no sólo a ofrecer sus productos y/o servicios sino también a producirlos. Una de estas estrategias, por ejemplo, es la deslocalización industrial, en la cual empresas llevan sus fábricas a países en vía de desarrollo donde las condiciones económicas, sociales y culturales son más favorables para la empresa en comparación a su país de origen. Colombia hace parte de esta estrategia de producción con varios sectores industriales como el automotriz, que se dedican a la creación de partes o componentes, así como al ensamblaje de éstos.

Con el funcionamiento de varias fábricas en diferentes países o regiones, las empresas precisan de herramientas que les permitan programar sus recursos y actividades de una manera eficiente y eficaz de tal forma que puedan dar respuesta al mercado y cumplir con sus objetivos; sin embargo, la programación es una tarea compleja tanto en entornos productivos pequeños como en el que se analiza en el presente trabajo, donde el tamaño de las instancias y las consideraciones tomadas convierten el problema en uno de tipo NP-HARD, ante lo cual se hace necesario el desarrollo de modelos de optimización y técnicas heurísticas y metaheurísticas que generen soluciones de buena calidad en tiempos aceptables.

El DAPFSP es una representación de este tipo de industrias en donde se cuenta con más de una fábrica de producción en diferentes regiones (fabricas distribuidas) donde los procesos se hacen en un flujo unidireccional y con una sola planta de ensamble en la que los componentes o trabajos elaborados en las fábricas distribuidas se centralizan para el ensamble del producto final.

El presente trabajo se centra en la investigación del DAPFSP con tiempos de alistamiento dependientes de la secuencia (SDST) y fábricas heterogéneas[5]; esta última consideración se debe principalmente a que las características de las fábricas ubicadas en diferentes regiones del mundo varían, y aunque sean capaces de realizar las mismas tareas, las condiciones de procesamiento cambian por diferentes factores como las tecnologías utilizadas, capacitación de la mano de obra, mantenimiento de la maquinaria, entre otras. El enfoque propuesto responde a recomendaciones hechas por Ruiz et al. (2015), quienes plantean que "*Características adicionales en la formulación del problema pueden hacerlo más realístico, por ejemplo, fábricas heterogéneas distribuidas podrían tenerse en cuenta para escenarios más complejos".*

[5] Fábricas heterogéneas hace referencia a que los tiempos de procesamiento y alistamiento de cada máquina depende de la fábrica en la que se encuentre.

3. Objetivos

3.1 Objetivo General

Desarrollar un algoritmo basado en VND para la solución del problema de *Flowshop* Distribuido y Permutado con etapa de ensamble considerando tiempos de alistamiento dependientes de la secuencia (DAPFSP-SDST) y fábricas heterogéneas.

3.2 Objetivos Específicos

- Identificar la estructura del DAPFSP a partir de una revisión de literatura de las variantes del problema de *flowshop* que lo componen.
- Realizar la revisión bibliográfica del problema de DAPFSP con tiempos de alistamiento dependientes de la secuencia para determinar si en estudios previos se ha abordado el problema con fábricas heterogéneas.
- Diseñar el modelo de programación matemática (MILP) para el problema propuesto, implementarlo en el software GAMS y validarlo utilizando las instancias encontradas en la literatura para el problema DAPFSP-SDST con fábricas iguales.
- Diseñar un método heurístico para solucionar el problema e implementarlo en el software MATLAB.
- Evaluar el desempeño del algoritmo mediante la comparación con instancias de la literatura para fábricas homogéneas o a través de un análisis estadístico de los resultados.
- Elaborar un artículo de carácter publicable con el análisis, resultados y conclusiones del tema abordado.

4. Revisión de la Literatura

La revisión de literatura en un proyecto de investigación es una herramienta que permite llegar a un conocimiento más preciso y profundo sobre el estado del tema de interés y generar nuevas ideas de investigación basadas en las limitaciones y avances que presenta el conocimiento actual (Guirao Goris, 2015). Por esta razón, se realiza el siguiente análisis de la literatura referente al DAPFSP desde un enfoque evolutivo para justificar la complejidad y los métodos de solución propuestos en el presente proyecto, así como para identificar las ramas que componen el problema de estudio.

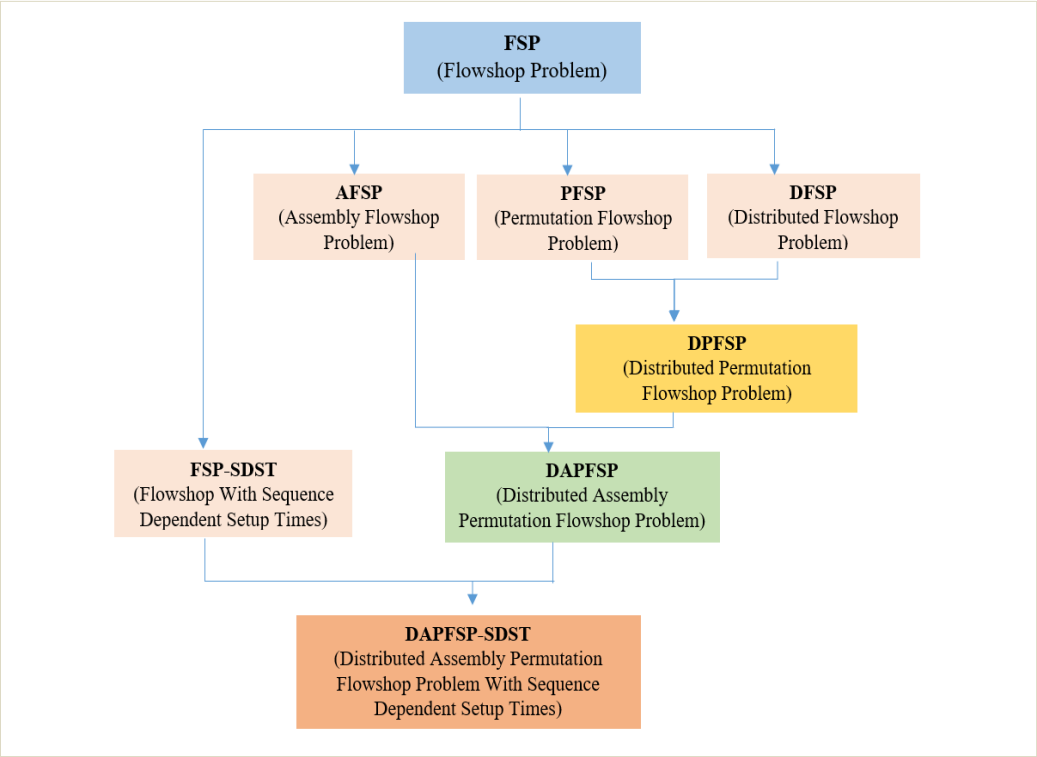


Figura 1. Líneas del Problema de Flowshop que componen el DAPFSP-SDST

La Figura 1 muestra desde un enfoque evolutivo las diferentes ramas de interés para la elaboración de este proyecto, donde se aprecia que el problema de *Flowshop* Distribuido y Permutado (DPFSP) es una generalización para el PFSP y el DFSP. En la literatura se considera que el problema de *flowshop* distribuido y permutado con etapa de ensamble (DAPFSP) es una generalización al DPFSP (Xiangtao, Xin, Minghao, & Jianan, 2015) en el que los trabajos producidos en las fábricas van a una etapa de ensamble para obtener ciertos productos.

i. Problema de Flowshop Permutado (PFSP)

El problema de *flowshop* ha sido de gran interés para los investigadores. A partir del trabajo pionero de Johnson (1954) en el que se plantea una regla para programar tareas en un flowshop con dos máquinas, se han desarrollado muchos estudios sobre *flowshop* bajo diferentes consideraciones y contextos. Entre las primeras consideraciones tomadas, incluso por Johnson, está mantener la misma secuencia de trabajos en todas las máquinas con el fin de reducir el espacio de búsqueda, siendo éste el inicio del *Flowshop* Permutado (PFSP). Desde que se demostró que éste problema es de tipo NP-Hard cuando hay más de dos máquinas (Garey, Johnson, & Sethi, 1976), se han desarrollado diversos métodos heurísticos para generar soluciones aproximadas sin exceder los tiempos computacionales; así en la última década se han propuesto más de cien nuevos algoritmos (Fernandez-Viagas, Ruiz, & Framinan, 2017), incluyendo algoritmos discretos de evolución diferencial (Pan, Tasgetiren, & Liang, 2008), Búsqueda local iterativa (Dong, Huang, & Chen, 2009) y heurísticas basadas en Algoritmos genéticos (Chen, Chen,Y., Chang, & Chen, M., 2012); sin embargo, los algoritmos que han sido identificados como los de mayor eficiencia son los basados en Algoritmos Voraces, como el formulado por Ruiz y Stützle (2007) que se destaca por haber generado mejores resultados que los algoritmos ya existentes.

ii. Problema de Flowshop Distribuido (DFSP)

Ahora bien, la mayor parte de estudios sobre programación de la producción se han desarrollado considerando una sola fábrica; no obstante, las tendencias en el mercado han generado interés en analizar la producción distribuida o en múltiples fábricas con un enfoque de tipo *flowshop* (DFSP). El primer estudio sobre programación de varias fábricas se origina en 1981 bajo un intento de unir la producción con la distribución (Williams, 1981), mientras que el primer caso aplicado surge en 1996 con la programación de un sistema de producción localizado en diferentes países que provee a consumidores europeos (Wilkinson, Cortier, Shah, & Pantelides, 1996). Posterior a estos estudios los investigadores se han enfocado en buscar mejores soluciones a través de la aplicación de diferentes heurísticas debido a que considerar varias fábricas implica un manejo de información más complejo, y por ende mayor dificultad en la programación.

La mayoría de métodos propuestos para tratar la producción distribuida se han basado en algoritmos genéticos: Jia, Nee, Fuh y Zhang (2003) presentan un Algoritmo genético modificado bajo el objetivo de minimizar el *makespan*, el costo de producción o un ponderado de ambos; los autores representan la solución factible a través de una mezcla de trabajos en forma de cromosoma y prueban la eficacia del método tanto en un problema tradicional (una fábrica), como en un caso de producción distribuida. Dos años después, Chan, Chung y Chan (2005) exponen una adaptación de Algoritmo genético en el que utilizan genes dominantes para evitar determinar la tasa de cruce y a través de las evoluciones permiten la variación de parámetros genéticos como el tamaño de la población, tasas de cruce y mutación; entre los resultados de este estudio se destaca la capacidad de los genes dominantes para mejorar el desempeño de la búsqueda.

En relación con las características de las fábricas, el 89,7% de los estudios sobre DFSP consideran homogeneidad entre éstas, por lo cual se recomienda trabajar el problema con fábricas heterogéneas para acercar el problema al contexto real (Behnamian & Fatemi Ghomi, 2016); al igual que se plantean otras futuras posibilidades de investigación como el transporte entre fábricas, interoperabilidad y buffers limitados.

iii. Flowshop Distribuido y Permutado (DPFSP)

Recientemente en 2009, se presenta por primera vez una extensión del *Flowshop* Permutado a la producción distribuida (Ruiz & Naderi, 2009); este problema se conoce como DPFSP (*Distributed Permutation Flowshop Problem)* y asume un conjunto de trabajos *n* que deben ser procesados en un conjunto de fábricas que se comportan como un *flowshop* permutado cada una. En éste trabajo seminal, Naderi y Ruiz analizan la complejidad del problema, y concluyen que ésta es de tipo NP-Hard cuando n>f, ante esto proponen varios métodos de solución basados en VND y un año después amplían su estudio con seis modelos de programación lineal entera mixta MILP, dos reglas básicas de asignación y 14 heurísticas basadas en reglas de despacho con el objetivo de optimizar el *makespan* (Ruiz & Naderi, 2010); así mismo desarrollan una evaluación experimental en la que para pequeñas instancias basadas en el trabajo de Taillard (1990) se comparan los modelos MILP con los algoritmos para evaluar su eficacia, mientras que con instancias más grandes se evalúan las heurísticas mediante el cálculo del RPD (Relative Percentage Deviation) respecto a la mejor solución obtenida; como resultado se obtienen un 54% de corridas con solución óptima y se muestra al VND como el método de mejor desempeño al obtener un RPD del 0,1 %.

Los siguientes autores en tratar el problema de DPFSP con el objetivo de minimizar el *makespan* son Liu y Gao (2010), quienes plantean un algoritmo inspirado en un mecanismo

electromagnético (EM) en el que diferentes electrones (soluciones) se atraen en función de la distancia entre ellos y la fuerza electromagnética que poseen, utilizando mecanismos de atracción-repulsión para mover los puntos hacia la optimalidad. No obstante, los autores necesitaron representar la solución en forma de permutación para poder ajustar el mecanismo al dominio discreto del problema, debido a que el EM original está fundamentado en espacios de solución continua. Así mismo, para mejorar el desempeño del algoritmo propuesto emplearon Búsqueda en Vecindario Variable (VNS) enfocada en la fábrica crítica (la que tiene mayor sub-*makespan*). El algoritmo se evalúa aplicando las mismas instancias de Taillard (1990) y se compara con los resultados de los algoritmos planteados por Ruiz y Naderi (2010); obteniendo como resultado la mejor solución conocida para 151 instancias de las 720 valoradas.

En una etapa temprana de desarrollo de algoritmos para el DPFSP, Gao y Chen (2011) plantean por primera vez un algoritmo Iterativo al formular un Algoritmo Genético Híbrido en el que se aplica búsqueda local para la exploración de soluciones en el entorno; el desempeño de éste es evaluado utilizando las instancias de Taillard (1990) y como resultado se obtiene que la combinación de AG con búsqueda local es mejor que con métodos VND al mostrar valores más bajos de RPD. Un año después, los mismos autores prueban una nueva versión de Algoritmo genético basado en el conocimiento, representado a través de secuencias parciales de trabajos y operadores de infección que reparan individuos débiles dentro del algoritmo (Gao, Chen, & Liu, 2012); ésta segunda versión logra superar el desempeño del Algoritmo Genético híbrido y los de búsqueda local mencionados anteriormente.

Demostrando un interés constante en la temática de DPFSP, Gao y Chen (2013) presentan un nuevo método de solución con un enfoque diferente, esta vez el algoritmo está construido a partir de Búsqueda Tabú y no de Algoritmo genético. Para lograr una comparación más estricta con el

algoritmo planteado en 2011, los algoritmos se ejecutan en el mismo equipo computacional bajo un lenguaje de programación C++. Los resultados arrojan que el nuevo enfoque propuesto permite superar el desempeño no sólo del algoritmo genético híbrido, sino de todos los algoritmos antes propuestos, la mayoría basados en algoritmos genéticos; de esta manera se actualiza la mejor solución conocida para 420 instancias de un total de 720.

Cinco años después del trabajo pionero sobre DPFSP, Ruiz y Naderi (2014) regresan con una formulación basada en Búsqueda Dispersa con la que buscan evitar la aleatoriedad que presentan algoritmos evolutivos como el genético dado el bajo desempeño que han mostrado en estudios anteriores. Para comparar el algoritmo propuesto con los anteriores, los autores codifican todos los algoritmos bajo un mismo lenguaje siguiendo la lógica de los autores originales; de esta forma evalúan estadísticamente 11 métodos heurísticos en 720 instancias usando diseño de experimentos y análisis de Varianza, los cuales demuestran que las soluciones obtenidas por Búsqueda dispersa superan a las de todos los algoritmos planteados previamente en todas las 720 instancias.

El estudio más reciente encontrado en la literatura sobre DPFSP es el propuesto por Li, Duan, Ji y Yang (2016); en éste se incluye una nueva complejidad al problema: considerar transporte entre fábricas con tiempos determinísticos y capacidad de carga limitada para los vehículos. Esta consideración responde a recomendaciones hechas por otros autores como Behnamian y Fatemi-Ghomi (2016), quienes encuentran importante trabajar el problema con interoperabilidad y transporte entre fábricas; sin embargo, otras sugerencias como las fábricas heterogéneas aún no se han sido objeto de estudio, lo cual resalta al igual que en el DFSP, la oportunidad de investigación.

iv. Flowshop con Etapa de Ensamble (AFSP)

La última rama del problema de *flowshop* que hace parte de la estructura del DAPFSP es el

flowshop con etapa de ensamble (AFSP); en este problema se distinguen dos etapas: la primera, en la que se producen los componentes y la segunda, en la que éstos se ensamblan para formar productos finales. En la literatura encontrada, el primer artículo donde se relaciona la producción con el ensamble es el de Kusiak (1988); éste se contextualiza en un sistema de manufactura flexible en el que se mecanizan ciertas piezas que luego son llevadas por un vehículo de guiado automático a un sistema de ensamble para formar un producto.

Sin embargo, la primera investigación hallada sobre AFSP es la de Lee, Cheng y Lin (1993), quienes inspirados en una planta de ensamble de carros de bomberos, plantean un problema en el que dos tipos de trabajos (cuerpo y chasis) son producidos en dos máquinas paralelas y luego ensamblados en una máquina para formar diferentes tipos de vehículos, éste se conoce como *The two-stage Assemby Problem (2-SAFSP)*. En el contexto propuesto por los autores, la producción sigue un conjunto de órdenes que debe ser entregado en el menor tiempo posible, razón por la cual se considera la minimización del *makespan* como función objetivo. Además, se presenta una demostración de que el problema es NP-completo y en respuesta a ello se proponen tres heurísticas para hallar soluciones aproximadas las cuales representan el primer intento de tratar el AFSP a través de métodos heurísticos.

Como una ampliación al problema propuesto por Lee et al. (1993) en el que se consideraron 2 máquinas paralelas en la primera etapa; Potts, S.V.Sebast'janov, Van Wassenhove y Zwaneveld (1995) consideran *m* máquinas paralelas teniendo en cuenta la aplicación del AFSP en la manufactura de computadoras; en ésta, varias partes como procesadores, discos duros, mouse y teclados se producen en máquinas diferentes para luego ser ensambladas en una estación de acuerdo a los requerimientos de los clientes. La solución de este problema persigue el objetivo de completar todos los productos en el menor tiempo posible, esto es, minimizar el *makespan*; para

ello se propone una técnica compacta de suma vectorial con garantía de desempeño en el peor caso y dos años después se propone un algoritmo Branch & Bound en el que se incorporan varias reglas de dominancia (Potts & Hariri, 1997).

Dadas las condiciones reales en las que operan los sistemas productivos, además de una etapa de producción y una de ensamble, Koulamas y Kyparisis (2001) consideraron útil analizar el problema de Potts et al. (1995) con una etapa intermedia por la cual se transporten las partes producidas hacia una siguiente estación o fábrica encargada del ensamble; este problema es conocido en la literatura como *Three-Stage Assembly Problem (3-SAFSP).*

En los años siguientes al trabajo de Lee et al. (1993), se proponen varias heurísticas para solucionar el problema 2-SAFSP bajo diferentes objetivos de desempeño. Al-Anzi y Allahverdi (2006) desarrollan tres heurísticas: la primera basada en Recocido Simulado (SA), la segunda en Búsqueda Tabú (TS) y la tercera es un híbrido de las dos anteriores en el cual se permiten intercambios que no están permitidos en la lista tabú, es decir, se utiliza el enfoque del recocido simulado para que exista una probabilidad de intercambio con soluciones que no son las mejores dentro del entorno de búsqueda. En el mismo año, enfocados en la aplicación del 2-SAFSP a sistemas con bases de datos distribuidas, los autores presentan dos heurísticas basadas en Búsqueda Tabú y en Optimización por Enjambre de Partículas (PSO) utilizando un objetivo de desempeño basado en la fecha límite de entrega (Al-Anzi & Allahverdi, 2006). Las dos heurísticas se comparan por medio de experimentos computacionales usando una solución aleatoria; en los resultados se encuentra que la heurística basada en PSO, utilizada por primera vez para resolver un problema de programación supera a las demás cuando el rango de fechas de entrega es corto y el número de componentes *n* a producir es pequeño, mientras que la Búsqueda Tabú es mejor para *n* grandes.

Hasta este punto, los estudios nombrados sobre AFSP han considerado máquinas paralelas en la etapa de producción. Sin embargo, Yokoyama (2008) propone una nueva situación en la cual la producción se lleva a cabo en un *flowshop* compuesto por múltiples máquinas en las que se toma en cuenta tanto tiempos de operación como de ajuste (setup times). Para solucionar el problema, el autor presenta un procedimiento de programación dinámica y un método Branch & Bound. Este estudio concierne al presente proyecto de investigación en la medida que aplica el concepto de *flowshop* en la etapa de producción.

Ahora bien, para acercar el AFSP a la realidad de la industria, los autores han visto la necesidad de considerar tiempos de alistamientos dependientes de la secuencia (SDST) en los problemas de *flowshop* con etapa de ensamble; es así como desde 2009 se encuentra una serie de estudios con éste nuevo enfoque. Por ejemplo, Maboudian y Shafaei (2009) formulan un modelo matemático no lineal con el bi-objetivo de minimizar el *makespan* y la máxima tardanza para un problema de AFSP en el que se consideran SDST en ambas etapas; para validar el desempeño del modelo se resuelven instancias pequeñas usando el software Lingo 8.0, el cual utiliza método Branch & Bound para la búsqueda de soluciones. Los resultados revelan que el modelo matemático no promete soluciones óptimas aún con grandes tiempos computacionales por lo que los autores sugieren la implementación de heurísticas que resuelvan el problema de una manera más eficiente.

En otro estudio referente a SDST, Maboudian, Tavakkoli-M, Hatami, y Ebrahimnejad (2010) consideran importante involucrar tiempos de alistamiento dependientes de la secuencia en la etapa de producción de un 3-SAFSP. En éste, con el objetivo de minimizar el tiempo medio de flujo y la máxima tardanza se plantea un modelo no-lineal y se desarrollan dos heurísticas basadas en Recocido Simulado y Búsqueda Tabú, las cuales inicializan la búsqueda con dos soluciones obtenidas con las reglas SPT (tiempo de procesamiento más corto) y EDD (Fecha de entrega más

cercana). Para evaluar la eficiencia de las heurísticas se plantean dos medidas de desempeño: porcentaje de error promedio y desviación estándar; de lo cual se obtiene un mejor desempeño para la heurística basada en recocido simulado. Por otra parte, los autores recomiendan considerar buffers de capacidad limitada entre las etapas.

