Solución al "Flexible Job Shop Problem" con Tiempos de Alistamiento Dependientes de la Secuencia Mediante un Algoritmo Híbrido Genético

Lina Mayerly Lozano Suarez

Fabian Alexander Torres Cárdenas

Trabajo de Grado para Optar al Título de Ingeniería Industrial

Director

Carlos Eduardo Díaz Bohórquez

Magister en Ingeniería Industrial

Universidad Industrial de Santander

Facultad de Ingenierías Físico-Mecánicas

Escuela de Estudios Industriales y Empresariales

Bucaramanga

2018

Agradecimientos

A Dios, por permitirme llegar a esta etapa en mi vida.

A mi madre Rosa, por ser el apoyo incondicional en cada uno de mis momentos vividos y por su gran amor que le da sentido a mi vida todos los días.

A mi padre Cristóbal, por el gran esfuerzo que me ha ayudado a superar dificultades a lo largo de este camino.

A mi hermana Deisy, por ser la motivación para superar las dificultades y por ser una persona incondicional en todos los momentos.

A mis abuelos maternos por brindarme el apoyo durante los inicios de mi carrera.

A mis amigos y compañeros de Opalo por su apoyo a lo largo del proyecto.

Al profesor Carlos Díaz por su orientación y dedicación para el desarrollo del proyecto.

Lina Mayerly Lozano Suarez

A Dios, por darme la fortaleza para superar las dificultades.

A mi madre María Cecilia, por su incondicional apoyo y motivación cada día.

A mi tío Ignacio por su apoyo y respaldo a lo largo de la carrera en los momentos difíciles.

A mis hermanas Diana, Nataly y Dania por ser mi fortaleza y compañía durante este proceso.

A mis amigos y compañeros de Opalo por su apoyo a lo largo del proyecto.

Al profesor Carlos Díaz por su orientación y dedicación para el desarrollo del proyecto.

Fabian Alexander Torres Cárdenas

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Lista de Apéndices

(Los apéndices están adjuntos en el CD y puede visualizarlos en base de datos de la biblioteca UIS)

RESUMEN

Título: Solución al "Flexible Job Shop Problem con Tiempos de Alistamiento Dependientes de la Secuencia" (SDST-FJSP) Mediante un Algoritmo Híbrido Genético*

Autores: Lina Mayerly Lozano Suarez
Fabian Alexander Torres Cárdenas**

Palabras clave: Flexible Job-Shop Scheduling Problem, Algoritmo Híbrido Genético, Tiempos de Alistamiento Dependientes de la Secuencia,

Descripción:

En la presente investigación se aborda el problema de programación de operaciones que involucra la secuenciación y asignación de máquinas, denominado en la literatura como Flexible Job Shop Problem con Tiempos de Alistamiento Dependientes de la Secuencia (SDST-FJSP), este problema es considerado NP.Hard. Es uno de los problemas que tiene gran relevancia porque se aproxima a ambientes reales de producción donde es necesario tener en cuenta que después de una determinada operación se requiera realizar un ajuste de herramientas, trasladar un producto, limpieza, entre otras actividades de alistamiento.

Para dar solución a este problema se diseñó un Algoritmo Hibrido Genético donde parte de la población inicial se generó mediante la metaheurística Recocido Simulado, el criterio de selección utilizado fue la selección por torneo, se utilizaron dos tipos de operadores de cruce uno que modifica la secuencia de las operaciones, cruce POX (Precedence Preserving Order-based)y otro para la asignación de las máquinas, cruce de dos puntos y por último el operador de mutación que modifica la secuencia de las operaciones respetando las restricciones de precedencia PPS (Precedence Preserving Shift).

Se realizó un diseño de experimentos 2^3 con el fin de determinar cómo influyen los factores en la variable respuesta Makespan dando como resultado que el factor tamaño de población tenía efecto significativo sobre el Makespan.

Finalmente se validó el algoritmo propuesto con 20 instancias de la literatura y se comparó con diferentes metodologías propuestas por diversos autores, dando como resultado buenas soluciones en las instancias de menor tamaño. Sin embargo para las instancias de mayor tamaño el algoritmo presenta dificultad para encontrar buenas soluciones comparado con la mejor solución encontrada.