Como se ha visto, el problema de AFSP ha evolucionado en complejidad a medida que se intenta adaptar a la realidad. En el principio sólo se consideraba una máquina en la producción y una máquina de ensamble, luego se amplió a *m* máquinas paralelas, luego a *m* máquinas en serie (*flowshop*) y después se agregó SDST a las etapas de producción y de ensamble. No obstante, ninguna de las investigaciones encontradas en la literatura había considerado más de una máquina de ensamble hasta el estudio de Fatemi Ghomi, Jolai, Mozdgir y Navaei (2013), quienes guiados por sistemas productivos reales como las ensambladoras de automóviles, en los que se utiliza más de una máquina o estación de ensamble, proponen una generalización del problema 2-SAPFSP en la que se consideran *k* máquinas de ensamble no idénticas en la segunda etapa, además de tiempos de alistamiento dependientes de la secuencia en la etapa de producción. Para resolver este problema, los autores desarrollan un modelo de programación lineal entera mixta y un algoritmo híbrido basado en VNS con el objetivo de minimizar la suma ponderada de *makespan* y tiempo medio de flujo; así mismo, en un estudio posterior proponen 4 metaheurísticas basadas en Recocido Simulado y Algoritmo competitivo Imperialista con el objetivo de minimizar los costos por mantenimiento de inventario y esperas (Fatemi-Ghomi et al., 2014). Entre las heurísticas planteadas se destacan el Recocido Simulado y el híbrido de VNS por sus bajos tiempos de corrida.

En los últimos tres años se han propuesto diversos algoritmos para resolver problemas de tipo 2-SAFSP y 3-SAFSP con diferentes funciones objetivo (tabla 2); aun así, no existen instancias ni métodos estadísticos o computacionales definidos que permitan una comparación objetiva entre

ellas. Tampoco se encuentra una revisión de literatura que ayude a establecer las tendencias en la investigación del problema, recomendaciones y resultados logrados.

Tabla 2.
Algoritmos propuestos en los últimos tres años para resolver problemas de tipo 2-SAFSP y 3-SAFSP

AÑO	PROBLEMA	FUNCION OBJETIVO	ALGORITMO PROPUESTO
2014	Two-stage Assembly Flowshop scheduling problem**(2-SAFSP)**	MIN (Makespan, maximum earliness, maximum tardiness)	Hybrid electromagnetism-like algorithm
		MIN (Makespan, mean completion time)	Imperialist Competitive Algorithm
		MIN (Maximum earliness, maximum tardiness)	➢ Variable neighborhood search. ➢ Hybrid electromagnetism-like algorithm.
	Three-stage Assembly flowshop scheduling problem**(3-SAFSP)**	MIN (Total flow time, total tardiness)	Elitist Non-Dominated Sorting Genetic Algorithm (NSGA).
2015	Two-stage Assembly Flowshop scheduling problem**(2-SAFSP)**	MIN Total tardiness	➢ Insertion Algorithm ➢ Genetic Algorithm ➢ Simulated Annealing
	(3-SAFSP)-SDST	MIN (Average completion time, maximum tardiness)	Adaptive Hybrid Estimation of distribution Algorithm
2016	**(2-SAFSP)**	MIN Makespan	Artificial Immune Systems
		MIN Tardy jobs	➢ Genetic Algorithm ➢ Improved Genetic Algorithm ➢ Simulated Annealing
	(2-SAPSP) with Preventive maintenance	MIN (Makespan, Unavailability of system)	➢ Mixed-Integer linear programming. ➢ Imperialist Competitive Algorithm ➢ Non-dominated Ranking GA.

v. Flowshop Distribuido y Permutado con Etapa de Ensamble (DAPFSP)

A diferencia de problemas de *flowshop* de ensamble como el 2-SAPFSP y el 3-SAPFSP en los que la etapa de producción está compuesta por m máquinas paralelas, cada una de las cuales puede procesar sólo un tipo de trabajo, en el DAPFSP la producción está distribuida; es decir, no existen

máquinas paralelas sino *flowshop* (fábricas) paralelos que pueden procesar cualquier tipo de trabajo.

Los primeros esfuerzos en analizar el DAPFSP surgen en los años ochenta, entorno al proceso de deslocalización industrial de grandes fábricas japonesas que debido a un fuerte aumento de los costos laborales internos optaron por distribuir sus instalaciones en varios países asiáticos (Douglas & Francisco, 1996). La operación e integración de las diversas fábricas distribuidas generaron varios retos para los investigadores, quienes a través de diversos enfoques y planteamientos buscan asegurar el buen funcionamiento de estos complejos sistemas de producción, además de proponer sistemas que apoyen la toma de decisiones.

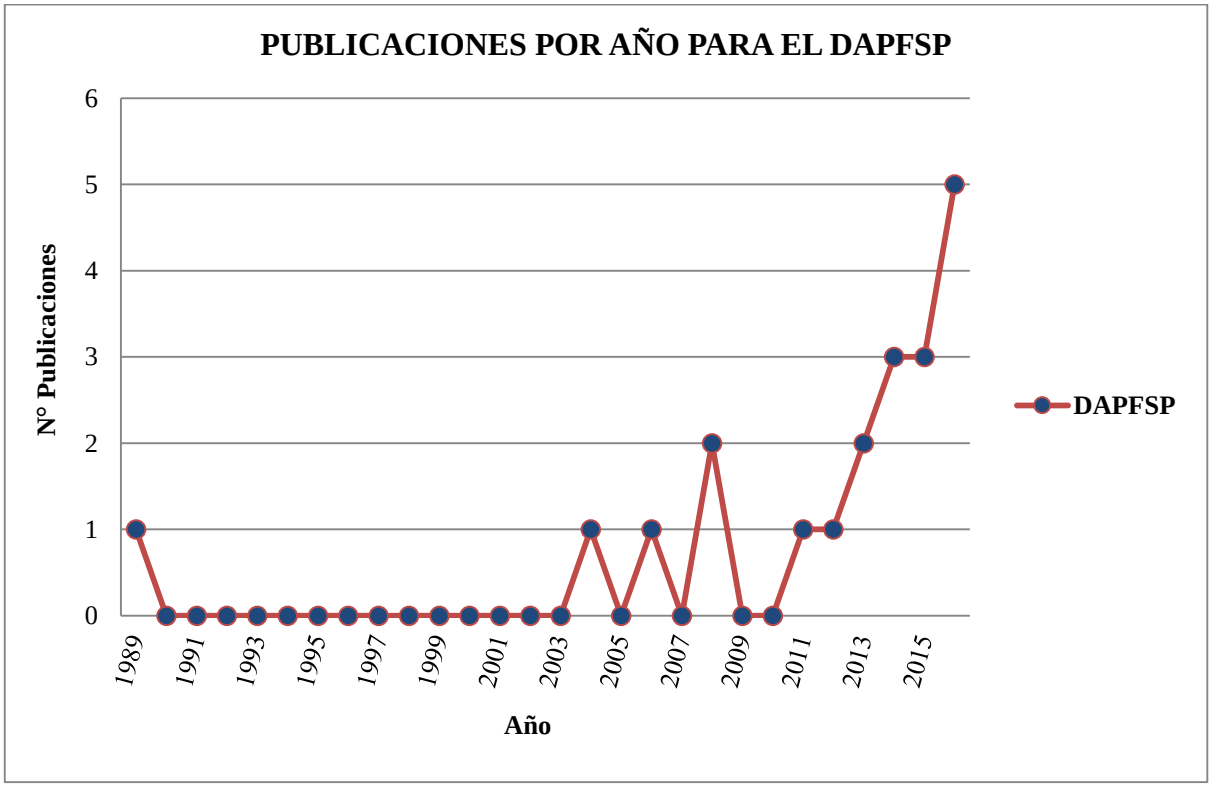


Figura 2. Número de publicaciones por año para el DAPFSP.

Como se evidenció en el análisis bibliométrico (apéndice A) y en la figura 2, más del 70% de las publicaciones sobre el DAPFSP han sido realizadas a partir el 2011; es decir; su estudio es

reciente y está focalizado en los últimos seis años; sin embargo, se considera de relevancia para la comunidad científica pues los autores que mayor atención han puesto en el estudio de este problema tienen una gran trayectoria investigativa reflejada en el índice-h de Hirsch (tabla 3).

Tabla 3.
Índice-h para los autores con más de dos artículos publicados en DAPFSP.

Autor	*h-index*	N° documentos
Ling Wang	44	363
Rubén Ruiz	29	77
Ali Allahverdi	26	98
Shengyao Wang	12	47
Fawaz S Al-Anzi,	11	44

Nota. Obtenido de Scopus. El índice-h se define como el número de artículos publicados h que han sido citados al menos h veces (Hirsch, 2005).

El primer estudio que se encuentra en la literatura sobre producción distribuida con etapa de ensamble propone la programación orientada a objetos como un método para la planeación de la producción con el que se busca minimizar el costo total de producción e inventario de acuerdo a lo que implica tomar la decisión correcta en términos de ingresos y ganancias (Matz & Terry, 1989). Más adelante, ante la necesidad de controlar la calidad de los componentes que se producen en diferentes fábricas, se plantea un sistema de información para el control de la calidad en tiempo real (Mahdavi, Shirazi, Cho, Sahebjamnia, & Ghobadi, 2008), asumiendo un reto diferente para la operación de este tipo de sistemas de producción.

Si bien en los anteriores estudios se proponen dos enfoques diferentes para entender el DAPFSP, el primer estudio en darle un nombre a este tipo de sistema de producción y en especificar a nivel conceptual las consideraciones que lo conforman es el de Ruiz, Hatami y

Romano (2013). En éste se presenta el DAPFSP como una generalización del DPFSP en el que la primera etapa está compuesta por *f* fábricas de producción idénticas dispuestas de manera paralela, siendo cada una un *flowshop* permutado que produce ciertos trabajos que serán ensamblados en una sola máquina para conformar los productos finales. Dado que la complejidad del DAPFSP es de tipo NP-Hard, los autores proponen un modelo de programación lineal entera mixta (MILP) basado en el quinto modelo de los seis propuestos en 2010 para el DPFSP con el objetivo de minimizar el *makespan*; además, un algoritmo basado en VND, tres algoritmos constructivos y dos reglas de asignación de trabajos. Los métodos propuestos se evalúan computacionalmente, donde el RPD y tiempo computacional promedio obtenidos se analizan por medio de un análisis estadístico ANOVA, el cual muestra un mejor desempeño para los algoritmos basados en VND que para los constructivos. Para futuras investigaciones los autores recomiendan considerar diferentes estrategias para la creación de estructuras de vecindario en el VND, así como etapas de transporte, tiempos de alistamiento, fábricas heterogéneas e implementación de diferentes metaheurísticas.

A partir del estudio de Ruiz et al. (2013) se han planteado distintos algoritmos con el objetivo de minimizar el *makespan*; en (Li, Zhang, Minghao, & Wang, 2015) por ejemplo, se presenta un algoritmo genético en el que se emplean operadores de cruce modificados y tres tipos de búsqueda local: los primeros con el fin de mejorar la generación de individuos y los segundos para mejorar la calidad de cada solución después del cruce, ante el hecho de que en el trabajo de Ruiz et al. (2013) el uso de heurísticas constructivas no posibilitó la mejora de las soluciones. Las cuatro versiones del algoritmo genético: una sin búsqueda local y las otras tres con una búsqueda local diferente cada una obtuvieron mejores soluciones que los algoritmos propuestos anteriormente después de compararlos mediante RPD; sin embargo, los autores recomiendan mejorar las

estrategias de búsqueda local limitándolas a grupos de trabajos o a cambiar el orden en el que se ensamblan los productos.

Ahora bien, en el último año se han propuesto dos algoritmos para resolver el DAPFSP: un algoritmo memético basado en Búsqueda de Vecindario Variable (Liu, Wang, & Zhang, 2016) y un algoritmo de optimización híbrido basado en Biogeografía (Lin & Zhang, 2016) .Sin embargo, los resultados del primero no son comparables con los de otros algoritmos dado que se utilizaron instancias diferentes a las propuestas por Ruiz et al. (2013); en tanto que el segundo, después de compararse con los algoritmos planteados permitió la actualización de la mejor solución conocida para el *makespan* en 71 instancias pequeñas y 91 instancias grandes, es decir; es el que ha logrado mejores resultados entre los algoritmos que han usado las mismas instancias.

Para resumir, si bien el DAPFSP es un problema reciente, los estudios para resolverlo han crecido desde su planteamiento en el trabajo de Ruiz et al. (2013), todos ellos enfocados a minimizar el *makespan*, ante lo cual se podrían explorar otros objetivos de optimización. También cabe resaltar la importancia metodológica que tiene el uso de las instancias propuestas en 2013 ya que permiten la comparación entre los algoritmos, bien sea por RPD o tiempo computacional; además, es claro que existen diferentes posibilidades para investigaciones futuras sea con la toma de consideraciones de mayor complejidad o en el planteamiento de nuevas heurísticas o estrategias.

vi. Flowshop Distribuido y Permutado con Etapa de ensamble considerando tiempos de alistamiento dependientes de la secuencia (DAPFSP-SDST)

Un año después de presentar su trabajo seminal sobre DAPFSP en 2013, Ruiz et al. toman el problema que habían planteado y en respuesta a una de sus recomendaciones le adiciona una nueva

consideración: tiempos de alistamiento dependientes de la secuencia (SDST) en las etapas de producción y ensamble con el fin de adaptarlo más a la realidad de los sistemas productivos (Ruiz, Hatami, & Romano, 2014). Para solucionar este problema, se hace uso de las mismas reglas de asignación de trabajos que habían propuesto en 2013 y se presentan dos heurísticas sencillas: la primera está basada en la generación de una secuencia de productos con el menor tiempo de completamiento de ensamble mientras que la segunda da prioridad a los productos conformados por los trabajos que se produzcan más rápido en la etapa de producción. Combinando las dos heurísticas con las dos reglas de asignación resultan cuatro algoritmos que son evaluados a través de 900 instancias pequeñas y 270 instancias grandes, y dado que en la literatura no se encontraban instancias de referencia, los tiempos de procesamiento, ensamble y alistamiento se generan de manera aleatoria. Para la evaluación de los algoritmos propuestos se calcula el porcentaje de desviación relativa (RPD) sobre la mejor solución conocida y se realiza un análisis ANOVA para determinar si las diferencias son estadísticamente significativas; éste último expone un mejor desempeño para la segunda heurística en instancias pequeñas, mientras que en instancias grandes el desempeño depende del rango de los tiempos de alistamiento.

El segundo y más reciente estudio sobre este problema lo desarrollan Ruiz et al. (2015) profundizando en las heurísticas sencillas que habían planteado en 2013 y proponiendo dos metaheurísticas: la primera basada en el algoritmo de Búsqueda Descendente en Vecindario Variable (VND) y la segunda en un Algoritmo Voraz. Los algoritmos se calibran y analizan a través de un diseño de experimentos, mientras que su desempeño se evalúa por medio de amplios experimentos computacionales y estadísticos con los mismos tamaños de instancias antes utilizados; se obtiene que las heurísticas constructivas simples generan buenos resultados de

manera instantánea mientras que las basadas en VND y algoritmo voraz tardan más tiempo en generar buenos resultados dada su búsqueda por resultados cercanos a la optimalidad.

Para concluir, se resaltan las oportunidades que existen para la investigación del DAPFSP-SDST dado que en la literatura se encuentran sólo dos estudios relacionados con el problema; así la investigación podría enfocarse por ejemplo en el planteamiento de diferentes metaheurísticas, funciones objetivo, así como en la consideración de fábricas heterogéneas o transporte entre etapas, de las cuales no se encuentran precedentes de su estudio en la literatura encontrada, además de que son constantemente recomendadas por los autores. Otro punto a tener en cuenta es la importancia que tienen las instancias propuestas por Ruiz et al. (2013) pues tomarlas en estudios posteriores con métodos de evaluación de desempeño similares permitirá la comparación entre los diferentes métodos de solución planteados.

5. Marco Teórico

5.1. Problema de Flowshop (FSP)

El *flowshop* es una clase de proceso productivo en el que las piezas o trabajos tienen una misma ruta de procesamiento; es decir, un conjunto *n* de trabajos deben procesarse en un conjunto *m* de máquinas en un mismo orden. Éste se caracteriza porque las máquinas están dispuestas para facilitar un flujo unidireccional de los trabajos. El objetivo de este problema es generar una secuencia de trabajos que permita optimizar uno o varios criterios.

En la formulación del problema de f*lowshop* tradicional se encuentran las siguientes consideraciones (Gupta, 1979):

- *Respecto a los trabajos*
  - Todos los trabajos están disponibles en el tiempo cero.
  - Cada trabajo puede tener su propia fecha de vencimiento, la cual es fija.
  - Cada trabajo es independiente del otro.
  - Cada trabajo consiste de ciertas operaciones, cada una de las cuales se ejecuta por sólo una máquina.
  - Todos los trabajos tienen un mismo orden de procesamiento.
  - El tiempo de procesamiento para cada trabajo es determinístico y es conocido.
  - Cada trabajo puede ser procesado por una máquina a la vez.
  - Los trabajos pueden esperar su procesamiento antes de una máquina, es decir, se permite inventario en proceso.
- *Respecto a las máquinas*
  - Cada etapa consiste de sólo una máquina.
  - Cada máquina está desocupada al inicio del periodo de programación
  - Las máquinas son independientes entre sí, por lo que pueden operar a su propio ritmo.
  - Cada máquina puede procesar máximo un trabajo al tiempo.
  - No existen interrupciones ni daños en las máquinas en el periodo programado.
- *Respecto a políticas de operación*
  - Las máquinas no pueden permanecer ociosas si hay un trabajo esperando.
  - Cada trabajo se considera una entidad indivisible.

- No se permite cancelar el procesamiento de un trabajo si éste ya se inició.
- No existen trabajos prioritarios.
- Un trabajo no se puede procesar en más de una máquina al tiempo.
- Las máquinas no pueden utilizarse para otro propósito que la producción programada.
- Los trabajos se procesan en la misma secuencia, es decir; no se permiten saltos ni adelanto de trabajos.

El problema de *Flowshop* se ha modificado y adaptado de acuerdo con las necesidades de investigación que han surgido. A continuación, se presentan algunas de las variantes que existen.

5.1.1. Flowshop Permutado (PFSP). El Problema de *Flowshop* permutado es un problema de optimización combinatoria en el que un conjunto de *n* trabajos debe procesarse en un conjunto de *m* máquinas. Como una simplificación al problema tradicional de *Flowshop*, los trabajos se procesan en el mismo orden en todas las máquinas, lo que quiere decir que la secuencia de la primera máquina se mantiene para todas las siguientes. El objetivo es encontrar una permutación de trabajos que optimice un determinado criterio, que tradicionalmente es el *makespan*.

5.1.2. Flowshop Distribuido y Permutado (DPFSP). Los sistemas distribuidos son aquellos que se componen de más de una fábrica o centro de producción. En el DPFSP un grupo de *n* trabajos debe procesarse en *f* fábricas, cada una de las cuales es un *Flowshop* Permutado que contiene un mismo número *m* de máquinas. Cada trabajo consta de *m* operaciones que deben ejecutarse una después de la otra y que se procesan en un tiempo determinado. Este problema se puede denotar como DF/prmu/Cmax si el criterio que se pretende optimizar es el *makespan*.

Dado que este problema representa una ampliación al FSP tradicional, nuevas consideraciones son tenidas en cuenta:

- Si un trabajo es asignado a una fábrica, éste no puede ser transferido a otra.
- Todas las fábricas pueden procesar todos los trabajos.
- Todas las fábricas tienen un mismo número de máquinas *m.*

Dada la presencia de varias fábricas, este tipo de sistema productivo involucra la toma de dos tipos de decisiones: i) asignar los trabajos a las fábricas y ii) la programación de cada fábrica en función de los trabajos que se le hayan asignado.

El número de soluciones posibles para el DPFSP con fábricas homogéneas se puede calcular a través de la multiplicación del número de formas en la que se pueden particionar *n* trabajos en *f* fábricas por el número de combinaciones de trabajos para una determinada partición. Así, el número de soluciones para *n* y *f* fábricas está dado por la siguiente formula (Ruiz & Naderi, 2010):

$$n! \binom{n+f-1}{f-1}$$

5.1.3. Flowshop Distribuido y Permutado con etapa de ensamble (DAPFSP). El *Flowshop* con etapa de ensamble (AFSP) es un sistema de producción híbrido en el que las operaciones necesarias para la generación de ciertas piezas se llevan a cabo en una etapa independiente y luego las piezas ya producidas son enviadas a una etapa de ensamble (Koulamas & Kyparisis, 2001). De manera similar, el DAPFSP también está compuesto por dos etapas: Producción y ensamble; sin embargo, se diferencia de un AFSP en que las *f* fábricas de la etapa de producción son cada una un *Flowshop* Permutado (PFSP). Por otro lado, la etapa de ensamble se

compone de una sola máquina en la que se obtienen *t* productos finales de acuerdo a un programa de ensamble que especifica las partes que corresponden a cada producto (Ruiz et al., 2013).

Aparte de las consideraciones tomadas en el problema de *Flowshop* tradicional y el DPFSP; para el DAPFSP se incluyen otras relacionadas con el proceso de ensamble:

- Cada producto se compone de un determinado conjunto de trabajos.
- Cada trabajo sólo pertenece a un producto.
- El producto sólo puede empezar a ensamblarse cuando todos los trabajos que lo componen se han terminado de procesar en las fábricas.

5.2. Optimización

Es un procedimiento versátil que puede ser usado en diferentes campos de aplicación como la informática, estadística, matemáticas, economía entre otras. El problema de optimización consiste en calcular el conjunto de variables (denominadas de decisión), tales que minimizan o maximizan una cierta función objetivo.

Un problema es de optimización si hay muchas posibles soluciones y entre ellas un conjunto de soluciones factibles que pueden compararse de alguna manera, además puede ser formulado según la siguiente ecuación (Moratto & Pérez, 2016):

$$P = \begin{Bmatrix} Opt\ f(x) \\ sujeto\ a: \quad X \in F \subset SS \end{Bmatrix}$$

Donde $f(x)$ es la función a optimizar, F es el conjunto de soluciones factibles y SS es el espacio de soluciones.

Existen dos categorías de problemas de optimización: aquellos con solución codificada por valores reales y la optimización combinatoria en la que la solución está codificada por valores enteros.

5.2.1. Optimización combinatoria. Consiste en encontrar el máximo o mínimo de una determinada función sobre un conjunto finito de soluciones *S* por lo que las variables deben ser discretas. Los problemas de este tipo se distinguen porque siempre existe un algoritmo exacto que permite obtener la solución óptima.

Un problema de optimización combinatoria está definido por los siguientes elementos (Blum & Roli, 2003):

- Un conjunto de variables X= $\{X_1, X_2, \ldots, X_n\}$
- Los dominios de las variables $D_1, \ldots, D_n$
- Restricciones sobre las variables.
- Una función objetivo *f* que minimizarse (o maximizarse).

El conjunto de posibles soluciones factibles es:

$$S = \{s = \{(X_1, V_1), \ldots, (X_n, V_n)\} \mid V_i \in D_i, s\ que\ satisface\ todas\ las\ restricciones\}$$

Así, para resolver el problema de optimización combinatoria se debe encontrar una solución $s^* \in$ S con el que se logra un valor mínimo o máximo (según sea el caso) de la función objetivo, s^* es el óptimo global:

$$f(s^*) \leq f(\mathrm{s}) \qquad \forall_s \in S$$

5.3. Complejidad Computacional

Dentro de la teoría de complejidad computacional los problemas de optimización son clasificados según su complejidad, la cual está basada en el tiempo que le toma a un algoritmo encontrar la solución al problema.

Dentro de las clases de complejidad se encuentran las siguientes:

- **P**: Está compuesta por todos aquellos problemas de decisión que pueden ser resueltos en una máquina de Turing determinística en un periodo de tiempo polinómico proporcional a los datos de entrada. De manera general se puede decir que P corresponde a la clase de problemas que se pueden resolver en una computadora tradicional.

- **NP**: En este tipo de complejidad se incluyen aquellos problemas que no pueden resolverse en un tiempo polinomial o que probablemente tienen algoritmos en tiempo polinomial basado en principios que por ahora son desconocidos para la comunidad científica. Este conjunto de problemas puede ser resuelto en tiempo polinómico por una máquina de Turing no determinista, sin embargo, esta máquina es una idealización y hasta la fecha no se ha creado el primer algoritmo no determinístico capaz de resolver los problemas NP en tiempos polinómicos (Garey & Johnson, 1979). Los problemas de este tipo son verificables en tiempos polinómicos, es decir, dada una posible solución para una instancia, es posible comprobar que es válida en un tiempo n^k.

- **NP-Complete**: Este tipo de complejidad es un subconjunto de NP y los problemas de este tipo se caracterizan por ser los de mayor complejidad dentro de su grupo ya que es menos probable que se pueda encontrar una solución en tiempo polinómico para éstos dado que cualquier problema en NP se puede reducir a cada uno de los problemas de NP-Complete. Si se lograra tener una

solución polinómica para un problema NP-Complete, todos los problemas en NP tendrían también una solución en tiempo polinómico y por lo tanto $P = NP$ (Garey & Johnson, 1979).

- **NP-Hard**: Dentro de esta clasificación se encuentran aquellos problemas que son por lo menos tan difíciles como los NP pero que no se sabe a ciencia cierta su complejidad, por lo cual pueden ser más difíciles de resolver que los problemas de tipo NP-Complete. De la misma manera que los NP-Complete, si se encuentra un algoritmo que trabaje en tiempo polinómico para resolver un NP-Hard, entonces es posible construir un algoritmo que trabaje en tiempo polinómico para cualquier problema en NP (Garey & Johnson, 1979).

Los problemas de tipo NP-Hard se caracterizan porque el tiempo de ejecución de un algoritmo que intenta solucionarlo, aumenta de manera exponencial respecto al tamaño de problema.

5.4. Programación Entera y Entera Mixta

Aplica para problemas en los que todas o algunas variables deben tomar obligatoriamente valores enteros o binarios en algunos casos; por su parte, en la programación lineal entera mixta (MILP) las variables pueden ser enteras no negativas y continuas. La dificultad de esta programación se encuentra en la gran cantidad de variables que presentan la mayoría de problemas; para resolverla los métodos más conocidos son el Branch and Bound, Branch and Cut y Relajación Lagrangiana.

5.5. Métodos de Solución

5.5.1. Métodos exactos. Estos métodos permiten obtener la mejor de todas las soluciones posibles; es decir, la solución óptima. Cuando se busca solucionar problemas de programación o *scheduling* se pensaría que lo ideal es encontrar el óptimo, sin embargo, esto involucra tiempos

computacionales muy altos por lo que sólo se hace posible para problemas con instancias pequeñas, numerosas variables continuas y funciones objetivo simples.

5.5.1.1. Branch and Bound. Es una búsqueda estructurada en la que el espacio de soluciones es dividido en subproblemas o ramas cada vez más pequeñas (Branch), las cuales se van excluyendo de la búsqueda (Bound) después de cada partición cuando exceden el costo de una solución conocida.

Este método resuelve la relajación lineal de uno de los subproblemas y si la solución obtenida es entera, entonces la rama deja de analizarse puesto que tal solución es también la óptima en la programación entera. En el caso de que la solución no sea entera el subproblema se divide en otros dos subproblemas y la subdivisión ocurre cada vez que un nodo encuentra una solución no entera. El procedimiento finaliza cuando todos los nodos del árbol tienen una solución entera o bien una solución no entera que es mayor a las soluciones enteras de otros nodos. El nodo con la mejor solución entera provee una solución óptima al programa entero original.

La utilidad de este método se encuentra en el hecho de que sólo se enumera una pequeña parte de las soluciones pues el resto son eliminadas al establecerse que no pueden ser óptimas (Lawler & Wood, 1966).

5.5.1.2. Planos de corte. Este método consiste en resolver una serie de problemas relajados linealmente restringiendo cada vez más la región factible, hasta alcanzar una solución óptima con valores enteros; así, si la solución óptima del problema relajado no tiene todas sus componentes enteras, es posible obtener una desigualdad válida (corte) para la región factible de la programación entera.

Un algoritmo básico de planos de corte en un primer paso relaja las condiciones de integralidad sobre las variables y resuelve el programa lineal resultante; si el programa lineal no es factible, el programa entero también lo es. Cuando la solución óptima del programa lineal cumple las condiciones de integralidad se ha encontrado un óptimo del problema; en caso contrario, se busca identificar desigualdades lineales que estén violadas por la solución fraccionaria del programa lineal y sean válidas para los puntos enteros factibles.

El algoritmo continúa hasta que:

- Se encuentra una solución entera.
- El programa lineal es infactible, lo que significa que el problema entero es infactible.
- No se pudo identificar alguna desigualdad lineal.

El éxito de este procedimiento depende en gran medida de la posibilidad y la eficiencia de encontrar desigualdades violadas (planos de corte) que puedan ser agregadas a la formulación para separar las soluciones fraccionarias.

5.5.1.3. Branch and Cut. Éste combina el método de Branch and Bound con el de planos de corte, es decir, permite resolver problemas de programación entera mediante la relajación de las restricciones de integralidad y la acotación del espacio de búsqueda para impedir soluciones no enteras (planos de corte).

Su procedimiento se basa en que, si después de resolver una relajación lineal de un subproblema o rama no se puede cerrar el nodo correspondiente, entonces se busca una desigualdad violada (corte) por el óptimo de relajación; si se encuentran desigualdades violadas, éstas se agregan a la formulación y se resuelve el problema lineal nuevamente, de lo contrario se continua con el

proceso de subdivisión del problema. El objetivo es reducir significativamente el número de nodos del árbol mejorando la formulación de los subproblemas.

5.5.1.4. Relajación Lagrangiana. Los problemas de programación entera están vinculados a una serie de restricciones que determinan su complejidad. A través de este método se busca transformar el problema original en uno más fácil de resolver el cual se denomina problema relajado; en éste, las restricciones son incorporadas a la función objetivo afectadas por un determinado peso denominado multiplicador de Lagrange, el cual penaliza el incumplimiento de las restricciones. El objetivo consiste en hallar un límite inferior o superior, según la función de optimización, lo más cercano al valor óptimo. Para alcanzar este objetivo se procede a resolver iterativamente, una secuencia de sub problemas modificados (Fisher, 1985).

5.5.2. Métodos heurísticos. Éstos surgen ante la existencia de una gran cantidad y variedad de problemas tanto teóricos como prácticos que son difíciles de resolver y que necesitan ser resueltos de manera eficiente, por ello se recurre a procedimientos que entreguen buenas soluciones sin ser necesariamente las óptimas.

Entre las razones por las cuales se utilizan estos métodos están (Martí, 2003):

- Dada la naturaleza del problema no se conoce ningún método exacto para su solución.
- Aunque exista un método exacto, su uso computacional es muy elevado.
- El método heurístico es más flexible que el exacto permitiendo incorporar otras condiciones de difícil modelado.
- El método heurístico se utiliza como una solución de partida o paso intermedio dentro de un procedimiento global que garantiza el óptimo de un problema.

- Se necesita una representación adecuada del espacio de soluciones, una solución o conjunto de soluciones iniciales y un mecanismo de exploración.
- Si en dos iteraciones determinadas, la solución es la misma, la nueva solución de la siguiente iteración no tiene por qué ser necesariamente la misma por que se tratan de dos instantes diferentes en el proceso de búsqueda.

5.5.2.1. Algoritmos constructivos. Procedimientos iterativos en los que la solución se construye paso a paso, es decir, una solución vacía o parcial se va construyendo por la adición de componentes en cada iteración hasta que se completa. Éste es uno de los métodos heurísticos más rápidos, aunque puede arrojar soluciones de menor calidad en comparación a las obtenidas por búsqueda local (Oviedo & Valdivieso, 2016).

5.5.2.2. Algoritmos de búsqueda local. Se basan en explorar el entorno o vecindario de una solución a través de una operación básica llamada movimiento. El procedimiento parte de una solución inicial $X_o$que pertenece a un entorno N (X_o), luego se le aplica un movimiento M_1 y se escoge una nueva solución bajo cierto criterio; este proceso se aplica reiteradamente describiendo una trayectoria en el espacio de soluciones.

La nueva solución en el entorno se puede elegir bajo diferentes criterios, uno de los más simples y comunes es tomar la solución con mejor evaluación en la función objetivo siempre y cuando la nueva solución sea mejor que la actual. El algoritmo se detiene cuando ya no puede encontrar una mejor solución, entonces la solución actual se denomina *óptimo local* en el entorno definido; éste hecho es el punto de referencia para algunos procedimientos metaheurísticos en los que se evita quedar atrapados en un óptimo local lejano del global (Martí, 2003).

5.5.2.3. Algoritmos de descomposición. Son aquellos en los que el problema original se descompone en problemas más sencillos de resolver, teniendo en cuenta que estos, de manera general pertenecen al mismo problema.

5.5.2.4. Métodos de reducción. Con estos se busca una característica en el problema con la que se pueda simplificar su resolución; para ello se buscan propiedades comunes a las buenas soluciones que luego se introducen como restricciones en el problema para así restringir el espacio de soluciones.

5.5.2.5. Reglas de asignación de trabajos. Estas proveen una guía para la secuenciación de los trabajos que deben realizarse en sistemas productivos. Estas son de gran importancia dado que permiten generar una secuencia de los trabajos según un criterio basado en algún dato de entrada de los trabajos y buscan mejorar el desempeño de la programación de un indicador en particular, por ejemplo: minimizar la cantidad de trabajos tardíos, minimizar el tiempo de flujo medio, minimizar el atraso máximo entre otros. Dentro de las reglas de asignación más conocidas se encuentran:

- **SPT** (*Shortest Processing Time*). Los trabajos con menor tiempo de procesamiento son programados primero en la secuencia de producción. Para la aplicación de esta regla se debe conocer el número de trabajos, los nombres y los tiempos de procesamiento de cada uno de ellos.

En general esta regla es la mejor para minimizar el flujo de trabajo y el flujo promedio de trabajo en el sistema, pero los trabajos con tiempos de procesamiento más largo podrían retrasarse de manera continua por darle prioridad a los más cortos, por lo tanto, requiere de un ajuste periódico para la realización de dichos trabajos.

- **FIFO** (*First In First Out*). Los trabajos son ejecutados en el mismo orden en el que llegan al centro de procesamiento. Esta regla tiene la ventaja de ser considerada como justa por los clientes lo cual es importante en sistemas de servicios.

- **LIFO** (*Last In First Out*). Los trabajos son ejecutados de manera inversa al orden de llegada al centro de procesamiento, es decir, aquellos trabajos que llegan de últimos serán programados primero en la secuencia de programación. Como consecuencia, esta regla suele elevar el número de trabajos tardíos por lo cual requiere de un ajuste periódico para programar aquellos trabajos que llegaron al inicio.

- **LPT** (*Large Processing Time*). Los trabajos con el mayor tiempo de procesamiento son programados primero en la secuencia de producción, al igual que en el SPT, se debe conocer el número de trabajos, el nombre y tiempo de procesamiento de los mismos.

- **EDD** (*Earliest Due Date*). Los trabajos son programados según la fecha de entrega programada, por lo tanto, aquellos que están próximos a entregarse son programados primero que aquellos que tienen mayor tiempo de entrega.

- **CR** (*Critical Ratio*). Los trabajos son programados de acuerdo con este índice, el cual es calculado como la diferencia entre la fecha de vencimiento y la fecha actual, dividida entre el número de días hábiles que quedan y posteriormente se ejecutan los trabajos con menor CR.

5.5.3. Métodos metaheurísticos. A diferencia de los heurísticos, los métodos metaheurísticos buscan huir de óptimos locales reorientando la búsqueda de acuerdo a la evolución del proceso. El punto de partida es una solución o conjunto de soluciones a partir de las cuales se obtienen otras parecidas, entre estas se elige una o varias soluciones que satisfacen un criterio; el proceso se repite hasta que se cumple alguna condición previamente definida.

"*Los procedimientos metaheurísticos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. Los metaheurísticos proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos*." (Osman & Kelly, 1996).

Algunas características comunes a las metaheurísticas son (Sadiq & Habib, 2000) :

- No saben si llegan a la solución óptima (ciegas). Por lo tanto, se debe establecer un criterio de parada.
- Son algoritmos aproximados que no garantizan la obtención de la solución óptima.
- Aceptan ocasionalmente malos movimientos (En los procesos de búsqueda la nueva solución no es necesariamente mejor en términos de la función objetivo que la inmediatamente anterior).
- Algunas veces aceptan, incluso, soluciones no factibles como paso intermedio para acceder a nuevas regiones no exploradas.

En este tipo de técnicas es importante un balance entre diversificación e intensificación; entendiendo diversificación como la evaluación de soluciones en distintas regiones del espacio de búsqueda e intensificación como la evaluación de soluciones en pequeñas regiones o vecindarios dentro del espacio de soluciones; lo anterior se puede expresar en otros términos como el equilibrio entre exploración y explotación.

5.6. Metaheurísticas

5.6.1. Búsqueda tabú. Esta metaheurística permite resolver problemas de optimización combinatoria y está diseñada para evitar que la búsqueda converja rápidamente en óptimos locales

pues permite moverse a soluciones que no son mejores a la actual. La Búsqueda Tabú se basa en principios de la inteligencia artificial como la "memoria" para guiar la búsqueda, esto es, mantiene información almacenada sobre las últimas soluciones evaluadas para evitar que el algoritmo vuelva a ellas. Para tener una historia o memoria de búsqueda, este procedimiento guarda un número *h* de soluciones anteriores en la llamada "*lista tabú"*, la cual se elimina del entorno usual para crear un *entorno reducido* $N^*(X)$.

El procedimiento de búsqueda es similar a cualquier búsqueda local, pues se parte de una solución x e iterativamente se mueve a otra solución x_i en el entorno de la primera $N(x)$; sin embargo, el movimiento sólo considera las soluciones que pertenecen al entorno reducido de la siguiente manera (Glover & Melián, 2003):

$$N^*(x) = N(x) - T$$

5.6.2. Recocido simulado. Es un método de búsqueda iterativa inspirado en el proceso de solidificación de algunos sistemas (ver tabla 4). En este proceso, a medida que baja la temperatura el conjunto de configuraciones (estados) que puede adoptar el sólido se va restringiendo.

Tabla 4.
Analogía entre los parámetros de optimización y la termodinámica.

TERMODINAMICA	OPTIMIZACIÓN
ESTADOS DEL SISTEMA	Soluciones Factibles
ENERGÍA	Costo
CAMBIO DE ESTADO	Solución en el entorno
TEMPERATURA	Parámetro de parada
ESTADO CONGELADO	Solución Heurística

La estrategia utilizada por este método es comenzar con una temperatura alta que también involucra una alta probabilidad de aceptar un cambio de no mejora y posteriormente reducir la temperatura en cada iteración de tal forma que disminuya tal probabilidad para ir acercandose a la solución óptima (figura 3). El permitir cambios a soluciones de menor calidad evita la converjencia a óptimos locales (Martí, 2003).

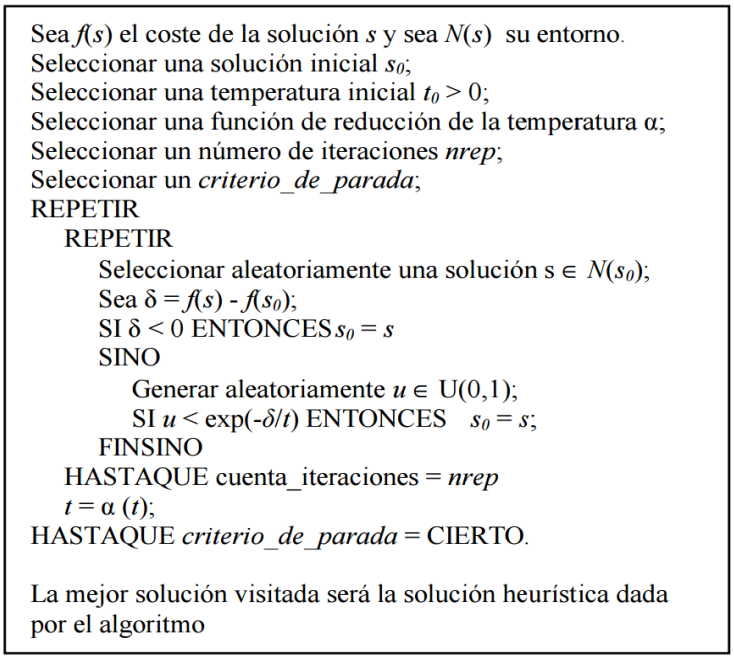


Figura 3. Algoritmo básico de recocido simulado para minimización. *Nota:* Obtenida del estudio realizado por Adenso y Dowsland (2003)

5.6.3. Algoritmo genético. Es una metaheurística basada en la teoría evolucionista de Charles Darwin sobre la supervivencia del más fuerte y la selección natural. En esta aproximación se considera un grupo de soluciones o individuos, la cual se denomina población; cada individuo está conformado por caracteres en serie en analogía a como los cromosomas están compuestos por genes. El algoritmo comienza con la generación aleatoria de una población inicial en la que se

evalúa la fortaleza o aptitud de cada individuo expresada en una mayor probabilidad de ser seleccionados.

La población inicial evoluciona un número de generaciones (iteraciones) usando dos tipos de operadores genéticos: (1) unitarios: Como la mutación e inversión donde se modifica un sólo cromosoma, (2) Cruce de un par de cromosomas elegidos aleatoriamente entre los que se combina material genético y se crean nuevas soluciones (descendencia). La nueva población se crea con los mejores individuos entre la población inicial y los modificados y el proceso se repite hasta cumplir con el número de iteraciones que se establecieron como criterio de parada (Chaovalitwongse, Werner, Reodecha, & Jungwattanakit, 2009).

5.6.4. Búsqueda en vecindario variable (VNS). Se basa en cambiar sistemáticamente la estructura de vecindarios por la que se realiza la búsqueda, además de una búsqueda local. Su campo de aplicación son los problemas de optimización combinatoria y optimización global.

Una estructura de entornos en el espacio de soluciones X es una aplicación N: X→ 2^x que asocia a cada solución $x \in X$ un entorno de soluciones $N(x) \subseteq X$ consideradas vecinas de *x.* las metaheurísticas de búsqueda local aplican una transformación o movimiento a la solución de búsqueda y por tanto utilizan una estructura de entornos. El entorno de una solución $x \in$ X estaría compuesto por todas las soluciones que se pueden obtener desde *x* mediante uno de los movimientos contemplados (Glover & Kochenberger, 2003).

Una solución factible $x^* \in X$ es un mínimo global si no existe una solución $x \in$ X tal que $f(x) < f(x^*)$. Decimos que la solución $x^* \in X$ es un mínimo local con respecto a N si no existe una solución $x \in N(x^*) \subseteq X$ tal que $f(x) \leq f(x^*)$. El VNS se basa en tres principios:

i. Un mínimo local en una estructura de vecindario no lo es necesariamente en otra.

ii. Un mínimo global es mínimo local en todas las posibles estructuras de vecindario.

iii. En muchos problemas, los mínimos locales son relativamente cercanos.

El pseudocódigo para esta metaheurística es el siguiente (figura 4):

***Inicio**:* seleccionar un conjunto de estructuras de vecindario *N(k)* que será usado en la búsqueda; encontrar una solución inicial *x; escoger un criterio de parada.*

Repetir la siguiente secuencia hasta alcanzar el criterio de parada:

(1) Hacer *k←1;*
(2) Repetir los siguientes pasos hasta que $k=k_{max}$*:*
   a) Generar un punto x' de manera aleatoria para el k-ésimo vecindario de *x, (x'∈N(x)).*
   b) Aplicar un método de búsqueda local con *x'* como solución inicial, denotando con x'' el óptimo local obtenido.
   c) Si el óptimo local es mejor que el actual, moverse a él (*x←x'')* y continuar la búsqueda con N_1(*k←1);* de lo contrario *k←k+1.*

Figura 4. Pseudocódigo para Búsqueda en Vecindario Variable (VNS). Nota: Obtenida del estudio realizado por Glover y Kochenberger (2003).

5.6.5. Búsqueda Descendente en Vecindario variable (VND). Consiste en determinar iterativamente una mejor solución a partir de la solución actual mediante algún movimiento. En la clásica búsqueda descendente se trata de reemplazar la solución actual por la mejor de todas las soluciones que se pueda encontrar mediante los movimientos contemplados; mientras que con la estrategia *ansiosa* se aplica un movimiento de mejora desde que se detecte alguna solución mejor. La elección de los movimientos a considerar puede ser determinante en el éxito de la búsqueda local, pero es difícil determinar *a priori* cuál de las posibilidades va a ser la más efectiva.

Si en una búsqueda local descendente se realiza un cambio de estructura de vecindario cada vez que se llega a un mínimo local se obtiene la Búsqueda Descendente en Vecindario Variable (VND). Denotando $N_k, \mathrm{k} = 1, \dots, k_{max}$, a un conjunto de estructuras de vecindario en el espacio X. Los entornos N_k pueden ser generados estableciendo ciertas en condiciones en el espacio de soluciones.