* Proyecto de grado.

** Facultad de Ingenierías Físico Mecánicas. Escuela de Estudios Industriales y Empresariales .Programa de Ingeniería Industrial. Director M.Sc. Carlos Eduardo Díaz Bohórquez.

ABSTRACT

Title: Solution to "Flexible Job Shop Problem with sequence dependent setup time" (SDST-FJSP) using an Algorithm Hybrid Genetic †

Authors: Lina Mayerly Lozano Suarez
Fabian Alexander Torres Cárdenas**

Keywords: Flexible Job-Shop Scheduling Problem, Algorithm Hybrid Genetic, Sequence Dependent Setup Times.

Description:

In the present investigation is addressed the problem of operations programming which involves the sequencing and assignment of machines, referred to in the literature as Flexible Job Shop Problem with Sequence Dependent Setup Times (SDST-FJSP), this problem is considered NP.Hard. It is one of the problems that has great relevance because it approaches real production environments where it is necessary to take into account that after a certain operation it is required to make an adjustment of tools, move a product, cleaning, among other activities of setup.

To solve this problem, a Algorithm Hybrid Genetic was designed where part of the initial population was generated by the Metaheuristic Simulated Annealing, the selection criterion used was the tournament selection, two types of crossover operators one that modifies the sequence of operations, POX crossover and another for the assignment of the machines, two points crossover and finally the mutation operator that modifies the sequence of operations respecting the restrictions of precedence PPS (Precedence Preserving Shift).

An experimental design was carried out 2^3 in order to determine how factors influence the Makespan response variable resulting in the fact that the population size factor had a significant effect on the Makespan.

Finally, the proposed algorithm was validated with 20 instances of the literature and compared with different methodologies proposed by different authors, resulting in good solutions in smaller instances. However, for larger instances, the algorithm has difficulty finding good solutions compared to the best solution found.

† Proyecto de grado.

** Facultad de Ingenierías Físico Mecánicas. Escuela de Estudios Industriales y Empresariales .Programa de Ingeniería Industrial. Director M.Sc. Carlos Eduardo Díaz Bohórquez.

Introducción

Actualmente las organizaciones necesitan ser competitivas, para esto es necesario una eficiente gestión de operaciones que permita la Optimización de los recursos disponibles y la satisfacción del cliente. Para lograr lo anterior, es necesario tomar decisiones de tipo operativo, estas son las más rutinarias como por ejemplo; la programación de trabajos cada día, que necesariamente involucra actividades tales como; asignación de trabajos a máquinas disponibles y secuenciación de las diferentes operaciones en las máquinas.

Las actividades descritas anteriormente representan un entorno productivo enfocado en procesos "*Job Shop*" el cual es conveniente para organizaciones que manejan una gran variedad de productos con bajos volúmenes de producción. Este tipo de taller de producción es intermitente, es decir; cada trabajo está formado por un conjunto de operaciones, cada una de las cuales deben ser procesadas en una determinada máquina con un tiempo definido. La característica principal es que cada trabajo tiene su propio flujo.

Debido a la gran importancia del problema y que se adapta a las necesidades actuales del mercado se ha decidido tratar una generalización del *Job Shop*, el *Flexible Job Shop*; este se diferencia del anterior porque de cada tarea tiene asignado un grupo de máquinas en las cuales podrá ser procesada, y no solo una máquina en específico. Por consiguiente, además de la secuenciación debe realizarse un proceso de asignación. El *Flexible Job Shop* (FJSP) incluye dos subproblemas: 1) asignación de máquinas: seleccionar una máquina de un conjunto de máquinas candidatas para las operaciones y 2) la secuenciación: la programación de las operaciones en todas las máquinas para obtener una solución factible.

Adicionalmente se considerará dentro del *Flexible Job Shop*, que los tiempos de alistamiento son dependientes de la secuencia (SDST-FJSP), tal característica se presenta en diferentes sistemas productivos porque después de una determinada operación se requiere realizar un ajuste de herramientas, cambiar plantillas, limpieza, entre otras, para que la máquina esté lista para la siguiente operación.