La solución final proporcionada por el algoritmo es un mínimo local con respecto a cada una de las *k* estructuras de vecindario, por lo cual la probabilidad de alcanzar un mínimo global es mayor que el usar una sola estructura (Glover & Kochenberger, 2003).

El pseudocódigo para el VND es el siguiente (figura 5):

***Inicio**:* seleccionar un conjunto de estructuras de vecindario *N(k)* y una solución inicial *x;*

Repetir hasta que no se obtenga mejora:

(1) Hacer *k←1;*
(2) Repetir los siguientes pasos hasta que $k=k_{max}$*:*
   a) Exploración del entorno: Obtener la mejor solución de *x*' del k-ésimo entorno de *x*.
   b) Moverse o no: Si la solución obtenida *x*' es mejor que x, hacer *x←x'* y *k←1;* de lo contrario, hacer *k←k+1*

Figura 5. Pseudocódigo para Búsqueda Descendente en Vecindario Variable (VND). Obtenida del estudio realizado por Glover y Kochenberger (2003).

5.7. Análisis de Varianza ANOVA

Es una técnica estadística que permite evaluar el efecto individual o la interacción de dos o más factores sobre una variable dependiente cuantitativa. En un ANOVA factorial existe una hipótesis nula por cada factor que expresa que las medias poblacionales definidas por los niveles del factor son iguales, además de una hipótesis nula por cada interacción de factores que expresa que su efecto es nulo; estas hipótesis se rechazan con base en un nivel crítico que puede ser el estadístico F o su valor-P asociado (Diaz, 2009).

La validez de este análisis supone que los errores están normalmente distribuidos con media 0 (normalidad), la varianza de la variable respuesta es la misma en todas las poblaciones (homocedasticidad) y las observaciones son independientes entre sí (independencia).

6. Descripción del DAPFSP-SDST con Fábricas Heterogéneas

El DAPFSP-SDST con fábricas heterogéneas es un sistema productivo que comprende dos etapas (figura 6):

- ***Etapa de procesamiento***: Está conformada por ***f*** fábricas, cada una de las cuales es un sistema *flowshop* con un mismo número de máquinas ***k*** dispuestas en serie. En ésta se procesan los trabajos o componentes que se envían a la etapa de ensamble.

Se considera que las fábricas son heterogéneas puesto que, aunque puedan procesar los mismos trabajos y cuenten con el mismo número de máquinas, los tiempos de procesamiento y de alistamiento de las máquinas varían entre fábricas. Además, cada fábrica es un *flowshop* permutado por lo cual la secuencia de procesamiento de los trabajos en la primera máquina se mantiene para las siguientes.

- ***Etapa de ensamble***: Está compuesta por una sola máquina y en ella se ensambla cada producto una vez que todos los trabajos o componentes que lo conforman hayan terminado su procesamiento en la etapa anterior.

En relación con los tiempos de alistamiento de las máquinas, se contempla que éstos son dependientes de la secuencia tanto en la etapa de procesamiento como en la de ensamble, a su vez que se permite el alistamiento temprano.

En el presente proyecto, la solución al problema involucra asignar los trabajos a las diferentes fábricas, generar la secuencia de procesamiento de los trabajos para cada fábrica y la de ensamble de los productos. El objetivo es minimizar el tiempo máximo de completamiento *makespan*.

Cuando N>F y el criterio de minimización es el *makespan*, a cada fábrica se le asigna al menos un trabajo; por lo tanto, el número total de soluciones para el DAPFSP está dado por la expresión (Ruiz & Naderi, 2010):

$$\text{TOTAL SOLUCIONES} = \binom{N-1}{F-1} * N! * P!$$

Donde $\binom{N-1}{F-1}$ expresa el número de particiones posibles de los trabajos en las fábricas, $N!$ el número de secuencias posibles de trabajos para una partición y $P!$ el número de secuencias de productos.

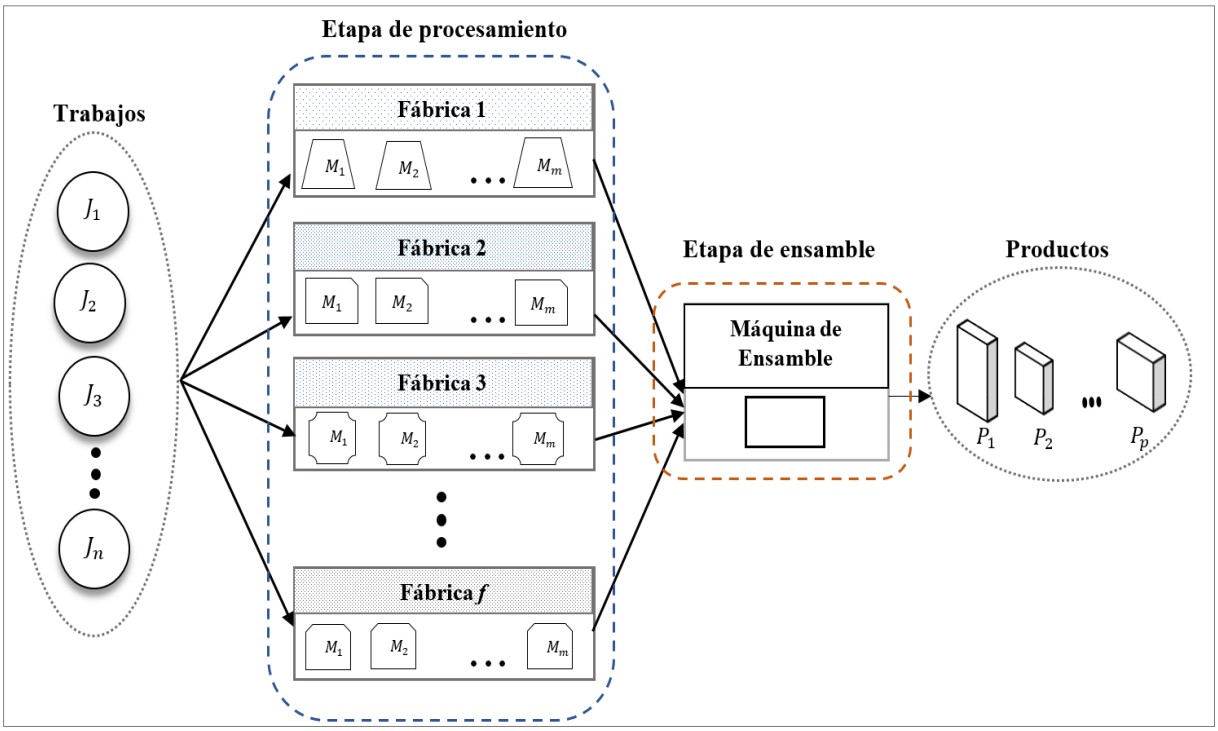


Figura 6. Esquema del DAPFSP-SDST con fábricas heterogéneas.

En la definición del problema se tienen en cuenta las siguientes consideraciones:

- Un trabajo puede procesarse en máximo una fábrica.
- Todas las fábricas pueden procesar cualquier trabajo.
- Todas las fábricas tienen el mismo número de máquinas, y éste es al menos dos.
- Todos los trabajos se procesan en las *m* máquinas en el mismo orden *1,2, 3,…,m*.
- La secuencia en la que se procesan los trabajos es la misma para todas las máquinas en una misma fábrica.
- Un trabajo puede ser procesado por máximo una máquina al tiempo y una máquina puede procesar sólo un trabajo al tiempo.
- Un trabajo sólo puede avanzar a la siguiente máquina si su procesamiento en la máquina actual ha terminado.
- El tiempo de alistamiento de una máquina depende del trabajo o producto que se acaba de procesar y del que sigue.
- Se permite alistamiento temprano de las máquinas en la etapa de procesamiento y de ensamble.
- Los tiempos de procesamiento y de alistamiento varían entre fábricas y máquinas, su valor se conoce y es determinístico.
- Se consideran un trabajo y un producto ficticios *cero*, que representan el estado inicial de una máquina.
- En la etapa de ensamble, la máquina puede procesar sólo un producto a la vez.
- Un producto comienza a ensamblarse si y sólo si todos los trabajos que lo componen han terminado de procesarse.

7. Modelo MILP para el DAPFSP-SDST

Para resolver el problema bajo estudio se desarrolla el siguiente modelo de programación lineal entera mixta. Su formulación está inspirada en el modelo desarrollado por Ruiz et al. (2013) para el problema DAPFSP y en el primero de los seis modelos desarrollados por Naderi y Ruiz (2010) para el problema de DPFSP; los cuales, al utilizar variables binarias para modelar la secuencia de los trabajos y productos, facilitan la creación de las restricciones relacionadas con los tiempos dependientes de la secuencia.

Conjuntos

- ***k, j*** Trabajos k= {0,1,2,…,n} , j= {1,2,…,n}
- ***f*** Fábricas f= {1,2,…,F}
- ***i*** Máquinas i= {1,2,…,m}
- ***s, p*** Productos s= {0,1,2,…,P} , p= {1,2,…,P}

Parámetros

- $\boldsymbol{PT_{j,i,f}}$**:** Tiempo de procesamiento del trabajo j en la máquina i de la fábrica f.
- $\boldsymbol{ST_{k,j,i,f}}$**:** Tiempo de alistamiento de la máquina i de la fábrica f para procesar el trabajo j cuando lo precede inmediatamente el trabajo k.
- $\boldsymbol{AT_{p}}$**:** Tiempo de ensamble del producto p.
- $\boldsymbol{AST_{s,p}}$**:** Tiempo de alistamiento de la máquina de ensamble para el producto p cuando lo precede inmediatamente el producto s.

- $\boldsymbol{ASIG_{j,p}}$**:** Parámetro binario con valor 1 si el trabajo j pertenece al producto p, y valor cero de lo contrario.

Variables

- $\boldsymbol{Y_{j,f}}$**:** Variable binaria con valor 1 si el trabajo j se asigna a la fábrica f, y cero de lo contrario.
- $\boldsymbol{X_{k,j,f}}$**:** Variable binaria con valor 1 si el trabajo k precede inmediatamente a j en la fábrica f, y cero de lo contrario.
- $\boldsymbol{Z_{s,p}}$**:** Variable binaria con valor 1 si el producto s precede inmediatamente a p en la fábrica f, y cero de lo contrario.
- $\boldsymbol{C_{j,i}}$**:** Tiempo de completamiento del trabajo j en la máquina i.
- $\boldsymbol{AC_p}$**:** Tiempo de completamiento del producto p en la etapa de ensamble.
- $\boldsymbol{Cmax}$**:** Makespan

Función objetivo:

Min $Cmax$

Restricciones

(1) $\sum_{f=1}^{F} Y_{j,f} = 1 \qquad \forall \boldsymbol{j}$

(2) $\sum_{j=1}^{n} X_{0,j,f} = 1 \qquad \forall \boldsymbol{f}$

(3) $\sum_{f=1}^{F} X_{j,j,f} = 0 \qquad \forall \boldsymbol{j}$

(4) $\sum_{f=1}^{F} \sum_{k=0,k\neq j}^{n} X_{k,j,f} = 1 \quad \forall \boldsymbol{j}$

(5) $\sum_{f=1}^{F} \sum_{j=1,j\neq k}^{n} X_{k,j,f} \leq 1 \quad \forall \boldsymbol{k > 0}$

(6) $\sum_{k=0,k\neq j}^{n} (X_{k,j,f} + X_{j,k,f}) \leq 2 * Y_{j,f} \qquad \forall \boldsymbol{j, f}$

(7) $\sum_{f=1}^{F}\left(X_{k,j,f} + X_{j,k,f}\right) \leq 1 \qquad \forall \boldsymbol{k} = \{\boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{n-1}\}, \forall \boldsymbol{j} > \boldsymbol{k}$

(8) $C_{j,i} \geq C_{j,i-1} + \sum_{k=0}^{n}\sum_{f=1}^{F}(X_{k,j,f} * PT_{j,i,f}) \qquad \forall \boldsymbol{j}, \boldsymbol{i}$

(9) $C_{j,i} \geq C_{k,i} + \sum_{f=1}^{F}[X_{k,j,f} * (PT_{j,i,f} + ST_{k,j,i,f})] + [(\sum_{f=1}^{F} X_{k,j,f}) - 1]M \qquad \forall \boldsymbol{k}, \boldsymbol{j}, \boldsymbol{i}$

(10) $\sum_{s=0,s\neq p}^{P} Z_{s,p} = 1 \qquad \forall \boldsymbol{p}$

(11) $\sum_{p=1}^{P} Z_{s,p} \leq 1 \qquad \forall \boldsymbol{s}$

(12) $Z_{s,p} + Z_{p,s} \leq 1 \qquad \forall \boldsymbol{s}, \boldsymbol{p}, \ \boldsymbol{s} > \boldsymbol{p}$

(13) $AC_p \geq (C_{j,m} * ASIG_{j,p}) + \sum_{s=0,s\neq p}^{P} Z_{s,p} * AT_p \qquad \forall \boldsymbol{j}, \boldsymbol{p}$

(14) $AC_p \geq AC_s + \left[Z_{s,p} * \left(AT_p + AST_{s,p}\right)\right] + \left(Z_{s,p} - 1\right)M \qquad \forall \boldsymbol{s}, \boldsymbol{p}$

(15) $Cmax \geq AC_p \qquad \forall \boldsymbol{p}$

(16) $C_{j,i} \geq 0 \qquad \forall \boldsymbol{j}, \boldsymbol{i}$

(17) $AC_p \geq 0 \qquad \forall \boldsymbol{p}$

(18) $Cmax \geq 0$

(19) $Y, X, Z \ \in \{0,1\}$

La función objetivo minimiza el *makespan*; la restricción (1) asegura que cada trabajo se asigne a sólo una fábrica y la (2) que cada fábrica tenga un trabajo *dummy* "cero". Las restricciones (3) a (7) marcan las pautas de orden para la generación de la secuencia de los trabajos: en (3) cada trabajo no puede ser sucesor o predecesor de sí mismo; en (4) y (5) cada trabajo es sucesor inmediato sólo una vez y puede preceder inmediatamente a otro máximo una vez (el último trabajo de la secuencia no precede a ningún otro); en (6) se asegura que cada trabajo sólo tenga su predecesor y sucesor inmediato en la misma fábrica a la que fue asignado y en (7) que cada trabajo no pueda ser sucesor y predecesor inmediato de otro al mismo tiempo. La restricción (8) se relaciona con la secuencia de los trabajos en las máquinas al establecer que un trabajo no puede

iniciar su procesamiento en una máquina *i* sin haber terminado su procesamiento en la máquina anterior *i-1*. La restricción (9) asegura que en las máquinas no haya interferencia de trabajos; es decir, que una máquina i no pueda procesar un trabajo *j* sin haber terminado de procesar el trabajo predecesor *k*; esta restricción se habilita cuando *k* precede inmediatamente a *j* en la fábrica correspondiente.

Las restricciones (10) y (11) aseguran que cada producto tenga sólo un predecesor y máximo un sucesor inmediato. La restricción (13) vincula la etapa de procesamiento con la de ensamble pues asegura que un producto no puede empezar su ensamble sin que todos los trabajos asociados a él estén terminados. La restricción (14) es análoga con la restricción (9) al establecer que un producto *p* no puede empezar a ensamblarse sin que su predecesor inmediato *s* haya terminado. Por último, en la restricción (15) se define el *makespan* como el mayor tiempo de completamiento de todos los productos.

Observaciones. En la restricción (7) se limita que los valores de *j* sean mayores que *k* para evitar generar dos veces el mismo grupo de restricciones y se limitan los valores de *k* hasta *n-1* porque no existe un *j*>*n.* De manera similar en la restricción (12) se limita que *s* sea mayor que *p* para evitar generar un mismo conjunto de restricciones dos veces.

7.1. Implementación en Gams

El modelo es ejecutado en un computador con procesador Intel Core i5-4570 de 3,2 GHz, memoria RAM de 8 GB y un sistema operativo Windows 7 Home premium de 64 bits. El solver elegido dentro de Gams es **CPLEX 12**, el cual permite resolver problemas grandes y difíciles de tipo MIP a través del algoritmo Branch and Cut.

En la evaluación de las instancias para fábricas homogéneas y heterogéneas se establece un tiempo máximo de ejecución de *900* segundos en referencia al estudio de Ruiz et al. (2013), quienes para evaluar su modelo MILP para el DAPFSP utilizan dos límites de tiempo: 900s y 3600s, de los cuales se toma el límite más corto dada la cantidad de instancias a evaluar y el tiempo disponible dentro del presente proyecto para su ejecución.

Para la etapa de validación, se corren 900 instancias pequeñas correspondientes al problema DAPFSP-SDST con fábricas homogéneas (Ruiz et al., 2014), las cuales resultan de 5 instancias por cada combinación de los factores presentados en la tabla 5.

Por otra parte, para el problema DAPFSP-SDST con fábricas heterogéneas se evalúan 720 instancias pequeñas y 270 instancias grandes que corresponden a 5 instancias por cada combinación de los factores presentados en las tablas 6 y 7:

Tabla 5.
Factores de Instancias pequeñas para el DAPFSP-SDST con fábricas homogéneas.

Factor	Número de niveles	Valores
Trabajos **N**	5	8, 12, 16, 20, 24
Máquinas **M**	4	2, 3, 4, 5
Fábricas **F**	3	2, 3, 4
Productos **P**	3	2, 3, 4
$5*4*3^2 =$**180** combinaciones 180*5=**900** instancias		

Nota. Adaptado del estudio de Ruiz et al. (2014)

Tabla 6.
Factores de Instancias pequeñas para el DAPFSP-SDST con fábricas heterogéneas.

Factor	Número de niveles	Valores
Trabajos **N**	4	8, 12,16,24
Máquinas **M**	4	2, 3, 4, 5
Fábricas **F**	3	2, 3, 4
Productos **P**	3	2, 3, 4
4^2*3^3=**144** combinaciones 144*5=**720** instancias		

Nota. Adaptado del estudio de Ruiz et al. (2014)

Tabla 7.
Factores de Instancias grandes para el DAPFSP-SDST con fábricas heterogéneas.

Factor	Número de niveles	Valores
Trabajos **N**	2	100,200
Máquinas **M**	3	5,10,20
Fábricas **F**	3	4,6,8
Productos **P**	3	30,40,50
$2*3^3*$=**54** combinaciones 54*5=**270** instancias		

Nota. Adaptado del estudio de Ruiz et al. (2015)

Con base al estudio de Ruiz et al. (2014) los tiempos de procesamiento y alistamiento en todas las instancias se han generado con las siguientes distribuciones de probabilidad:

Tabla 8.
Distribución de probabilidad de los tiempos de procesamiento, ensamble y alistamiento.

Parámetro	Distribución (s)
PT	*U* [1,99]
AT	*U* [1*Np, 99*Np]
ST **AST**	*U* [1,50]

Nota. Np representa la cantidad de trabajos asociados a cada producto p.
Adaptado del estudio de Ruiz et al. (2014)

Para crear las instancias del problema con fábricas heterogéneas, los tiempos de alistamiento y procesamiento de las fábricas homogéneas utilizados por Ruiz et al. (2014,2015) son asignados a la primera fábrica heterogénea, mientras que en el resto de fábricas los tiempos se generan de acuerdo con la tabla 8. Las instancias generadas se encuentran disponibles en Mendeley con el DOI:10.17632/6cpkw9v9gy.2

Para facilitar la ejecución de las instancias se crea una rutina en Matlab asociada a otra en Gams (figura 7).

La rutina en Matlab realiza las siguientes acciones:

i. Carga las tablas de la instancia correspondiente junto con los parámetros N, P, M y F.

ii. Crea el archivo gdx con los sets.

iii. Crea un archivo txt que contiene la ruta de la hoja de cálculo donde están los parámetros (PT, ST, AT, AST y ASIG).

iv. Inicializa y ejecuta la rutina en Gams.

v. Lee el archivo gdx con los resultados de la ejecución y con ellos crea el vector solución.

vi. Guarda el *makespan*, vector solución, elapsed time y el gap relativo en una hoja de cálculo de Excel.

Mientras que la rutina en Gams:

i. Lee el archivo gdx en el que se encuentran los sets.

ii. Llama la hoja de cálculo donde están los parámetros (PT, ST, AT, AST y ASIG) siguiendo la ruta especificada en el archivo txt.

iii. Ejecuta el modelo con el solver CPLEX12.

iv. Guarda en un archivo gdx los resultados para X, Z, elapsed time y rgap.

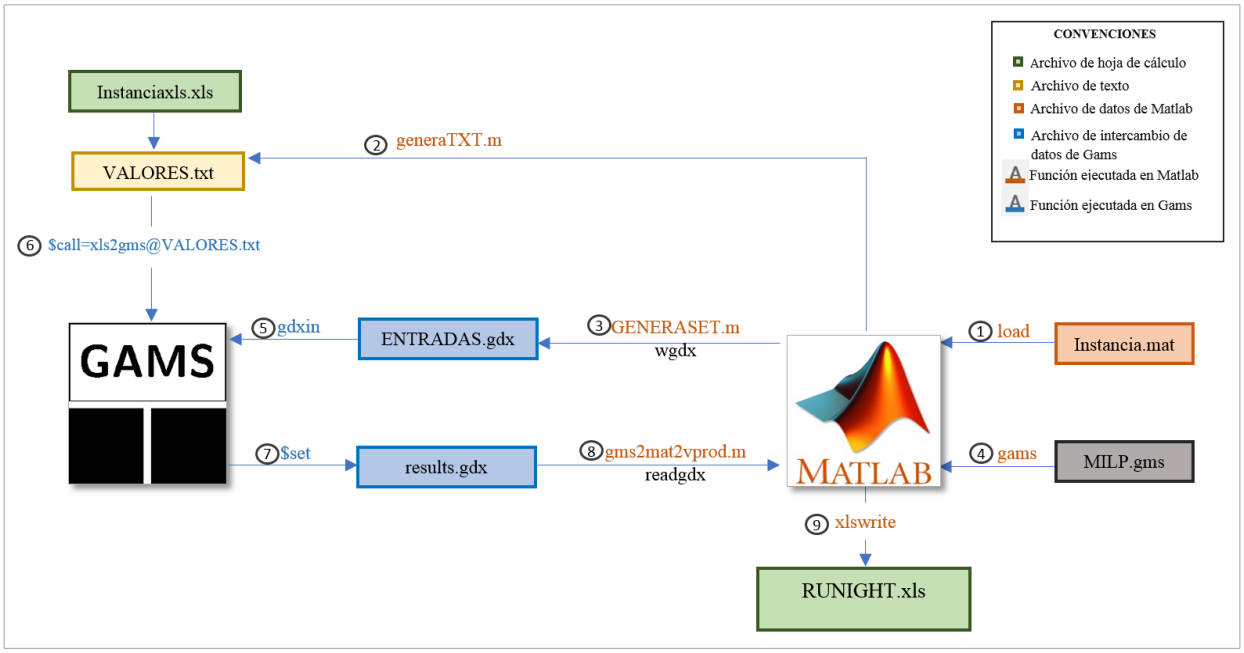


Figura 7. Flujo de información en la ejecución del modelo MILP.

El gap relativo (rgap) hace referencia a la diferencia porcentual de *makespan* entre la mejor solución estimada por gams y la solución encontrada, mientras que el elapsed time corresponde al tiempo de ejecución del solver.

$$rgap = \frac{encontrada - mejor\ estimada}{mejor\ estimada}$$

El script del modelo MILP implementado en Gams se puede encontrar en el apéndice B. Así mismo, las funciones y Scripts de Matlab que se usaron se detallan en los apéndices C y D.

7.2. Validación del Modelo MILP

Dado que en la literatura no se encuentran estudios para el DAPFSP-SDST con fábricas heterogéneas, para validar el modelo MILP desarrollado se hace uso de los resultados obtenidos en el estudio de Ruiz et al. (2014) sobre el DAPFSP_SDST con fábricas homogéneas, en el que se

desarrollan cuatro heurísticas CH11, CH12, CH21 y CH22 que se derivan de la combinación de dos heurísticas constructivas con dos reglas de asignación de trabajos (ver tabla 9).

Tabla 9.
Heurísticas constructivas y reglas de asignación utilizadas en el estudio de Ruiz et al. (2014)

Heurísticas Constructivas	Reglas de asignación
1) Basada en la generación de una secuencia de productos con el menor tiempo de completamiento de ensamble.	1) Asignar el trabajo j a la fábrica con el menor Cmax actual, es decir sin incluir a j.
2) Da prioridad a los productos conformados por los trabajos que se produzcan más rápido en la etapa de producción.	2) Asignar el trabajo j a la fábrica con el menor Cmax después de programar a j.

Para que el modelo MILP desarrollado pueda resolver las instancias del problema con fábricas homogéneas, se asume que todas las fábricas son como la primera, lo que se logra modificando las restricciones (8) y (9) de la siguiente manera:

(8*) $C_{j,i} \geq C_{j,i-1} + \sum_{k=0}^{n} \sum_{f=1}^{F} (X_{k,j,f} * PT_{j,i,1})$

(9*) $C_{j,i} \geq C_{k,i} + \sum_{f=1}^{F} [X_{k,j,f} * (PT_{j,i,1} + ST_{k,j,i,1})] + [(\sum_{f=1}^{F} X_{k,j,f}) - 1]M$

Como primera medida, se hicieron pruebas con algunas instancias aleatorias para fábricas homogéneas en las que se compararon los resultados con los obtenidos en la literatura. A través de estas comparaciones se encontró que existían consideraciones que no se tuvieron en cuenta en la programación inicial, como es el caso del alistamiento temprano en las etapas de procesamiento y ensamble. También se encontró que en la determinación del tiempo de completamiento de ensamble de los productos no se estaba teniendo en cuenta el tiempo de finalización de los trabajos asociados a ellos en la etapa anterior, afectando el cálculo de *makespan.*

Una vez corregidos dichos problemas se sometió el modelo a un análisis más profundo, en el cual se ejecutaron las 900 instancias halladas en la literatura para fabricas homogéneas dentro de un límite de tiempo de 900s. En la ejecución de dichas instancias se logra obtener la solución óptima (rgap ≤ 0,000001) en 315 de ellas; es decir, en un 35%.

Por otra parte, en 769 (85,44%) instancias el valor de *makespan* obtenido con el modelo MILP es mejor que el encontrado por las heurísticas de la literatura. Los resultados obtenidos con el modelo MILP y los datos relacionados con la validación se encuentran en el apéndice E.

El valor de *makespan* obtenido con el modelo MILP (S_{MILP}) se compara con el de la mejor solución obtenida en las cuatro heurísticas ($Best_{CH}$), obteniéndose los siguientes resultados:

Tabla 10.
Resultados obtenidos de la comparación del modelo MILP con los cuatro algoritmos.

	MILP es la óptima	**N° de instancias**	**Porcentaje de instancias**
$S_{MILP} \leq Best_{CH}$	NO	530	58,9%
	SI	311	34,6%
$S_{MILP} > Best_{CH}$	NO	55	6,1%
	SI	4	0,4%
Total		**900**	**100%**

La tabla 10 muestra cuatro casos que resultan en la comparación:

- ***Caso 1:*** La solución obtenida con el modelo MILP es mejor que la de las heurísticas, aún sin ser la óptima.
- ***Caso 2:*** La solución obtenida con el modelo MILP es mejor que la de las heurísticas y corresponde a la óptima.

- ***Caso 3:*** La solución obtenida con el modelo MILP no es mejor que la de las heurísticas ni corresponde a la óptima.
- ***Caso 4:*** La solución obtenida con el modelo MILP no es mejor que la de las heurísticas y corresponde a la óptima.

La validación del modelo MILP implica que la solución óptima que éste genere sea mejor o igual a las obtenidas con las heurísticas en todas las instancias; sin embargo, como el tiempo de ejecución del solver está limitado, no se lograron obtener todas las soluciones óptimas, de allí que sea necesario analizar qué casos apoyan o no la validación:

Por su parte los casos 1 y 2, que conforman el 93,5% de las instancias, apoyan la validación porque sin importar si la solución obtenida con el modelo MILP es la óptima, ésta es mejor o igual que las obtenidas con las heurísticas.

Ahora bien, en las 55 instancias que pertenecen al caso 3, el rgap siempre es mayor que la diferencia porcentual entre la solución del MILP y la mejor hallada con las heurísticas, lo que indica que a pesar de que el modelo no encontró una solución mejor en el tiempo límite asignado, se estima según Gams que existe una mejor solución posible que la hallada por las heurísticas, lo cual va de acuerdo con el criterio de validación en el cual la solución óptima del modelo MILP es mejor o igual a las obtenidas con las heurísticas. Esta afirmación se refuerza al realizar la comparación entre el rgap y la diferencia porcentual, en donde se encuentra que en promedio el modelo puede mejorar la solución encontrada S_{MILP} un 11,83% en tanto que las heurísticas sólo la mejoraron un 2,77%.

Rgap promedio: 11,83%

$$\textbf{Diferencia promedio} = \frac{\sum \frac{S_{MILP} - Best_{CH}}{Best_{CH}}}{55} = 2{,}77\%$$

Por último, en el caso 4 se encuentran cuatro instancias en las que la solución óptima del modelo no supera la mejor solución de las heurísticas (tabla 11), contradiciendo el criterio de validación. En estas instancias la mejor solución no siempre fue hallada con la misma heurística ni en condiciones similares respecto a los tamaños de las instancias; por este motivo, el modelo se revisa detalladamente para determinar alguna posible falla y se prueba con instancias para el DAPFSP sin tiempos de alistamiento para las cuales se contaba con las soluciones óptimas, de allí se obtiene que el modelo MILP encuentra la misma solución óptima en todas las instancias, por lo cual no hay forma de comprobar que existe un error en el modelo y se infiere que en estas cuatro instancias, hubo errores en el manejo de los datos por partes de los autores de las heurísticas.

Tabla 11.
Instancias que no cumplen con el criterio de validación del modelo MILP.

N	**P**	**F**	**M**	**Ins**	**CH11**	**CH12**	**CH21**	**CH22**	MILP
8	2	2	5	4	885	885	885	805	807
8	3	2	3	3	699	699	720	720	703
8	4	2	3	3	645	645	551	531	533
12	2	4	3	3	1128	1128	1079	1128	1082

En conclusión, el modelo se puede validar con un apoyo del 99,6% de las instancias evaluadas.

7.3. Análisis de Resultados del Modelo MILP

Con el modelo MILP propuesto se evaluaron 720 instancias pequeñas y 270 instancias grandes para el problema con fábricas heterogéneas en un tiempo límite de 900s. Los resultados obtenidos (*makespan*, gap relativo, tiempo de ejecución y vector solución) se encuentran en el apéndice F.

7.3.1. Instancias grandes. Para el tiempo límite de 900s, el modelo no encontró la solución óptima en ninguna de las instancias y sólo llegó a soluciones factibles en 6, es decir, en un 2,22% de las 270, con un gap relativo en promedio del 90,34%. Ahora bien, con un tiempo límite de 1800s el número de soluciones factibles aumentó a 27 (10%), lo cual es un porcentaje muy bajo respecto al tiempo de ejecución, además de que el gap relativo de esas soluciones es en promedio un 91,15%.

Los anteriores resultados permiten corroborar la pertinencia de desarrollar algoritmos metaheurísticos con los que se logre obtener soluciones de buena calidad en un tiempo computacional razonable para instancias grandes.

7.3.2. Instancias pequeñas. El modelo logró encontrar una solución factible en todas las instancias pequeñas, y la solución óptima en 307 de ellas, el equivalente al 42,64%. El total de corridas tomó un tiempo computacional de 109,15 horas.

Los resultados promedio de gap relativo y tiempo de ejecución para cada combinación se encuentran en las siguientes tablas:

Tabla 12.
Rgap y tiempo de ejecución promedio para las instancias con 8 trabajos.

N=8 Trabajos

Rgap (%)

	F	2				3				4			
	M	2	3	4	5	2	3	4	5	2	3	4	5
P	2	0	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0	0	0	0	0	0

Elapsed time (s)

	F	2				3				4			
	M	2	3	4	5	2	3	4	5	2	3	4	5
P	2	2,2	5,08	3,94	8,22	0,65	1,88	2,28	3,65	0,73	0,69	0,59	0,67
	3	0,71	6,29	3,04	5,11	1,23	1,37	2,47	1,4	0,37	0,93	1,34	1,6
	4	1,59	21,09	5,99	22,19	1,26	1,09	1,43	1,95	0,97	1,38	1,54	1,81

Tabla 13.
Rgap y tiempo de ejecución promedio para las instancias con 12 trabajos.

N=12 trabajos

Rgap (%)

	F	2				3				4			
	M	2	3	4	5	2	3	4	5	2	3	4	5
P	2	10,41	11,9	15,33	15,21	2,46	3,39	4,14	9,62	0,059	0	2,4	0
	3	5,46	9,19	12,23	8,79	0,21	3,5	2,94	2,58	0	0	0	0
	4	5,72	7,95	7,94	14,18	0	0	0	0	0	0	0	0

Elapsed time (s)

	F	2				3				4			
	M	2	3	4	5	2	3	4	5	2	3	4	5
P	2	770,13	860,06	900	900	379,3	550,35	596,71	699,19	265,68	240,29	500,05	182,3
	3	550,67	621,4	900	806,08	185,43	724,9	453,61	364,66	13,21	12,21	36,48	218,64
	4	822,08	900	721,7	900	164,67	24,65	192,17	116,53	15,05	41,52	39,41	64,08

Tabla 14.
Rgap y tiempo de ejecución promedio para las instancias con 16 trabajos.

N=16 trabajos

Rgap (%)

	F	2				3				4			
	M	2	3	4	5	2	3	4	5	2	3	4	5
P	2	17,98	17,09	23,92	21,45	8,14	9,94	8,47	11,78	4,47	3,81	2,49	6,6
	3	17,27	13,28	13,3	15,45	4,83	2,76	9,52	4,1	1,26	4,69	1,53	2,83
	4	8,76	6,57	11,68	11,18	8,91	7,09	2,72	2,02	0,47	2,53	1,62	2,02

Elapsed time (s)

	F	2				3				4			
	M	2	3	4	5	2	3	4	5	2	3	4	5
P	2	900	900	900	900	900	900	900	900	900	900	819,99	743,74
	3	900	900	900	900	900	739,15	900	900	900	716,6	773,01	727,81
	4	900	900	900	900	900	900	732,78	780,31	368,29	644,4	764,7	871,07

Tabla 15.
Rgap y tiempo de ejecución promedio para las instancias con 24 trabajos.

N=24 trabajos

Rgap (%)

	F	2				3				4			
	M	2	3	4	5	2	3	4	5	2	3	4	5
P	2	19,6	25,18	16,09	23,89	13,1	13,93	13,39	15,87	8,87	8,67	10,14	9,79
	3	13,52	15,81	18,45	14,98	9,71	12,52	9,3	14,54	5,31	9,35	8,13	9,95
	4	14,31	19,39	21,42	16,91	5,81	11,85	14,47	20,77	6,36	3,7	18,79	4,99

Elapsed time (s)

	F	2				3				4			
	M	2	3	4	5	2	3	4	5	2	3	4	5
P	2	900	900	900	900	900	900	900	900	900	900	900	900
	3	900	900	900	900	900	900	900	900	900	900	900	900
	4	900	900	900	900	900	900	900	900	900	900	900	900

En las tablas 12 a 15 presentadas anteriormente se observan ciertos comportamientos de aumento en el gap relativo y tiempo de ejecución a medida que cambian los niveles de los factores (tabla 16); por ejemplo, el rgap tiende a crecer a medida que el número de trabajos y máquinas aumenta, mientras que decrece con una cantidad mayor de fábricas y productos. De la misma manera, el elapsed time tiende a crecer con una mayor cantidad de trabajos y a decrecer con mayor número de fábricas.

Tabla 16.
Comportamientos de rgap y elapsed time observados con los promedios.

Factor	Rgap	Elapsed time
↑ N	↑	↑
↑ F	↓	↓
↑ P	↓	—
↑ M	↑	—

Aun cuando se observan estos comportamientos en el rgap a través de las tablas de promedios; para determinar estadísticamente la influencia de los factores N, F, P, y M en la variable rgap se elabora un diseño experimental.

7.3.3. Diseño experimental para la determinación de los factores incidentes en el gap relativo para el modelo MILP. El Diseño experimental seleccionado para analizar el efecto de los factores en el gap relativo es un diseño factorial completo 2^k, en el cual se consideran los factores F, P, M y N.

El diseño experimental sólo contempla los resultados para instancias pequeñas debido a que para la mayoría de instancias grandes el modelo MILP no encontró soluciones factibles dentro de los límites de tiempo establecidos.

Para F, P y M, se asignan como niveles el menor y mayor valor de cada factor; mientras que para N se toma como nivel alto N=24 y como nivel bajo N=16 dado que para N=8 y N=12 el rgap es cero en la mayoría de instancias evaluadas.

Tabla 17.
Factores y niveles utilizados en el diseño factorial 2^4 para analizar los resultados del modelo MILP.

Factor	Nivel bajo	Nivel alto
N	16	24
F	2	4
P	2	4
M	2	5

Tabla 18.
Tratamientos para un diseño factorial completo 2^4.

	A	***B***	***C***	***D***	
N°	N	F	P	M	**Tratamiento**
1	16	2	2	2	(1)
2	24	2	2	2	a
3	16	4	2	2	b
4	24	4	2	2	ab
5	16	2	4	2	c
6	24	2	4	2	ac
7	16	4	4	2	bc
8	24	4	4	2	abc
9	16	2	2	5	d
10	24	2	2	5	ad
11	16	4	2	5	bd
12	24	4	2	5	abd
13	16	2	4	5	cd
14	24	2	4	5	acd
15	16	4	4	5	bcd
16	24	4	4	5	abcd

Nota. Adaptado de Montgomery (2004).

Como en el diseño factorial propuesto los niveles de los factores están definidos y el número de réplicas para todos los tratamientos (tabla 18) es cinco, se emplea un ANOVA de efectos fijos para diseños balanceados. Los datos utilizados para el ANOVA se encuentran en el apéndice F.

Las hipótesis por contrastar para cada factor son en general:

$$H_0: el\ factor\ tiene\ efecto\ nulo$$

$$H_a: el\ factor\ tiene\ efecto$$

Y para cada interacción son:

$$H_0: la\ interacción\ tiene\ efecto\ nulo$$

$$H_a: la\ interacción\ tiene\ efecto$$

Si al realizar el ANOVA el valor-p para una fuente de variación es menor a 0,5%, se rechaza la hipótesis nula y se afirma que el factor o interacción (según sea el caso) tiene un efecto en la variable respuesta.

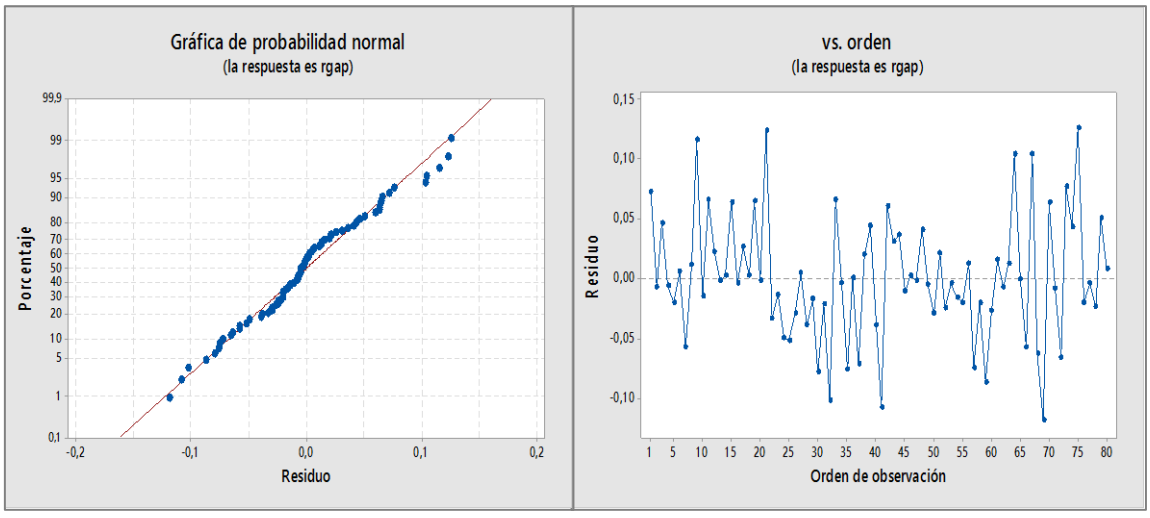


Figura 8. Gráfica de normalidad e independencia de residuos para el rgap.

Como se muestra en la figura 8, los datos siguen una distribución normal y son independientes entre sí ya que no se encuentran tendencias ni patrones; por otro lado, para verificar la homocedasticidad se realiza la prueba de Levene en la que con un valor-p=0,266 y un nivel de significancia del 5%, se encuentra que no hay evidencia significativa para afirmar que las varianzas son diferentes. Por lo tanto, los datos cumplen con los supuestos necesarios para el ANOVA.

Tabla 19.
ANOVA para rgap en el modelo MILP.

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	15	0,315101	0,021007	6,28	0,000
Lineal	4	0,297003	0,074251	22,19	0,000
N	1	0,035947	0,035947	10,74	0,002
P	1	0,043948	0,043948	13,14	0,001
F	1	0,201790	0,201790	60,32	0,000
M	1	0,015318	0,015318	4,58	0,036
Interacciones de 2 términos	6	0,014407	0,002401	0,72	0,637
N*P	1	0,001265	0,001265	0,38	0,541
N*F	1	0,000035	0,000035	0,01	0,919
N*M	1	0,000049	0,000049	0,01	0,904
P*F	1	0,001034	0,001034	0,31	0,580
P*M	1	0,004313	0,004313	1,29	0,260
F*M	1	0,007711	0,007711	2,30	0,134
Interacciones de 3 términos	4	0,003672	0,000918	0,27	0,893
N*P*F	1	0,000456	0,000456	0,14	0,713
N*P*M	1	0,000556	0,000556	0,17	0,685
N*F*M	1	0,001533	0,001533	0,46	0,501
P*F*M	1	0,001127	0,001127	0,34	0,564
Interacciones de 4 términos	1	0,000020	0,000020	0,01	0,939
N*P*F*M	1	0,000020	0,000020	0,01	0,939
Error	64	0,214113	0,003346		
Total	79	0,529215			

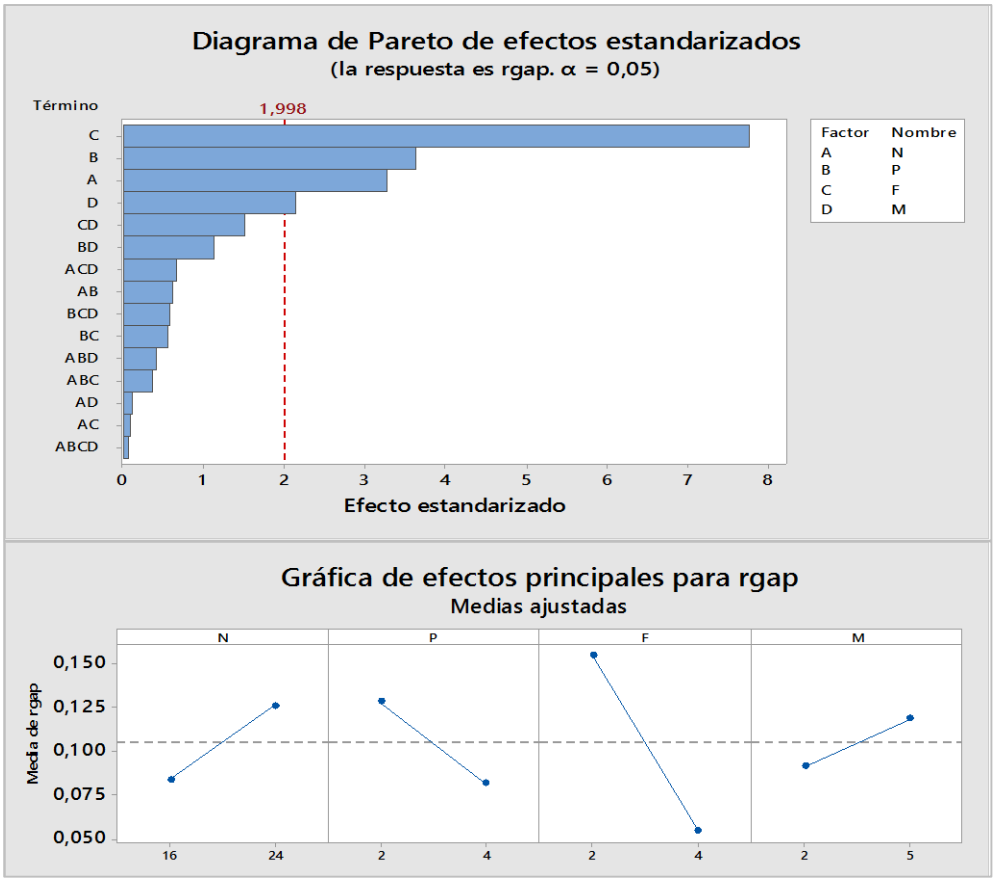


Figura 9. Diagrama de Pareto y gráfica de efectos principales para rgap.