Este tipo de problema es "considerado NP-Hard, incorporándole la flexibilidad aumenta en gran medida la complejidad del problema, porque se requiere un nivel adicional de decisión es decir, la selección de máquinas en las que debería realizarse las operaciones de cada trabajo" (Brandimarte, 1993, págs. 157-183).

Los enfoques para resolver el problema de programación de operaciones en un Flexible Job Shop pueden ser clasificados en dos categorías; enfoques concurrentes y enfoques jerárquicos. Para la solución de este tipo de problema se han empleado diferentes métodos: métodos exactos y métodos de aproximación tales como las heurísticas y Metaheurísticas, siendo estas las que han arrojado mejores resultados a tiempos computacionales razonables. Entre las más utilizadas se encuentran: Búsqueda Tabú (TS), Algoritmo Genético (GA), Búsqueda de Vecindario Variable (VNS) y Recocido Simulado (SA).

En este proyecto se abordará el SDST-FJSP de la siguiente manera; se utilizará una técnica con un enfoque integrado, utilizando las Metaheurísticas de Recocido Simulado para la construcción de una porción de la población inicial y un Algoritmo Genético en la búsqueda de la optimización.

La presente propuesta está organizada de la siguiente manera: en el capítulo 4 se encuentra la revisión de literatura relevante sobre el FJSP y FJSP-SDST. Luego en el capítulo 5 se presenta el Marco Teórico. El capítulo 6 está la descripción del Recocido Simulado y en capítulo 7 la descripción del Algoritmo Genético. Luego en el capítulo 8 se presenta el problema a tratar en este trabajo, el Flexible Job Shop Con Tiempos de Alistamiento Dependientes de La Secuencia (SDST-FJSP). En el capítulo 9 se describe el algoritmo propuesto para resolver el FJSP-SDST. Luego en el capítulo 10 se muestra la validación del algoritmo propuesto y en el capítulo 11 se muestran los resultados computacionales. Por último en el capítulo 12 y capitulo 13 se muestran las conclusiones del presente proyecto y las recomendaciones para futuras investigaciones.

Tabla 1.

Cumplimiento de objetivos del proyecto

Objetivo	Cumplimiento
Realizar una revisión de literatura sobre el Flexible Job Shop, así como los enfoques propuestos por distintos autores.	Capítulo 4 y Apéndice A
Definir el problema, los supuestos y el modelo matemático que lo represente	Capítulo 8
Diseñar un Algoritmo Híbrido Genético y programarlo en MATLAB, que sirva para la solución del problema *Flexible Job Shop* con las características anteriormente nombradas.	Capítulo 9 y Apéndice B
Comparar los resultados encontrados del problema en la literatura de diferentes autores en sus métodos de solución aplicados al problema "*Flexible Job Shop* con tiempos de alistamiento dependientes de la secuencia	Capítulo 11
Elaborar un artículo de carácter publicable que contenga los resultados del proyecto de investigación	Apéndice G

1. Planteamiento del problema

La programación de operaciones ha evolucionado a lo largo de los años para así, representar los entornos reales de los sistemas productivos. Cabe resaltar que es un área importante dentro de las organizaciones porque permite la optimización de los recursos y cumplir con las exigencias del cliente.

En los últimos años diferentes autores han ampliado las investigaciones del JSP al FJSP, agregándole el componente de flexibilidad de recursos, en este caso máquinas, para aumentar el rendimiento, gestionar el mantenimiento preventivo o abordar el desglose y otros eventos imprevistos que modifican la disponibilidad de recursos. El FJSP consiste en dos subproblemas: 1) asignación de máquinas: seleccionar una máquina de un conjunto de máquinas candidatas para las operaciones y 2) la secuenciación: la programación de las operaciones en todas las máquinas, es decir la secuencia con la que se debe ejecutar las operaciones de los trabajos en cada una de las máquinas asignadas.

El FJSP con la característica de tiempos de alistamiento dependientes de la secuencia (SDST-FJSP), se manifiesta en diferentes sistemas productivos porque después de una determinada operación se requiere realizar un ajuste de herramientas, cambiar plantillas, limpieza, entre otras actividades según corresponda para que la máquina esté lista para la siguiente operación.

El problema consiste en la programación a realizar para un conjunto de i trabajos i= {1, 2,…, n}, en un conjunto de k máquinas k= {1, 2,..., Ma}. Un trabajo i está formado por una secuencia de operaciones $OP_{i,1}$, $OP_{i,2}$,..., $OP_{i,Oi}$ que se realizan una tras otra en el orden dado. Cada operación $OP_{i,j,}$ es decir, la operación j del trabajo i, se puede ejecutar en cualquiera de entre

un subconjunto de máquinas compatibles ($MP_{i,j} \subseteq M_a$). El FJSP es una extensión del problema JSP, en este caso $MP_{i,j}=1$ para cada operación. Los tiempos de alistamiento dependen de la secuencia de los trabajos, cuando una de las operaciones del trabajo i se procesa antes de una de las del trabajo i' (i' ≠ i) en la máquina k, el tiempo de alistamiento dependiente de la secuencia es $S_{i,i',k} > 0$.

Debido a la alta complejidad NP-Hard del problema, se utilizan métodos aproximados para encontrar una solución. En el presente proyecto se pretende estudiar el SDST-FJSP mediante un Algoritmo Híbrido Genético el cual usará el Recocido Simulado para la construcción de parte de la población inicial y un Algoritmo Genético en búsqueda de la Optimización.

2. Justificación del Proyecto

El problema de asignación y secuenciación de máquinas FJSP es uno de los problemas que se encuentran con más frecuencia en los ambientes de producción y servicios. La planificación de los procesos en el área de producción, requiere ser eficiente en el manejo de los recursos y a la vez cumplir con las exigencias del cliente. Lo cual, se convierte en una ventaja competitiva para el desarrollo y mejora de las organizaciones.

Según se observa en la revisión de literatura diferentes autores están trabajando en estas líneas de investigación, en busca de mejorar las soluciones del problema aplicando tanto heurísticas y Metaheurísticas. Ampliando su nivel de dificultad agregando factores que aproximan el problema a la realidad como lo son: tiempo de procesamiento variable, interrupciones, averías aleatorias de máquinas, restricciones de transporte, tiempo de procesamiento limitado, entre otros.

Por tal motivo en el grupo de investigación OPALO se muestra interés en seguir avanzando en el estudio de este tipo de problemas debido a la importancia que tiene dentro de los sistemas productivos de bienes y servicios en las organizaciones empresariales.

En esta investigación se pretende entender el funcionamiento del problema y el método propuesto con el fin de determinar una solución factible a bajo costo, comparándola con las soluciones encontradas en la literatura.

3. Objetivos

3.1 Objetivo general

Evaluar un Algoritmo Híbrido Genético como alternativa de solución al problema "*Flexible Job Shop* con tiempos de alistamiento dependientes de la secuencia" (SDST-FJSP).

3.2 Objetivos específicos

- Realizar una revisión de literatura sobre el *Flexible Job Shop,* así como los enfoques propuestos por distintos autores.
- Definir el problema, los supuestos y el modelo matemático que lo represente.
- Diseñar un Algoritmo Híbrido Genético y programarlo en MATLAB, que sirva para la solución del problema *Flexible Job Shop* con las características anteriormente nombradas.
- Comparar los resultados encontrados del problema en la literatura de diferentes autores en sus métodos de solución aplicados al problema "*Flexible Job Shop* con tiempos de alistamiento dependientes de la secuencia".
- Elaborar un artículo de carácter publicable que contenga los resultados del proyecto de investigación.

4. Revisión de la Literatura

Mediante la revisión de literatura se encontró que los enfoques para resolver el problema de programación de operaciones del *Flexible Job Shop* (FJSP) pueden ser clasificados en dos categorías; enfoque jerárquico y enfoque integrado. El enfoque jerárquico se basa en la descomposición del problema para reducir su complejidad. Esta reducción de la complejidad se logra separando el problema en dos subproblemas: el nivel de decisiones de asignación o enrutamiento Flexible y las decisiones de secuenciación de operaciones. En cambio, el enfoque integrado reúne ambos niveles del problema y lo resuelve simultáneamente.