Con los datos de la tabla 19 y de la figura 9 se puede concluir que el gap relativo se ve afectado por los cuatro factores, aunque en mayor medida por el número de fábricas y en menor medida por el número de máquinas; ratificando la tendencia observada en las tablas de promedios antes presentadas, en las que el rgap crece con el aumento de trabajos y máquinas mientras que decrece con el aumento de fábricas y productos.

8. Algoritmo Basado en VND

El algoritmo desarrollado se ha basado en VND como respuesta a recomendaciones encontradas en la literatura, en las cuales se destaca el desempeño de esta metaheurística en comparación a algoritmos constructivos para la solución de problemas de tipo DPFSP (Ruiz & Naderi, 2009) y DAPFSP (Ruiz et al., 2013). Además, en su diseño se incorporan movimientos de reinserción y el operador genético de cruce como estrategias de exploración en las secuencias de ensamble.

8.1. Representación de la Solución

La solución del problema DAPFSP-SDST se representa a través de un vector de tamaño 1×(**N+F+P+1)** que contiene la secuencia de procesamiento de trabajos para cada fábrica y la secuencia de ensamble de los productos; en este vector las diferentes secuencias están separadas por un trabajo dummy “0”.

Por ejemplo, para un problema con 8 trabajos, 3 productos, 2 fábricas y 5 máquinas, una posible solución se representa de la siguiente manera (figura 10):

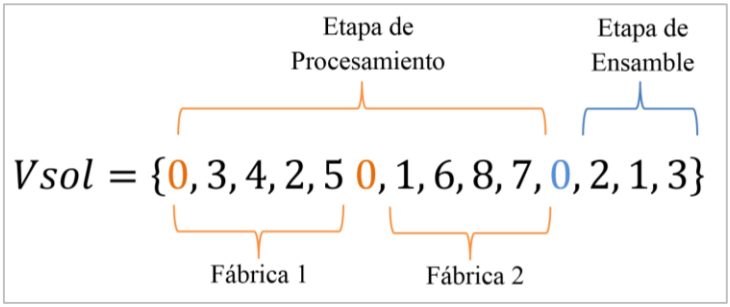


Figura 10. Ejemplo de vector solución para un problema N8P3F2M5

En la representación elegida para la solución no se tienen en cuenta las máquinas, considerando que cada fábrica es un *flowshop* permutado donde la secuencia de trabajos para la primera máquina se mantiene para las siguientes, en otras palabas, solo existe una secuencia por fábrica.

8.2. Descripción del Algoritmo

8.2.1. Generación de la solución inicial. Para la generación de la solución inicial se crea un conjunto de secuencias de ensamble semillas de tamaño TAM_PISC donde la primera secuencia se obtiene aplicando una regla de asignación SPT[6] (figura 11) y las restantes TAM_PISC-1 son generadas de manera aleatoria.

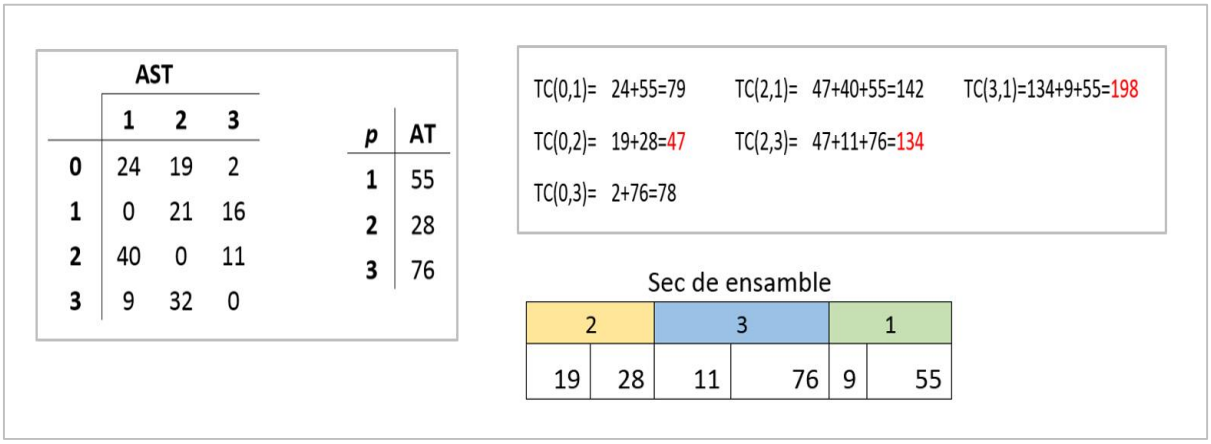


Figura 11. Ejemplo de secuencia de ensamble de tres productos generada con SPT

Para mejorar las secuencias semillas, a cada una de ellas se le aplica un método de reinserción en el que cada producto se inserta cada INT posiciones, con lo que de cada secuencia semilla se obtiene un grupo de $P * \left\lfloor \frac{(P-1)}{INT} \right\rfloor$ secuencias; si la mejor secuencia dentro de un grupo (menor *makespan*) es mejor que su secuencia semilla, se actualiza la secuencia semilla y a ésta se le aplica

[6] SPT: En esta regla de asignación se programan primero los productos con menor tiempo de completamiento después de asignarse.

reinserción, este procedimiento se lleva a cabo MEJ veces o hasta que la secuencia semilla no pueda ser mejorada. Las secuencias de ensamble mejoradas conforman la población inicial para las posteriores operaciones de cruce (figuras 12 y 13), adicionalmente la secuencia con menor *makespan* de éstas es la primera solución que entra a la búsqueda en los vecindarios.

***Inicio**: crear TAM_PISC secuencias de ensamble aleatorias.*

Repetir la siguiente secuencia hasta alcanzar el criterio de parada: *K = TAM_PISC*

> ***Repetir*** la siguiente secuencia hasta alcanzar el criterio de parada: *salir = verdadero*
>
> (1) Tomar la *K-esima* secuencia de ensamble *X*
> (2) Generar las secuencias de ensamble *X'* reinsertando cada producto en cada INT posiciones de *X*.
> (3) Generar una secuencia de producción para cada *X'* utilizando SPT y calcular el makespan a cada una.
> (4) Encontrar la *X''* con el menor makespan. (óptimo local)
> (5) Si el óptimo local es mejor o igual que la secuencia inicial, moverse a él. *(X←X'')* y *(cont=cont+1);* de lo contrario *(K←K+1)* y *(salir = verdadero).*
> (6) Si (*cont>=MEJ),* entonces *(K←K+1) y (salir = verdadero);* de lo contrario volver al paso 1.

Figura 12. Pseudocódigo de la generación de la solución inicial

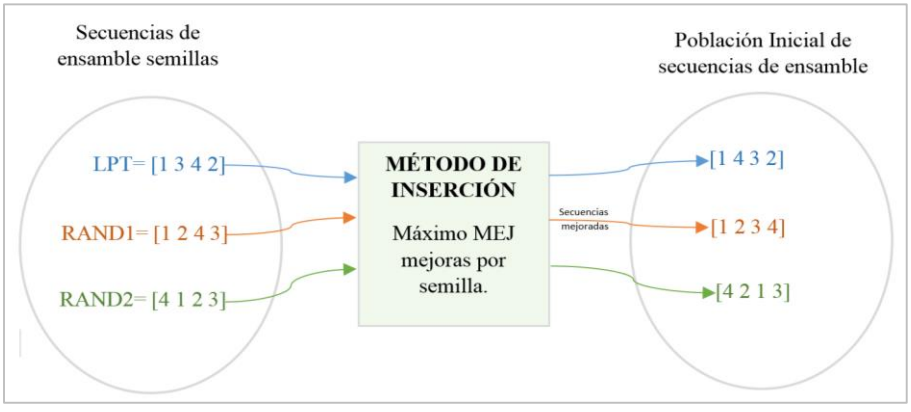


Figura 13. Diagrama de generación de población inicial de secuencias de ensamble.

Se debe aclarar que, dentro de la población de secuencias de ensamble, cada una de éstas hace parte de un vector solución en el que la etapa de procesamiento se genera con una regla de

asignación SPT en la cual se priorizan los trabajos que pertenecen a los primeros productos programados en la secuencia de ensamble (figura 14).

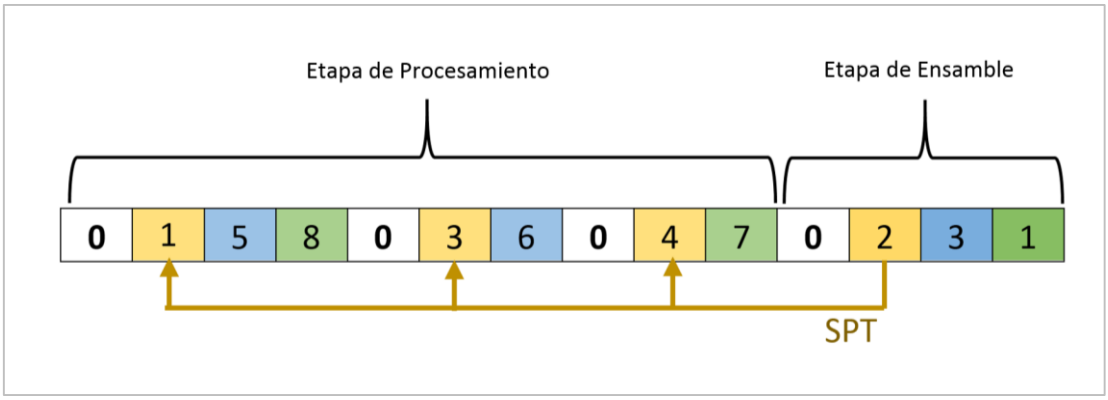


Figura 14. Ejemplo de vector solución en la población inicial de secuencias de ensamble. *Nota*: Al ser el producto 2 el primero en la secuencia de ensamble, los trabajos 1,3 y 4 que lo conforman se asignan primero a las fábricas con una regla SPT

8.2.2. Búsqueda en Vecindarios. Para la búsqueda en los vecindarios se toma un vector solución, el cual es obtenido de la población inicial de secuencias de ensamble o del cruce entre dos de ellas. El objetivo de esta etapa del algoritmo es mejorar la secuencia de procesamiento que se obtuvo con la aplicación del SPT, por esta razón la secuencia de ensamble se mantiene constante durante la búsqueda en los vecindarios (figura 15).

***Inicio**: Definir un conjunto de estructuras de vecindario N (J) y tomar la mejor solución obtenida X de la etapa anterior.*

Repetir Hasta que se obtenga mejora o se cumpla el criterio de parada

(1) Hacer (J ← 1) y (cont←1);
(2) **Repetir** hasta que (J = Jmax) o (cont = 100)
   a. Explorar el entorno: obtener la mejor solución X' del J-ésimo entorno de X.
   b. Si la solución obtenida X' es mejor que X, entonces (X←X'), (cont←cont+1) y (J←1); de lo contrario (J←J+1)

Figura 15. Pseudocódigo de búsqueda en estructuras de vecindario.

El algoritmo se compone de cuatro estructuras de vecindario que se detallan a continuación:

i) Reinserción: Esta estructura explora nuevas soluciones cambiando los trabajos de su posición actual, insertándolos en todas las demás posiciones de la secuencia de procesamiento; de esta manera se explora todo el vecindario dándole la connotación de búsqueda descendente. Por cada solución actual se prueban $N * (N + F - 2)$ soluciones de las cuales se busca aquella con el mínimo *makespan*; si existe más de una solución que cumpla este criterio, se selecciona una de ellas de manera aleatoria. La solución actual se remplaza por la solución seleccionada si esta última presenta menor *makespan*, de lo contrario, pasa a la siguiente estructura.

ii) Intercambio de dos elementos (swap): Esta estructura explora nuevas soluciones evaluando todos aquellos movimientos que intercambian las posiciones de dos trabajos de la secuencia de procesamiento actual (descendencia). Por cada solución actual se prueban $(N * \frac{N-1}{2})$ soluciones de las cuales se busca aquella con el mínimo *makespan*; si existe más de una solución que cumpla este criterio, se selecciona una de ellas de manera aleatoria. La solución actual se remplaza por la solución seleccionada si esta última presenta menor *makespan*, de lo contrario, pasa a la siguiente estructura.

iii) Intercambio de trabajos con una misma posición en las fábricas: Esta estructura toma los trabajos que ocupan una misma posición dentro de la secuencia de las diferentes fábricas y los reordena aleatoriamente, reubicando cada trabajo en la misma posición, pero en una fábrica diferente. Por cada solución actual se prueban en promedio $(N * F)$ soluciones de las cuales se busca aquella con el mínimo *makespan*; si existe más de una solución que cumpla este criterio, se selecciona una de ellas de manera aleatoria. La solución actual se remplaza por la solución seleccionada si esta última presenta menor *makespan*, de lo contrario, pasa a la siguiente estructura.

iv) Intercambio de trabajos que pertenecen a un mismo producto: Esta estructura toma las posiciones de los trabajos que pertenecen a un mismo producto y reordena los trabajos aleatoriamente en dichas posiciones. Por cada solución actual se prueban en promedio (N) soluciones de las cuales se busca aquella con el mínimo *makespan*; si existe más de una solución que cumpla este criterio, se selecciona una de ellas de manera aleatoria. La solución actual se remplaza por la solución seleccionada si esta última presenta menor *makespan*, de lo contrario, pasa a la siguiente estructura.

Cada vez que se encuentre una mejor solución, ésta se evalúa desde la primera estructura; sin embargo, para evitar que el algoritmo quede atrapado en la búsqueda de vecindarios, se establece un límite máximo de 100 mejoras y una lista de verificación que garantiza que el algoritmo no vuelva a una solución ya probada.

Las estructuras uno y dos buscan la mejor solución posible dentro del vecindario definido (búsqueda descendente); mientras que las estructuras tres y cuatro están diseñadas para explorar nuevas soluciones evitando que algoritmo quede atrapado en óptimos locales producto de la limitación en los movimientos de los vecindarios uno y dos.

8.2.3. Cruce de secuencias de ensamble. En esta etapa del algoritmo se cruza la mejor secuencia de ensamble obtenida hasta el momento con una de las secuencias de ensamble de la población inicial, la cual es escogida de manera aleatoria con una distribución de probabilidad lineal, donde la mejor solución tiene cinco veces más probabilidad de ser escogida que la peor.

Una vez seleccionados los dos padres, se escoge el punto de corte mediante la generación de un aleatorio que se limita entre la tercera y la antepenúltima posición de la secuencia de ensamble con el fin de evitar cruces en los cuales se cambie menos de dos elementos, pues las operaciones de

reinserción anteriormente probadas para las soluciones de la población inicial ya descartaron una posible mejora con estos cambios.

Los dos hijos generados con la operación de cruce son comparados aplicando SPT para la etapa de procesamiento y calculando el *makespan* para cada uno; el hijo con el mejor desempeño es sometido nuevamente al proceso de reinserción y posteriormente se aplican las estructuras de vecindarios para mejorar la secuencia de procesamiento.

La solución obtenida es comparada con la solución final[7]; si el *makespan* es menor, ésta pasaría a ser la nueva solución final y la secuencia de ensamble pasaría a ser parte de la población inicial, remplazando a la peor para mantener el tamaño de dicha población constante. Si no es mejor que la solución final, pero sí mejor que alguna de las que conforman la población inicial, ésta pasaría a remplazar la peor (figura 16).

[7] Se llama solución final a la mejor secuencia encontrada por el algoritmo en un momento dado.

***Inicio**:* Cuando FINAL ≠ 0

Repetir hasta que se cumpla el criterio de parada

(1) Generar un punto de cruce aleatorio
(2) Tomar la mejor secuencia de ensamble conocida A.
(3) Seleccionar una secuencia de ensamble de la población inicial B
(4) Cruzar A y B para generar A' y B'.
(5) Generar una solución para A' y B' aplicando SPT para la secuencia de producción.
(6) Encontrar la secuencia con menor *makespan X*
(7) *Repetir hasta alcanzar el criterio de parada: salir = verdadero*
    a. Generar las secuencias de ensamble *X'* reinsertando cada producto en cada INT posiciones de *X*.
    b. Generar una solución para cada X' utilizando SPT y calcular el makespan a cada una.
    c. Buscar X'' con el menor makespan. (optimo local)
    d. Si el óptimo local X'' es mejor o igual que la secuencia inicial X, moverse a él. (X←X'') y (cont=cont+1); de lo contrario (salir = verdadero).
    e. Si (cont>=MEJ), entonces (salir = verdadero); de lo contrario volver al paso a.
(8) Probar la secuencia X en las estructuras de vecindario.
(9) Si X es mejor que FINAL, entonces (FINAL←X) y la secuencia de ensamble de X remplaza la peor secuencia de ensamble de la población inicial.
(10) Si (cruces>= 5), entonces (salir=verdadero) de lo contrario volver al paso 1.

Figura 16. Pseudocódigo del cruce de secuencias de ensamble

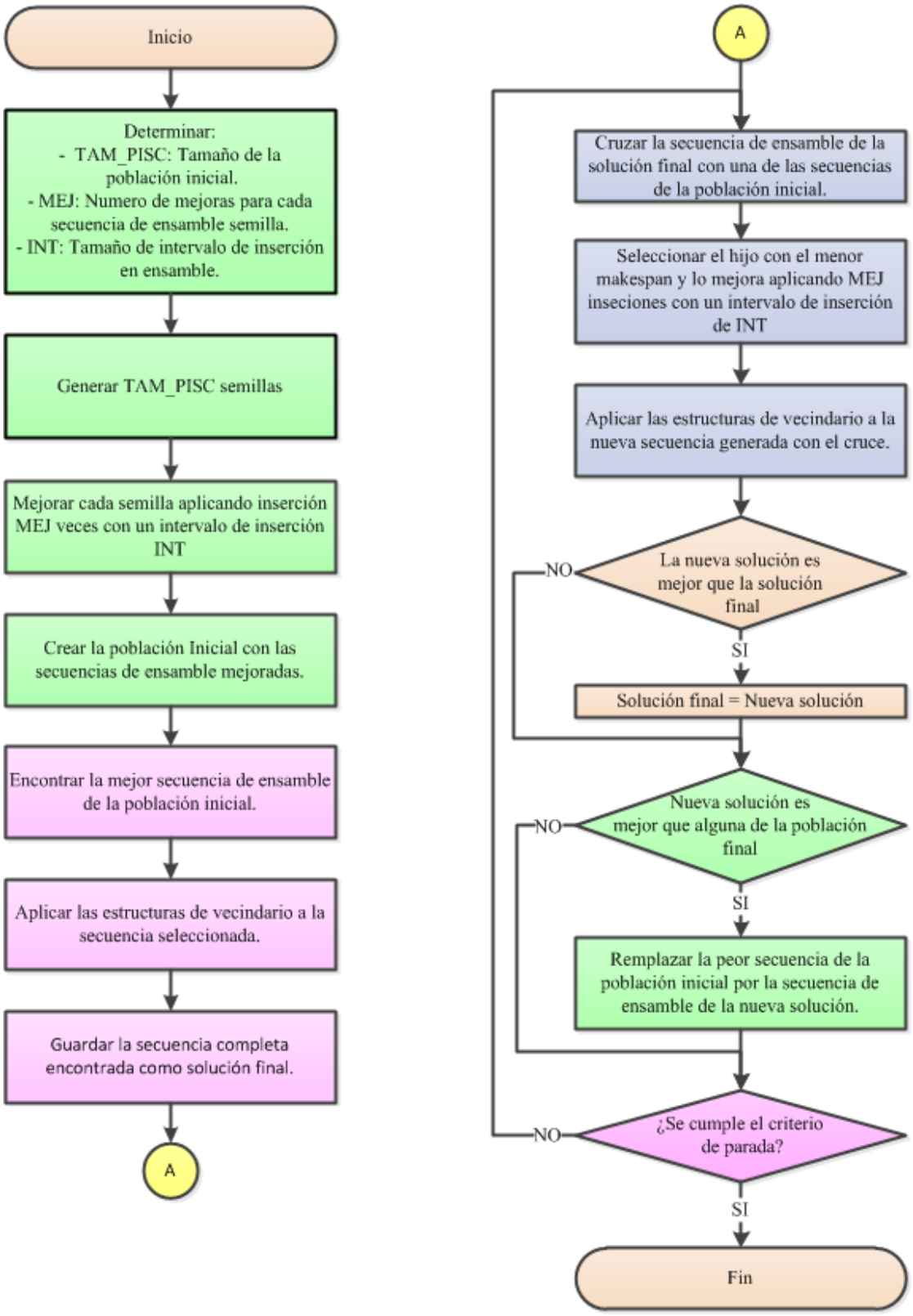


Figura 17. Diagrama de Flujo del algoritmo

8.3. Implementación en MATLAB

La implementación del algoritmo se llevó a cabo en el software Matlab R2017a, en el cual se diseñó una rutina principal llamada *ALGORITMO_VND* compuesta por varias subrutinas definidas como funciones. La rutina principal contiene la secuencia de funciones que conforman el algoritmo y guarda los resultados en un archivo xlsx, mientras que las subrutinas ejecutan las diferentes etapas del algoritmo. Los scripts y el detalle de las rutinas se encuentran en el apéndice G.

El algoritmo se ejecuta en un computador con procesador Intel Core i5-4570 de 3,2 GHz, memoria RAM de 8 GB y un sistema operativo Windows 10 Pro de 64 bits

9. Calibración de Parámetros del Algoritmo

Para la ejecución del algoritmo diseñado y con el propósito de lograr mejores resultados; se lleva a cabo un proceso de calibración en el que se determinan qué factores tienen efecto significativo sobre los resultados del algoritmo y cuál es la combinación de niveles de estos factores con los que se logra un mejor desempeño; es decir, menores valores de makespan.

Los parámetros que se analizan en la calibración son: el número de secuencias de ensamble semillas o tamaño de la población inicial (TAM_PISC), el número máximo de mejoras permitidas para cada semilla mediante reinserción (MEJ) y el tamaño del intervalo entre una reinserción y otra para la secuencia de ensamble (INT).