Adicionalmente las metodologías utilizadas para resolver el FJSP se pueden clasificar en: Algoritmos exactos, Heurísticas, Metaheurísticas e Híbridos. Es muy frecuente el uso de las dos últimas metodologías porque mejoran la calidad de las soluciones en tiempos computacionales menores, entre las cuales se pueden destacar: Algoritmos Genético (Genetic Algorithm, GA), Recocido simulado (Simulated Annealing, SA), Búsqueda Tabú (Tabú Search, TS), Optimización por Enjambre de Partículas (Particle Swarm Optimization, PSO).

Esta revisión se divide en tres partes: Revisión de literatura del FJSP, tiempos de alistamiento y revisión de literatura del SDST-FJSP.

4.1 Revisión de literatura del FJSP

El *Flexible Job Shop* ha sido un tema de gran interés en la comunidad científica desde el trabajo de Brucker P. y R. Schlie (1990). De ahí en adelante han preferido resolver el problema por dos enfoques: jerárquico e integrado.

4.1.1 **Enfoque Jerárquico.** Algunos autores mostraron su interés por el enfoque jerárquico Brandimarte (1993) realizó un estudio de un algoritmo jerárquico para el FJSP donde ambos subproblemas fueron abordados mediante Búsqueda Tabú. Posteriormente Zribi, Kacem, El Kamel, y Borne (2007) propusieron un nuevo método jerárquico para el FJSP Multiobjetivo para esto consideraron 3 criterios: Makespan, carga de trabajo de la máquina crítica y carga total de las máquinas. El problema se dividió en dos subproblemas: asignación y secuenciación, para el primero propusieron dos métodos: el primero se basa en un enfoque heurístico y una búsqueda local; el segundo se basa en un algoritmo de Ramificación y Acotamiento (Branch and Bound) y para el segundo aplicaron un Algoritmo Híbrido Genético (AHG). Ellos deducen que un pequeño valor de la probabilidad de mutación garantiza un buen compromiso de exploración y mutación y para valores altos la convergencia es perturbada por la mutación y el algoritmo no puede optimizar las soluciones de manera eficiente.

En el mismo año, Saidi- Mehrabad, Fattahi, y Jolai (2007) desarrollaron 4 algoritmos a partir de las Metaheurísticas Recocido Simulado y Búsqueda Tabú, todos con enfoques jerárquicos. Así mismo Pan, Li, Xie, Jia , y Yu-ting, (2010) usaron Búsqueda Tabú para el subproblema de asignación y Optimización por Enjambre de Partículas (Particle Swarm Optimization, PSO) para el subproblema de secuenciación.

El más reciente trabajo que utilizó este enfoque fue el de Tejada Muñoz (2016) con un GA para la Optimización Multiobjetivo y para los dos subproblemas. En el primero buscaba la solución óptima de máquinas a cada operación buscando minimizar: máxima carga de trabajo entre todas las máquinas (WM) y suma de trabajo de todas las máquinas (WT), en este subproblema usaron un algoritmo genético modificado con pocos individuos. En el segundo

subproblema buscan la mejor secuenciación de las operaciones en cada una de las máquinas mediante un GA convencional.

4.1.2 Enfoque integrado. Por otro lado está el *enfoque integrado*, el primer estudio conocido fue realizado por Brucker P. y R. Schlie (1990) en el cual desarrollaron un algoritmo de tiempo polinomial que se basa en un método gráfico para resolver el FJSP con dos trabajos. Luego, Hurink, Jurisch, y Thole (1994) emplearon una Búsqueda Tabú modificada a la empleada por Brandimarte utilizando una estructura de vecindario más sofisticada. Años más tarde Paulli y Dauzere-Peres (1997) definieron una estructura de vecindario donde no hay una distinción entre re-asignar y re-secuenciar una operación, esta se utiliza en la Búsqueda Tabú para solucionar el FJSP. Adicionalmente formulan una versión de un modelo grafo disyuntivo que tiene en cuenta que las operaciones tienen que ser asignadas a las máquinas.