Para la calibración se crea un diseño factorial completo 2^k de tres factores, en el cual se utilizan tres réplicas para cada tratamiento, lo cual representa 24 corridas por cada instancia analizada

(apéndice H). Este diseño es aplicado a cuatro instancias de diferente tamaño, las cuales corresponden a una de las más grandes, una de las más pequeñas y dos intermedias de las *test instances* trabajadas por Ruiz et al. (2015):

- N100-P30-F4-M5
- N100-P50-F8-M20
- N200-P30-F4-M5
- N200-P50-F8-M20

Los niveles máximos de cada factor se establecieron bajo el criterio de no exceder los 30 minutos en tiempo de ejecución para la instancia más grande (N200-P50-F8-M20), mientras que los niveles bajos se establecieron como la mitad de los anteriores (tabla 20).

Tabla 20.
Factores y niveles utilizados en cada diseño factorial 2^k para calibrar los parámetros del algoritmo.

Factor	Nivel bajo	Nivel alto
TAM_PISC	5	10
MEJ	10	20
INT	$N^2/10000$	$2N^2/10000$

A partir del diseño experimental, se efectúa un análisis de varianza, el cual permite identificar qué factores tienen efecto significativo sobre la variable respuesta *makespan* y cuál es el tratamiento con el que se obtienen mejores resultados para esta variable.

Para revisar si los datos cumplen con el supuesto de homocedasticidad, se utiliza el test de igualdad de varianzas de Levene en el cual se contrastan las siguientes hipótesis:

H_o: *Todas las varianzas son iguales*

H_a: *Por lo menos una varianza es diferente*

Mediante la aplicación del test en Minitab18, se encuentra que en todas las instancias el valor-p supera el 5% de significancia (tabla 21), por lo que se concluye que no hay evidencia estadística significativa para rechazar la igualdad de las varianzas.

Además, con el fin de validar los supuestos de normalidad e independencia, en los siguientes análisis de varianza se presenta la gráfica de probabilidad normal y la de residuos vs orden por cada instancia.

Tabla 21
Resultados del test de Levene aplicado a las instancias de calibración.

Instancia	Estadístico de Prueba	Valor-p
N100-P30-F4-M5	0,32	0,933
N100-P50-F8-M20	0,43	0,869
N200-P30-F4-M5	1,14	0,387
N200-P50-F8-M20	0,35	0,920

9.1. Análisis de Varianza para la Instancia N100-P30-F4-M5

Dado que en la gráfica de probabilidad normal de la figura 18 no es clara la normalidad de los residuos al observarse puntos alejados de la línea, este supuesto se verifica a través de la prueba de Kolmogorov Smirnov en la que se obtiene un valor-p mayor al 15%, con el que se concluye que no hay evidencia significativa para rechazar la normalidad de los residuos. Igualmente, al no percibirse tendencias ni patrones en el orden de los residuos, se prueba su independencia.

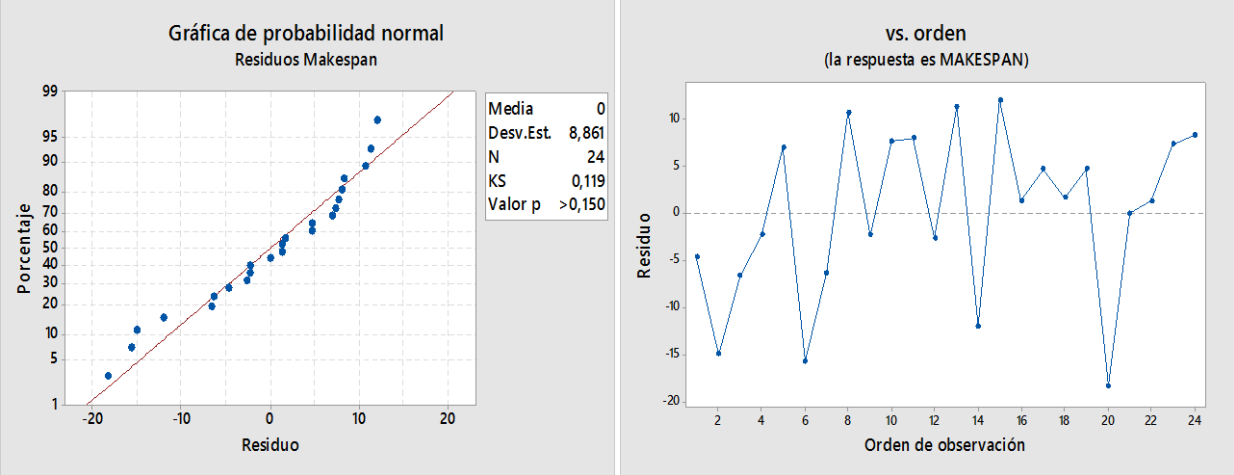


Figura 18. Gráficas de normalidad e independencia de residuos para N100P30F4M5.

Según la tabla 22 y la figura 19, se concluye que los factores con efecto significativo en la variable *makespan* son el número de mejoras MEJ y el intervalo entre inserciones INT al presentar valores-p menores al 5%; destacándose el efecto del parámetro MEJ al superar en gran medida a los demás efectos. Así mismo se encuentra que el *makespan* disminuye con niveles altos de TAM_PISC y MEJ y niveles bajos de INT.

Tabla 22
ANOVA para la instancia N100-P30-F4-M5

Fuente	**GL**	**SC Ajust.**	**MC Ajust.**	**Valor F**	**Valor p**
Modelo	7	9831,6	1404,52	12,44	0,000
Lineal	3	9592,5	3197,49	28,33	0,000
TAM_PISC	1	260,0	260,04	2,30	0,149
MEJ	1	8550,4	8550,38	75,75	0,000
INT	1	782,0	782,04	6,93	0,018
Interacciones de 2 términos	3	239,1	79,71	0,71	0,562
TAM_PISC*MEJ	1	2,0	2,04	0,02	0,895
TAM_PISC*INT	1	92,0	92,04	0,82	0,380
MEJ*INT	1	145,0	145,04	1,28	0,274
Interacciones de 3 términos	1	0,0	0,04	0,00	0,985
TAM_PISC*MEJ*INT	1	0,0	0,04	0,00	0,985
Error	16	1806,0	112,87		
Total	23	11637,6			

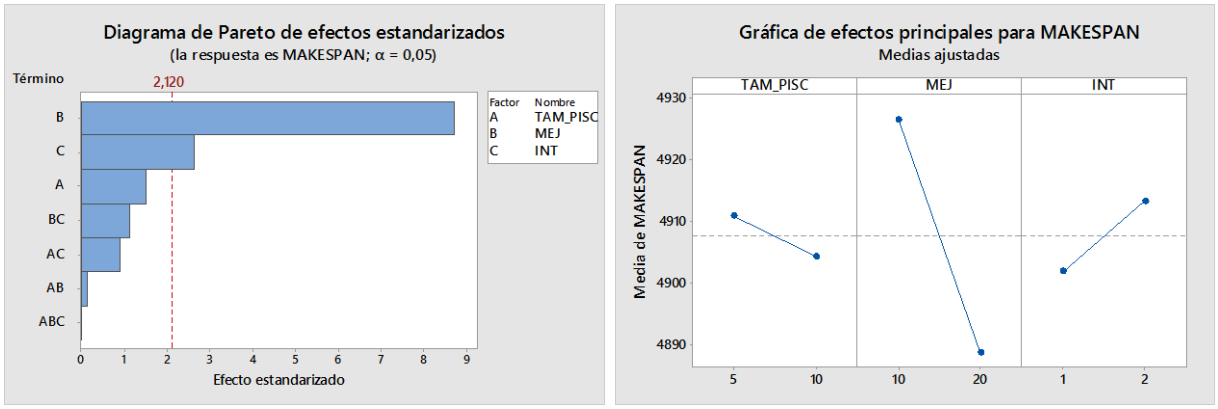


Figura 19. Diagrama de Pareto y efectos principales para makespan en N100P30F4M5

9.2. Análisis de Varianza para la Instancia N100-P50-F8-M20

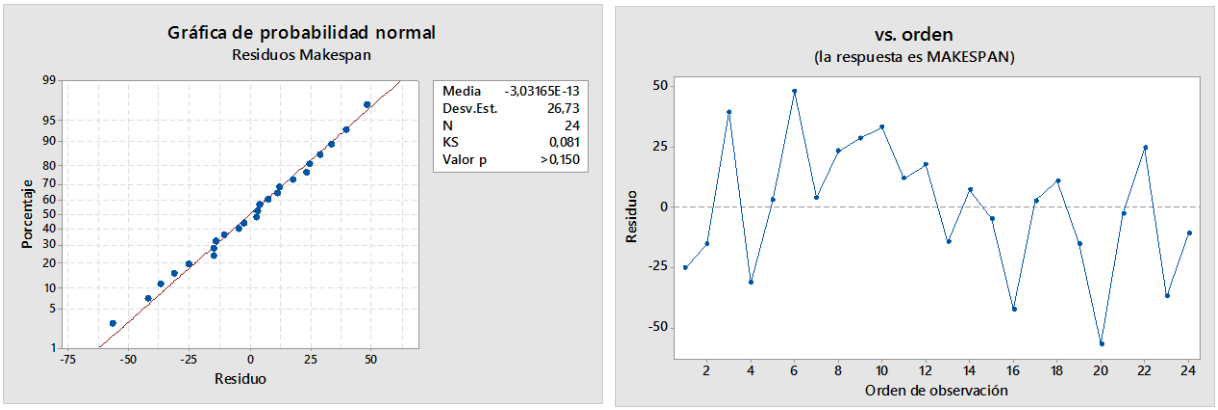


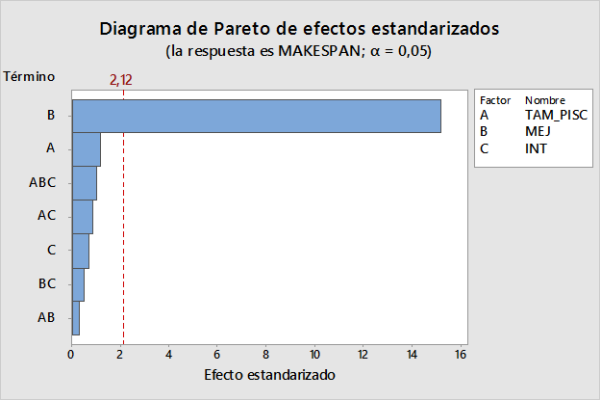
Figura 20. Gráficas de normalidad e independencia de residuos para N100P50F8M20

Con la Figura 20 se pueden comprobar los supuestos de normalidad e independencia dado que el valor-p obtenido con el test de Kolmogorov Smirnov es superior al 5% de significancia y no se evidencian patrones o tendencias en la gráfica de residuos vs orden.

Los datos presentes en la tabla 23 y en la figura 21 evidencian que, para la instancia analizada, el único factor con efecto significativo en el *makespan* es el número de mejoras MEJ, el cual supera de manera importante el efecto de los demás. El efecto del parámetro MEJ es reducir el *makespan* en su nivel alto, por lo cual se espera un mejor desempeño del algoritmo entre más mejoras se permitan por cada semilla.

Tabla 23.
ANOVA para la instancia N100-P50-F8-M20.

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	241294	34471	33,57	0,000
Lineal	3	239130	79710	77,64	0,000
TAM_PISC	1	1380	1380	1,34	0,263
MEJ	1	237208	237208	231,04	0,000
INT	1	541	541	0,53	0,478
Interacciones de 2 términos	3	1124	375	0,36	0,779
TAM_PISC*MEJ	1	113	113	0,11	0,745
TAM_PISC*INT	1	771	771	0,75	0,399
MEJ*INT	1	241	241	0,23	0,635
Interacciones de 3 términos	1	1040	1040	1,01	0,329
TAM_PISC*MEJ*INT	1	1040	1040	1,01	0,329
Error	16	16427	1027		
Total	23	257721			



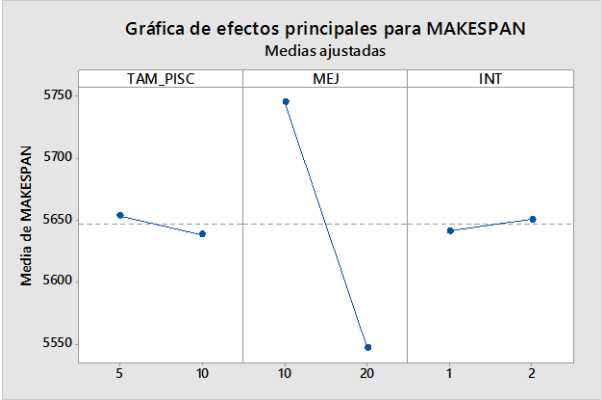


Figura 21. Diagrama de Pareto y efectos principales para makespan en N100P50F8M20

9.3. Análisis de Varianza para la Instancia N200-P30-F4-M5

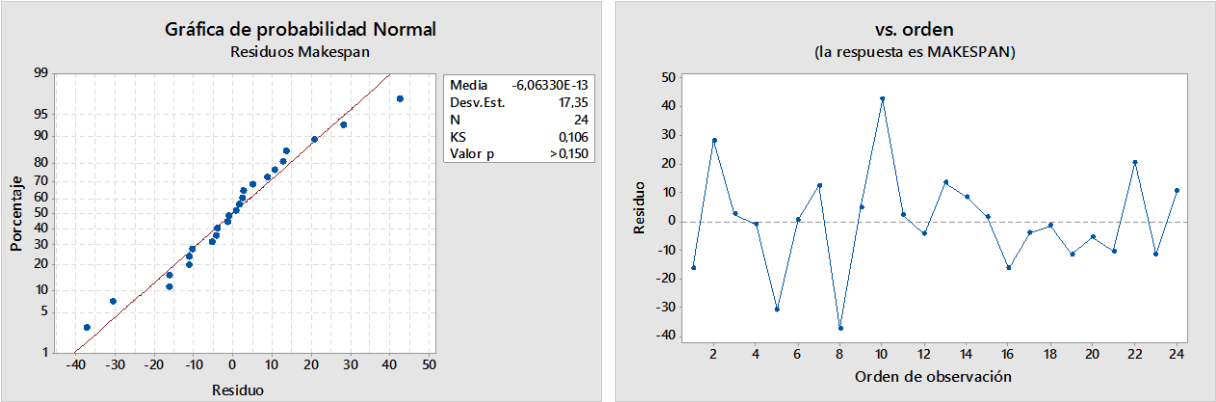


Figura 22. Gráficas de normalidad e independencia de residuos para N200P30F4M5

Tabla 24.
ANOVA para la instancia N200-P30-F4-M5.

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	43226,3	6175,2	14,27	0,000
Lineal	3	42894,5	14298,2	33,04	0,000
TAM_PISC	1	1162,0	1162,0	2,68	0,121
MEJ	1	34277,0	34277,0	79,20	0,000
INT	1	7455,4	7455,4	17,23	0,001
Interacciones de 2 términos	3	331,5	110,5	0,26	0,856
TAM_PISC*MEJ	1	0,4	0,4	0,00	0,977
TAM_PISC*INT	1	155,0	155,0	0,36	0,558
MEJ*INT	1	176,0	176,0	0,41	0,533
Interacciones de 3 términos	1	0,4	0,4	0,00	0,977
TAM_PISC*MEJ*INT	1	0,4	0,4	0,00	0,977
Error	16	6924,7	432,8		
Total	23	50151,0			

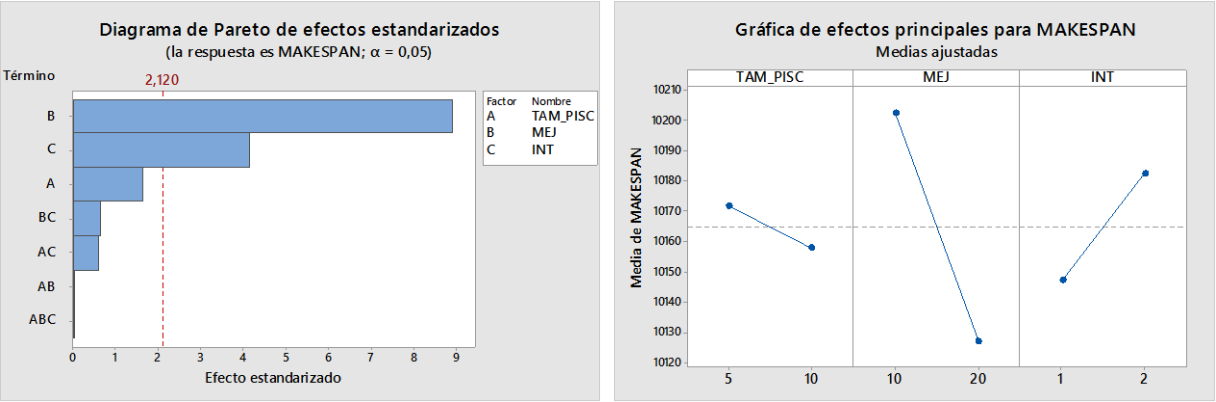


Figura 23. Diagrama de Pareto y efectos principales para makespan en N200P30F4M5

Bajo los supuestos de normalidad e independencia demostrados a partir de la figura 22 y observando los datos de la tabla 24 y la figura 23, se puede concluir que, para la instancia analizada, el *makespan* está afectado significativamente por el número de mejoras MEJ y el tamaño del salto de reinserción INT, el cual es el mismo comportamiento encontrado con la primera instancia, aunque el número de trabajos se haya duplicado. En cuanto al sentido de los efectos, se manifiesta una disminución del *makespan* con un nivel alto de mejoras, tal y como se ha evidenciado en las dos anteriores instancias.

9.4. Análisis de Varianza para la Instancia N200-P50-F8-M20

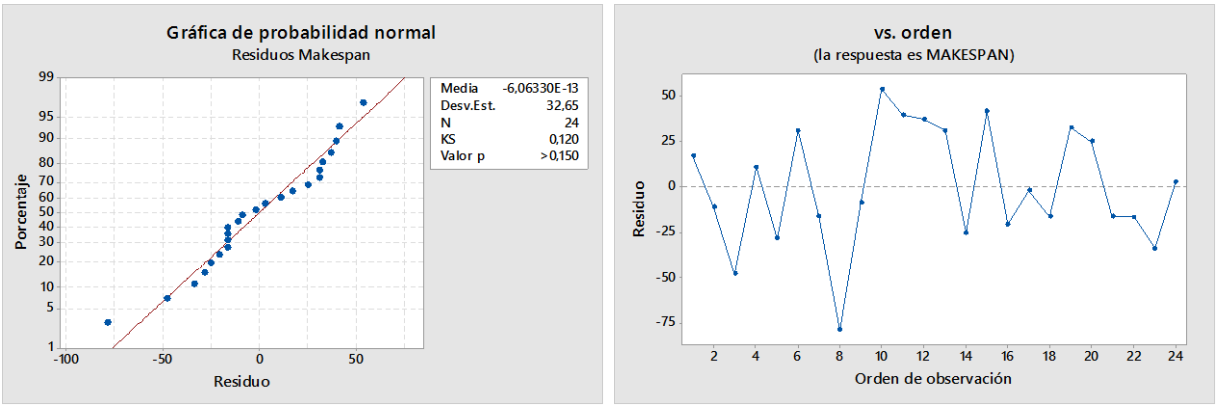
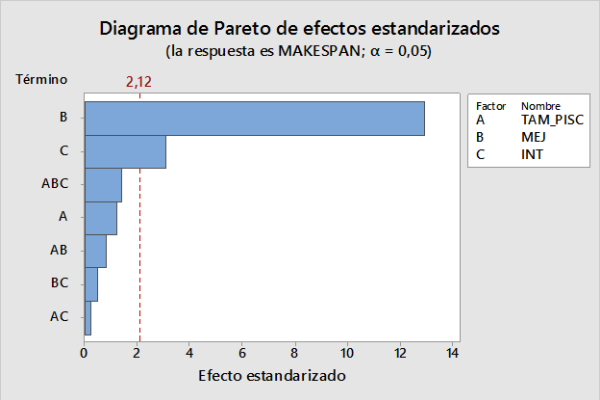


Figura 24. Gráficas de normalidad e independencia de residuos para N200P50F8M20.

Tabla 25
ANOVA para la instancia N200-P50-F8-M20.

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	277323	39618	25,86	0,000
Lineal	3	272701	90900	59,32	0,000
TAM_PISC	1	2360	2360	1,54	0,232
MEJ	1	255441	255441	166,70	0,000
INT	1	14900	14900	9,72	0,007
Interacciones de 2 términos	3	1629	543	0,35	0,787
TAM_PISC*MEJ	1	1148	1148	0,75	0,399
TAM_PISC*INT	1	113	113	0,07	0,790
MEJ*INT	1	368	368	0,24	0,631
Interacciones de 3 términos	1	2993	2993	1,95	0,181
TAM_PISC*MEJ*INT	1	2993	2993	1,95	0,181
Error	16	24517	1532		
Total	23	301839			



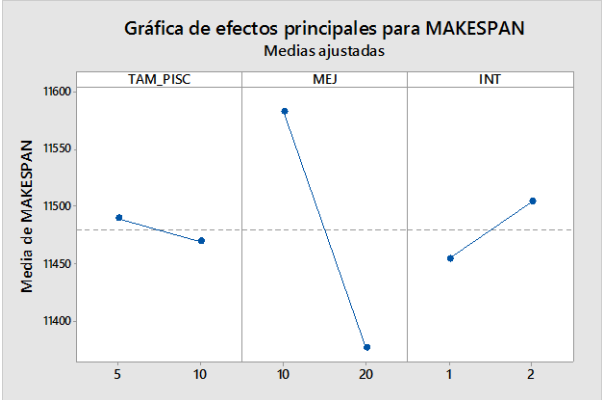


Figura 25. Diagrama de Pareto y efectos principales para makespan en N200P50F8M20

De acuerdo con la tabla 25 y la figura 25, los factores con efecto significativo en el *makespan* son el número de mejoras MEJ y el tamaño del salto en la reinserción INT, además se aprecia que la variable *makespan* disminuye a medida que incrementa el número de mejoras MEJ o disminuye el intervalo entre inserciones INT.

Como conclusión, se tiene que los factores MEJ e INT ejercen un efecto significativo sobre el *makespan* para la mayoría de las instancias probadas y por ende se debe considerar la mejor

combinación de los niveles de éstos para asegurar buenos resultados en la variable respuesta. Por otra parte, el factor TAM_PISC no muestra efectos estadísticamente significativos en ninguno de los cuatro diseños experimentales.

Según los resultados del diseño experimental, la combinación de factores que consigue el mejor desempeño del algoritmo incluye a MEJ en su nivel alto y a INT en su nivel bajo; sin embargo, para determinar el nivel adecuado del factor TAM_PISC se tiene en cuenta su influencia en el tiempo de ejecución, por lo cual se realiza el diseño experimental para las cuatro instancias evaluando la variable TIME (Apéndice I). Los resultados de este análisis permiten concluir que los efectos del factor TAM_PISC sobre la variable TIME son significativos, generando un aumento en el tiempo de ejecución para el nivel alto de este factor, por lo cual es conveniente usar el nivel bajo para garantizar la eficiencia del algoritmo planteado.