A partir del año 2000 Mastrolilli y Gambardellan (2000) usaron técnicas de búsqueda local y presentaron dos estructuras de vecindario (Nopt1, Nopt2) para resolver el FJSP aplicándolas a la Búsqueda Tabú haciendo que trabaje más rápido y eficiente que otros algoritmos. Alvarez Valdes, Fuertes, Tamarit, Gimenez, y Ramos (2005) aplicaron una heurística basada en varias reglas de prioridad en una fábrica de vidrios, la cual fabrica más de 3000 tipos diferentes de piezas y trabaja bajo una política de fabricación a pedido.

Recientemente los Algoritmos Genéticos (GAs) han sido implementados con éxito en el FJSP, buscando minimizar diversos objetivos como: el Makespan, tiempo de flujo, carga de trabajo total, carga de trabajo de la máquina crítica, anticipación, tardanza total, tardanza promedio, costo de producción, entre otros.

Kacem, Hammadi, y Borne (2002) presentaron dos nuevos enfoques integrados para resolver el FJSP, el primero es un enfoque de localización (AL) por sus siglas en inglés y el segundo es un enfoque evolutivo controlado por el modelo de asignación, generado por el primer enfoque para resolver el FJSP Multiobjetivo.

Después, Wu y Weng (2005) proponen un método de programación multivalente, el cual consiste en agentes de trabajo (Job Agents, JAs) y agentes de máquina (machine agents, MAs) cuyos objetivos son de anticipación y tardanza del trabajo en un *Flexible Job Shop,* cabe resaltar que el método propuesto es eficiente computacionalmente resolviendo en menos de 1,5 minutos un programa con más de 2000 trabajos en 10 máquinas.

Saidi *et al*. (2007) también utilizaron el enfoque integrado con dos algoritmos; enfoque integrado de heurística de recocido simulado (integrated approach with simulated annealing heuristic, ISA) y enfoque integrado con heurística de búsqueda tabú (integrated approach with tabú search heuristic, ITS). El algoritmo ISA usa dos tipos de movimientos para el problema de asignación y secuenciación. En cambio, en el algoritmo ITS la estructura de vecindario no distingue entre asignar y secuenciar una operación. Con el fin de comparar los algoritmos propuestos desarrollaron el modelo matemático para el FJSP y lo resolvieron por el método de ramificación y acotamiento, mediante el cual propusieron instancias de tamaño medio y pequeño con los límites superiores e inferiores de las soluciones óptimas.

En el siguiente año Pezzella, Morganti, y Ciaschetti (2008) propusieron un Algoritmo Genético (GA) para el FJSP donde aplicaron estrategias para generar la población inicial, la selección y el cruce. Para la población inicial utilizaron permutación aleatoria de trabajos y máquinas, mínimo global para abordar el subproblema de asignación y un mix de tres reglas

de despacho: aleatoria, mayoría de trabajo restante (MostWork Remaining, MWR) y mayor cantidad de operaciones restantes (Most number of Operations Remaining, MOR) para el subproblema de secuenciación. Luego, en la selección se emplearon diferentes métodos: torneo binario, torneo de tamaño n y clasificación lineal. Después, para generar nuevos individuos utilizaron dos tipos de operadores: un operador de cruce de asignación y otro para la secuenciación, operador basado en preservar la precedencia (Precedence Preserving Order-based crossover, POX) el cual determina cuales genes de cada padre debe heredar cada hijo mediante la selección aleatoria de los mismos, respetando las restricciones de precedencia. Por último, para la mutación utilizaron tres operadores: mutación de asignación, mutación inteligente y operador de cambio en preservar la precedencia (Precedence Preserving Shift mutation, PPS). Ellos afirman que el algoritmo propuesto obtuvo mejores resultados a otro GA para el mismo problema por las diferentes estrategias empleadas.