10. Evaluación de Desempeño del Algoritmo

10.1. Comparación del Algoritmo con el Modelo MILP en Instancias Pequeñas Para Fábricas Heterogéneas

Para evaluar el desempeño del algoritmo propuesto, éste se ejecuta en las 720 instancias pequeñas generadas para fábricas heterogéneas[8] y los resultados se comparan con los obtenidos

[8] Las instancias se encuentran en Mendeley a través del DOI: 10.17632/6cpkw9v9gy.2

mediante el modelo MILP para las mismas instancias. Los resultados del algoritmo para estas instancias se encuentran en el Apéndice J.

En la comparación, el algoritmo obtuvo el mismo *makespan* que el modelo MILP en 287 instancias, de las cuales 204 soluciones corresponden a la óptima; así mismo, se obtiene que el algoritmo logra mejores soluciones que el MILP en 197 instancias y peores soluciones en 236.

De la misma manera, en la tabla 26 se percibe que el algoritmo tiene mejor calidad de respuestas al exhibir un RPD promedio levemente más bajo que el modelo MILP; sin embargo, para ratificar qué método tiene el mejor desempeño es preciso analizar el tiempo que tardan los métodos para lograr dicha calidad de respuesta.

Tabla 26.
RPD del algoritmo y el modelo MILP en los casos de comparación para pequeñas instancias con fábricas heterogéneas

Comparación	**Instancias**	**Porcentaje de instancias**	**RPD (%)**	
			MILP	ALG
$S_{MILP} = ALG_{SOL}$	287	39,86%	0	0
$S_{MILP} < ALG_{SOL}$	236	32,78%	0	2,7
$S_{MILP} > ALG_{SOL}$	197	27,36%	3,28	0

Los dos métodos se comparan también a través del porcentaje de desviación relativa (RPD), para el cual se considera como mejor solución encontrada (*Best)* el menor valor de *makespan* entre la solución obtenida con el modelo MILP (S_{MILP}) y el algoritmo (ALG_{SOL}):

$$Best = \boldsymbol{Min}(\boldsymbol{S_{MILP}}, \boldsymbol{ALG_{SOL}})$$

$$\boldsymbol{RPD(ALG)} = \frac{\boldsymbol{ALG_{SOL} - Best}}{\boldsymbol{Best}}$$

$$\boldsymbol{RPD(MILP)} = \frac{\boldsymbol{S_{MILP} - Best}}{\boldsymbol{Best}}$$

En la tabla 27 se presentan los RPD promedio para cada combinación de los factores N, F y P. El número de máquinas M no se tiene en cuenta en esta categorización porque éste no es un componente de decisión en el MILP y el algoritmo.

En la tabla 27 se observa cómo a medida que el tamaño de las instancias crece, para el modelo MILP es más difícil encontrar la solución óptima o una buena solución, mientras que el algoritmo se vuelve más competitivo, lo que se demuestra con los valores decrecientes del RPD para el algoritmo y crecientes para el MILP.

Aunque los dos métodos tienen un desempeño similar en cuanto a calidad de la respuesta, al evaluar los tiempos de ejecución promedio se encuentra que el algoritmo obtiene soluciones de buena calidad en un tiempo significativamente menor que el modelo MILP, puesto que el promedio de ejecución del algoritmo es 0,051s en tanto que el del modelo MILP es 546,52s. Esta diferencia en los tiempos de ejecución de los dos métodos es más evidente en las instancias con más de doce trabajos.

Tabla 27.
RPD y tiempo de ejecución para el modelo MILP y el algoritmo propuesto sobre la mejor solución en instancias pequeñas.

			RPD (%)		**Tiempo(a)**	
N	***P***	***F***	MILP	ALG	MILP	ALG
8	2	2	0	2,21	4,86	0,019
8	2	3	0	2,06	2,12	0,015
8	2	4	0	1,31	0,67	0,017
8	3	2	0	1,69	3,79	0,025
8	3	3	0	1,05	1,62	0,027
8	3	4	0	0,68	1,06	0,034
8	4	2	0	2,38	12,71	0,048
8	4	3	0	0,12	1,43	0,054
8	4	4	0	0,55	1,43	0,051
12	2	2	0,15	3,30	857,76	0,033
12	2	3	0,18	1,01	556,39	0,032
12	2	4	0	1,31	297,08	0,034
12	3	2	0,21	1,42	719,62	0,043
12	3	3	0	1,80	432,15	0,035
12	3	4	0	0,49	70,139	0,038
12	4	2	0,54	1,40	836,10	0,067
12	4	3	0	0,46	124,51	0,066
12	4	4	0	0,21	40,01	0,061
16	2	2	0,66	1,47	900	0,032
16	2	3	0,66	0,98	900	0,036
16	2	4	0,04	1,03	841,30	0,030
16	3	2	1,69	0,49	900	0,043
16	3	3	0,92	0,61	900	0,036
16	3	4	0,29	0,14	779,43	0,039
16	4	2	2,17	0,69	900	0,068
16	4	3	0,23	0,73	828,46	0,066
16	4	4	0,15	0,39	662,12	0,064
24	2	2	2,84	0,32	900	0,089
24	2	3	1,86	0,38	900	0,061
24	2	4	1,16	0,17	900	0,067
24	3	2	3,67	0,01	900	0,074
24	3	3	2,69	0,16	900	0,064
24	3	4	3,26	0,17	900	0,063
24	4	2	4,00	0,11	900	0,10
24	4	3	3,12	0,43	900	0,11
24	4	4	1,86	0,25	900	0,11
PROMEDIOS TOTALES			**0,90**	**0,89**	546,52	0,051

10.2. Evaluación del Algoritmo en Instancias Grandes para Fábricas Homogéneas

Dado que el modelo MILP no encontró solución factible en la mayoría de instancias grandes para fábricas heterogéneas dentro de los límites de tiempo (capítulo 7.3.1); no existen datos suficientes con los cuáles compararse para determinar el desempeño del algoritmo. Por esta razón,

el algoritmo se prueba con las 270 instancias para fábricas homogéneas evaluadas por Ruiz et al. (2015) y los resultados (ALG_{SOL}) se comparan con las mejores soluciones de la literatura ($BEST_{SOL}$).

Para ejecutar el algoritmo con la condición de fábricas homogéneas, se toman los tiempos de procesamiento y alistamiento de la primera fábrica y se utilizan en las demás. Los resultados del algoritmo en estas instancias y los datos de la comparación se encuentran en el apéndice K.

Tabla 28.
RPD del algoritmo y el modelo MILP en los casos de comparación para pequeñas instancias con fábricas homogéneas.

Comparación	Instancias	Porcentaje de instancias	RPD (%)	
			$BEST_{SOL}$	ALG_{SOL}
$BEST_{SOL} < ALG_{SOL}$	201	74,44%	-	0,79
$BEST_{SOL} > ALG_{SOL}$	69	25,56%	1,69	-

Al realizar la comparación del algoritmo con las mejores soluciones de la literatura para instancias grandes con fábricas homogéneas (tabla 28), se encuentra que el algoritmo mejoró 69 de las 270 instancias, y en las 201 instancias que no logró mejorar, el RPD fue tan solo del 0,79% en promedio; es decir, si bien el algoritmo no pudo mejorar la solución para ciertas instancias, la solución que éste encuentra está muy cercana a la mejor solución encontrada, por lo cual se le atribuye un buen desempeño teniendo en cuenta que su diseño está basado en fábricas heterogéneas.

El tiempo de ejecución del algoritmo para las 270 instancias fue de 32 horas aproximadamente, con un tiempo promedio de 424 segundos por instancia (7 minutos y 6 segundos aproximadamente).

11. Resultados del Algoritmo en Instancias grandes para Fábricas Heterogéneas.

El algoritmo fue ejecutado en las 270 instancias grandes para fábricas heterogéneas; sin embargo, los resultados no se pudieron comparar con el modelo MILP dado que éste encontró muy pocas soluciones factibles para estas instancias; y tampoco con valores de referencia en la literatura puesto que no se han desarrollado otros métodos de solución para este problema hasta el momento.

Los resultados de la ejecución del modelo se presentan con el fin de que sirvan como base para futuros estudios en los que se desarrollen nuevos métodos de solución o se tengan en cuenta otras consideraciones.

La ejecución de las 270 instancias tardó aproximadamente 35,84 horas; es decir, 478s en promedio por cada instancia. Los resultados obtenidos en esta etapa se encuentran en el apéndice J.

12. Conclusiones

- A través del análisis bibliométrico se logra identificar un aumento significativo de los estudios relacionados con el DAPFSP a partir de 2011, corroborando la importancia actual que

tiene el problema dentro de la comunidad científica. Mientras que la revisión de literatura con un enfoque evolutivo permitió validar la oportunidad de investigación del DAPFSP-SDST con fábricas heterogéneas como un tema de interés en la investigación de operaciones por atender requerimientos reales de la industria.

- El modelo MILP desarrollado logró actualizar la mejor solución encontrada en 769 de las 900 instancias pequeñas evaluadas para fábricas homogéneas, aparte de encontrar la misma solución en 72 de ellas; es decir, se encontró la mejor solución para el 93,5% de las instancias.

- Mediante el diseño experimental realizado al modelo MILP se logra concluir que los cuatro factores (N, P, F y M) influyen en la calidad de la solución cuando se establece un tiempo límite de 900s. Sin embargo, el mayor efecto lo tienen el número de fábricas F y el número de productos P, donde los niveles altos de estos factores generan un mejor desempeño del modelo.

- La dificultad del modelo MILP para encontrar la solución óptima a medida que crece el tamaño de las instancias pequeñas y la baja cantidad de soluciones factibles que éste logra encontrar en instancias grandes para el tiempo límite de 1800s, evidencian la importancia de los modelos metaheurísticos en la programación de problemas de tipo NP-Hard en los que los modelos exactos no logran obtener soluciones competitivas en tiempos computacionales razonables.

- En la calibración del algoritmo se encuentra que para las cuatro instancias analizadas los mejores resultados para *makespan* se producen cuando el número de mejoras mediante reinserción se encuentra en su nivel alto y el intervalo entre inserciones se encuentra en su nivel bajo. Mientras que para el tamaño de la población inicial se selecciona el nivel bajo, pues aun cuando su efecto no es significativo para el *makespan*, éste sí aumenta significativamente el tiempo de ejecución afectando la eficiencia del algoritmo.

- El algoritmo presenta un mejor desempeño que el MILP para instancias pequeñas con fábricas heterogéneas, dado que, aunque ambos métodos encuentran respuestas de calidad similar en cuanto a *makespan*, existe una diferencia en tiempo de ejecución bastante significativa, donde el modelo MILP tarda en promedio 546s en resolver una instancia mientras el algoritmo lo hace en 0,051s.

- El algoritmo presenta un buen desempeño en comparación a las heurísticas y metaheurísticas usadas en la literatura para el problema con fábricas homogéneas, ya que aún compitiendo en un problema para el cual no fue estructurado, éste logró actualizar la mejor solución encontrada en 69 instancias de las 270 analizadas, y en las 201 instancias en las que no encontró mejor solución, el RPD promedio fue del 0,79%; es decir, encontró una solución muy cercana.

13. Recomendaciones

Para futuras investigaciones se recomienda:

- Desarrollar nuevos modelos exactos y algoritmos en los que se utilicen las instancias generadas en la presente investigación y con los que se puedan comparar y validar los resultados obtenidos para el DAPFSP-SDST con fábricas heterogéneas.

- Contemplar nuevas consideraciones para el problema que le permitan adaptarse a diferentes realidades de la industria, tales como interoperabilidad, transporte entre fábricas, buffers limitados y transporte entre la etapa de procesamiento y la de ensamble,

- Evaluar la influencia de las fabricas heterogéneas en el desempeño de las heurísticas y metaheurísticas desarrolladas para el DAPFSP-SDST en la literatura, a través de diseños experimentales que permitan ver variaciones en la calidad de las respuestas y el tiempo de ejecución de los algoritmos.

- Desarrollar algoritmos evaluando diferentes funciones objetivo o multiobjetivo para analizar el comportamiento del problema cuando se consideran otros criterios de decisión como el costo de producción, número y/o tiempo promedio de entregas tardías.

- Permitir soluciones iguales o peores en las estructuras de vecindarios con el fin de fortalecer la exploración del algoritmo y evitar la convergencia temprana en óptimos locales.

- Generar la población inicial de secuencias de ensamble con diferentes heurísticas constructivas para evaluar con cuales de ellas el algoritmo logra un mejor desempeño.

- Mejorar la eficiencia del modelo MILP y del algoritmo mediante el uso de programación paralela con la cual se busque dar solución a instancias grandes en menor tiempo computacional.

Referencias Bibliográficas

Adenso, B., & Dowsland, K. (2003). Heuristic design and fundamentals of the Simulated Annealing. *Revista Iberoamericana de Inteligencia Artificial, 19*, 93-102.

Al-Anzi, F., & Allahverdi, A. (2006). A Hybrid Tabu Search Heuristic for the Two-Stage Assembly Scheduling Problem. *International Journal of Operations Research, 3*(2), 109-119.

Al-Anzi, F., & Allahverdi, A. (2006). A PSO and a Tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application. *Computers & Operations Research, 33*(4), 1056-1080.

Behnamian, J., & Fatemi Ghomi, S. (2016). A survey of multifactory scheduling. *Journal of Intelligent Manufacturing*, 231-249.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 268-308.

Chan, F., Chung, S., & Chan, P. (2005). An adaptive genetic algorithm with dominated genes for distributed scheduling problems. *Expert Systems with Applications*, 364-371.

Chaovalitwongse, P., Werner, F., Reodecha, M., & Jungwattanakit, J. (2009). A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. *Computers & Operations Research, 36*(2), 358-378.

Chen, S., Chen,Y., Chang, P., & Chen, M. (2012). Extended artificial chromosomes genetic algorithm for permutation flowshop scheduling problems. *Computers & Industrial Engineering*, 536-545.

Diaz, A. (2009). *Diseño estadístico de experimentos* (segunda ed.). Medellín: Universidad de Antioquia.

Dong, X., Huang, H., & Chen, P. (2009). An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Computers and Operations Research*, 1664-1669 .

Douglas, H., & Francisco, P. (1996). Multi-agent mediator architecture for distributed manufacturing. *Journal of Intelligent Manufacturing, 7*, 257-270.

Fatemi Ghomi, S., Jolai, F., Mozdgir, A., & Navaei, J. (2013). Two-stage assembly flow-shop scheduling problem with non-identical assembly machines considering setup times. *International Journal of Production Research, 51*(12), 3625-3642.

Fatemi Ghomi, S., Jolai, F., Mozdgir, A., & Navaei, J. (2014). Heuristics for an assembly flow-shop with non-identical assembly machines and sequence dependent setup times to minimize sum of holding and delay costs. *Computers & Operations Research, 44*, 52-65.

Fernandez-Viagas, V., Ruiz, R., & Framinan, J. (2017). Invited Review: A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal Of Operational Research*, 707-721.

Fisher, M. (1985). An applications oriented guide to Lagrangian relaxation. *Interfaces, 15*(2), 10–21.

Gao, J., & Chen, R. (2011). A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Computational Intelligence Systems,* 497-508.

Gao, J., & Chen, R. (2013). An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 641-651.

Gao, J., Chen, R., & Liu, Y. (2012). A Knowledge-based Genetic Algorithm for Permutation Flowshop Scheduling Problems with Multiple Factories. *International Journal of Advancements in Computing Technology*, 121-129.

Garey, M., & Johnson, D. (1979). *A Guide to the Theory of NP-Completeness.* New York: W. H. FREEMAN AND COMPANY.

Garey, M., Johnson, D., & Sethi, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 117-129.

Glover, F., & Kochenberger, G. (2003). *Handbook of metaheuristics.* Dordrecht: Kluwer Academic Publishers.

Glover, F., & Melián, B. (2003). Tabu search. *Revista Iberoamericana de Inteligencia Artificial, 19*, 29-48.

Guirao Goris, S. (2015). Utilidad y tipos de revisión de literatura. *Ene*.

Gupta, J. (1979). A review of flowshop scheduling research. En *Disaggregation* (págs. 363-388).

Hirsch, J. (2005). An index to quantify an individual's scientific research output. *PNAS*, 16569–16572.

Jia, H., Nee, A., Fuh, J., & Zhang, Y. (2003). A modified genetic algorithm for distributed scheduling problems. *Journal of intelligent manufacturing*, 351-362.

Johnson, S. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics, 1*(1), 61-68.

Kahn, K., Castellion, G., Griffin, A. (2004). *The PDMA Handbook of new product development*. New York, EEUU: Wiley.

Koulamas, C., & Kyparisis, G. (Junio de 2001). The three-stage assembly flowshop scheduling problem. *Computers & Operations Research, 28*(7), 689-704.

Kusiak, A. (1988). Scheduling flexible machining and assembly systems. *Annals of Operations Research, 15*(1), 337-352.

Lawler, E., & Wood, D. (1966). Branch-and-Bound Methods: A Survey. *Operations Research, 4*(4), 699-719.

Lee, C.-Y., Cheng, T., & Lin, B. (1993). Miniminzing the Makespan in the 3-Machine Assembly-Type flowshop Scheduling problem. *Management Science, 39*(5), 616-625.

Li, X., Zhang, X., Minghao, Y., & Wang, J. (2015). A genetic algorithm for the distributed assembly permutation flowshop scheduling problem. *Evolutionary Computation (CEC).*

Li, Z.-Y., Duan, W.-Z., Ji, M.-C., & Yang, Y.-X. (2016). The distributed permutation flowshop scheduling problem with different transport timetables and loading capacities. *Congress on Evolutionary Computation (CEC).*

Lin, J., & Zhang, S. (2016). An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem. *Journal of computers and Industrial Engineering, 97*(C), 128-136.

Liu, B., Wang, K., & Zhang, R. (2016). Variable neighborhood based memetic algorithm for distributed assembly permutation flowshop. *Evolutionary Computation (CEC)*.

Liu, H., & Gao, L. (2010). A Discrete Electromagnetism-Like Mechanism Algorithm for Solving Distributed Permutation Flowshop Scheduling Problem. *International Conference on Manufacturing Automation*, 156-163.

Maboudian, Y., & Shafaei, R. (2009). Modeling a bi-criteria two stage assembly flow shop scheduling problem with sequence dependent setup times. *Industrial Engineering and Engineering management*. doi:10.1109/IEEM.2009.5373156

Maboudian, Y., Tavakkoli-M, R., Hatami, S., & Ebrahimnejad, S. (2010). Two meta-heuristics for three-stage assembly flowshop scheduling with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology, 20*(9), 1153-1164.

Mahdavi, I., Shirazi, B., Cho, N., Sahebjamnia, N., & Ghobadi, S. (2008). Modeling an e-based real-time quality control information system in distributed manufacturing shops. *Computers in Industry, 59*, 759-766.

Martí, R. (2003). Procedimientos Metaheurísticos en Optimización Combinatoria. *Departament d'Estadística i Investigació Operativa, Facultat de Matemàtiques,Universitat de València, 1*, 1-62.

Matz, T., & Terry, W. (1989). An object-oriented programming paradigm for synchronous manufacturing. *Computers and Engineering, 17,* 124-129.

Montgomery, D. C. (2004). Diseño y análisis de experimentos. México D.F: Limusa.

Moratto, E., & Pérez, L. (2016). *Metaheurística basada en el Algoritmo Competitivo Imperialista (ICA) aplicada a la solución del problema de Flow shop Híbrido (HFS) con máquinas paralelas no relacionadas* (Tesis de pregrado). Universidad Industrial de Santander, Bucaramanga, Colombia.

Osman , I., & Kelly, J. (1996). *Metaheuristics:Theory and Applications.* Boston: Kluwer Academic.

Oviedo, D., & Valdivieso, K. (2016). *Un Algoritmo Híbrido para el problema de ruteo de vehículos con tiempos de viaje estocásticos y ventanas de tiempo suave* (Tesis de pregrado). Universidad Industrial de Santander,Bucaramanga, Colombia.

Pan, Q., Tasgetiren, M., & Liang, Y. (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers and Industrial Engineering,* 795-816.

Potts , C., & Hariri, A. (1997). A branch and bound algorithm for the two-stage assembly scheduling problem. *European Journal of Operational Research, 103*(3), 547-556.

Potts, C., S.V.Sebast'janov, S. V., Van Wassenhove, L., & Zwaneveld , C. (1995). The Two-Stage Assembly Scheduling Problem: Complexity and Approximation. *Operations Research, 43*(2), 346-355.

Ruiz , R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research,* 2033–2049.

Ruiz, R., & Naderi, B. (2009). Variable neighborhood descent methods for the distributed permutation flowshop problem. *Multidisciplinary International Conference on Scheduling: Theory and Applications*.

Ruiz, R., & Naderi, B. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, 754-768.

Ruiz, R., & Naderi, R. (2014). A scatter search algorithm for the distributed permutation flowshop scheduling problem. *European Journal of Operational Research,* 323-334.

Ruiz, R., Hatami, S., & Romano, C.-A. (2013). The Distributed Assembly Permutation Flowshop Scheduling Problem. *International Journal of Production Research, 51*(17), 5292-5308.

Ruiz, R., Hatami, S., & Romano, C.-A. (2014). Simple constructive heuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. *International Conference on Control, Decision and Information Technologies*.

Ruiz, R., Hatami, S., & Romano, C.-A. (2015). Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. *International Journal of Production Economics, 169*, 76-88.

Sadiq, S., & Habib, Y. (2000). *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems.*

Taillard, E. (1990). Some efficient heuristic methods for the flow-shop sequencing problem. *European Journal of Operational Research*, 65-74.

Wilkinson, S. J., Cortier, A., Shah, N., & Pantelides, C. (1996). Integrated production and distribution scheduling on a Europe-wide basis. *Computers & Chemical Engineering*, 1275-1280.

Williams, J. F. (1981). Heuristic techniques for simultaneous scheduling of production and distribution in multi-echelon structures: Theory and empirical comparisons. *Management Science*, 336 - 352.

Xiangtao, L., Xin, Z., Minghao, Y., & Jianan, W. (2015). A genetic algorithm for the distributed assembly permutation flowshop scheduling problem. IEEE.

Yokoyama, M. (2008). Flow-shop scheduling with setup and assembly operations. *European Journal of Operational Research, 187*(3), 1184-1195.