Con el objetivo de mejorar la eficiencia del GA Sun, Pan, Lu, y Ma (2010) proponen un Algoritmo Genético mejorado para resolver el FJSP, adicionalmente agregan el requisito que cada trabajo debe terminar en un periodo de tiempo establecido, es decir, agregan el criterio justo a tiempo (Just In Time, JIT) con el fin de evitar costos de inventarios por terminar antes de lo que se establece y costos por faltantes por terminar después. De manera simultánea emplean un método que transforma el FJSP con flexibilidad parcial y solicitud JIT al FJSP total.

En los últimos años Driss, Mouss, y Laggoun (2015) resuelven el FJSP con instancias de la literatura y prueban el nuevo algoritmo Híbrido Genético (NGA) con datos de una empresa de fabricación de medicamentos denominada Saida Group ubicada en Argelia, África. La programación correspondía a 10 trabajos en 31 máquinas. La fábrica está compuesta de 3

centros: pesajes con 10 máquinas, producción con 8 máquinas y acondicionamiento con 13 máquinas. Los resultados evidenciaron que el NGA obtuvo buenos resultados para el FJSP.

En la aplicación del Recocido Simulado para resolver el FJSP, Najid, Dauzere-Pérés, y Zaidat (2002) expresan que para aplicarlo son necesarios tres componentes principales: una representación concisa del problema, una función de vecindario y un programa de enfriamiento. Posteriormente Yazdani, Gholami, Zandieh, y Mousakhani (2009) propusieron un método para determinar la temperatura inicial, la cual se calcula mediante el promedio de las desviaciones del valor de la función objetivo, además utilizaron dos estructuras de búsqueda de vecindario, para los subproblemas de secuenciación y asignación.

Años más tarde Cruz-Chávez, Martínez-Rangel, y Cruz-Rosales (2017) en su investigación proponen un algoritmo de Recocido Simulado Acelerado por un mecanismo de programación parcial, el cual acelera la búsqueda en el vecindario y el programa de enfriamiento define el valor de la temperatura inicial en función de la desviación estándar.

4.1.3 **Los enfoques híbridos aplicados al FJSP.** Tang, Zhanj, Lin, y Zhang (2011) propusieron un algoritmo híbrido basado en el Algoritmo Genético (GA) y la Búsqueda de Vecindario Variable (VNS) para resolver este problema, buscando minimizar el Makespan. En el mismo año, también aplicaron un método de Optimización caótica para superar las debilidades del Algoritmo Genético (GA) y Optimización por Enjambre de Partículas (PSO), las cuales son: el GA tiene una tasa de convergencia lenta y el PSO es fácil de que genere un óptimo local, además utilizaron una estrategia de ajuste de parámetros autoadaptativos que se integra para mejorar la calidad y eficiencia de la solución.

También los enfoques híbridos se utilizaron para la Optimización Multiobjetivo; minimizar Makespan, minimizar carga de trabajo máxima de la máquina y minimizar carga de trabajo total. Kacem *et al*. (2002b) proponen un enfoque de Pareto basado en la hibridación de la Lógica Difusa (Fuzzy logic, FL) y Algoritmos Evolutivos (Evolutionary Algorithms, EAs) para resolver el FJSP, este enfoque híbrido explota las capacidades de representación del conocimiento de FL y las capacidades de adaptación de EAs.

Gao, Gen, Sun, y Zhao (2007) desarrollaron un nuevo enfoque híbrido que utiliza el Algoritmo Genético con una heurística denominada cuello de botella móvil (CBM). Un año más tarde Gao, Sun, y Gen (2008) proponen un híbrido entre el Algoritmo Genético (GA) y Búsqueda por Entornos Variables Descendente (VND) con el fin de adaptarlos a la estructura del problema FJSP utilizaron operadores avanzados de mutación y cruce. De la misma manera Gen, Gao, y Lin (2009) desarrollaron un Algoritmo Genético basado en múltiples etapas con la heurística cuello de botella móvil para el problema FJSP.

Chang, Chen, Liu, y Chou (2015) emplearon un algoritmo genético combinado con el método Taguchi. El cromosoma comprende dos secciones: cada gen representa una operación de un trabajo (Operation Sequence, OS) y cada gen representa un número de máquina (Machine Selection, MS). La población inicial se genera mediante selección global y selección aleatoria, la selección se realizó mediante el método de ruleta, para el cruce se usaron dos métodos de cruce: de dos puntos y cruce uniforme, luego se usó el método de Taguchi para identificar la combinación óptima de cromosomas para acelerar la convergencia del algoritmo.

4.2 Tiempos de alistamiento

Según Allahverdi (2015) los tiempos de alistamiento juegan un papel importante en los ambientes modernos de manufactura y servicios para la entrega de productos confiables a tiempo. Los tiempos de alistamiento no agregan valor, sin embargo, necesitan ser considerados en las decisiones de programación que se toman con el objetivo de aumentar la productividad, eliminar desperdicio, mejor utilización de los recursos y cumplir los plazos. El interés por los tiempos de alistamiento en problemas de programación de operaciones se ha considerado desde la mitad de los años 60 y se ha ido incrementando hasta la actualidad.

Específicamente para el problema de Job Shop diferentes autores han trabajado con: tiempos de alistamiento dependientes/ independientes de la secuencia por trabajo. Sotskov, Tautenhahn, y Werner (1999) propusieron diversas técnicas de inserción combinadas con Búsqueda de Haces para resolver el Job Shop con tiempos de alistamiento independiente. Ballicu, Giua, y Seatzu (2002) modelan el Job shop con tiempos de alistamiento dependientes de la secuencia con un modelo de programación entera mixta.

Para el *Flexible Job Shop* se ha trabajado con tiempos de alistamiento dependientes de la secuencia, Choi y Choi (2002) resolvieron el problema con tiempo de alistamiento dependiendo de la operación, es decir, que el tiempo de alistamiento está determinado por la relación que hay entre las operaciones de todos los trabajos. Por otro lado, Saidi-Mehrabad y Fattahi (2007) basados en lo expuesto por Choi y Choi (2002) simplificó el problema al considerar los tiempos de alistamiento dependientes solo entre trabajos, es decir, el tiempo de alistamiento solo existe para operaciones de diferentes trabajos. Adicionalmente según lo encontrado en la literatura en algunos trabajos realizan la distinción entre tiempos de alistamiento dependientes de la secuencia: separables e inseparables. Los tiempos de alistamiento separables es donde se

permite cualquier holgura mayor que cero entre el final del alistamiento y el inicio de la operación correspondiente y los tiempos de alistamiento inseparables es cuando una actividad debe iniciarse inmediatamente después de que el alistamiento haya finalizado.

4.3 Revisión de literatura del problema Flexible Job Shop con Tiempos de alistamiento dependientes de la secuencia (SDST-FJSP)

En los últimos años han existido pocos trabajos que se refieran al SDST-FJSP, el cual incluye dos factores adicionales al *Job Shop* Clásico; la flexibilidad y los tiempos de alistamiento dependientes de la secuencia. Para el cual se ha utilizado diferentes métodos de solución tanto en enfoque jerárquico como en enfoque integrado para su solución

4.3.1 Enfoque Jerárquico. Para el enfoque jerárquico el primer investigador en mostrar interés fue Imanipour (2006) modelando el problema con un modelo de programación entera mixta no lineal y propuso un híbrido entre Búsqueda Tabú (TS) y Búsqueda de vecindario codiciosa (Creed Neighborhood Search, GNS) para encontrar la mejor secuencia de operaciones, después utilizó una heurística para programar las operaciones. El planteó 5 problemas con flexibilidad total para evaluar el algoritmo. Según los resultados computacionales no existe diferencia significativa en el tiempo computacional a medida que los rangos de tiempos de alistamiento disminuyen.

Más tarde Saidi-Mehrabad y Fattahi (2007) realizaron un algoritmo de Búsqueda Tabú (TS) para resolver el FJSP, considerando tiempos de alistamiento dependientes de la secuencia inseparables cuyo objetivo era minimizar el Makespan. La primera fase busca la mejor secuencia de operaciones de los trabajos y la segunda busca la mejor opción de máquinas. Según los resultados obtenidos el algoritmo propuesto es capaz de alcanzar una solución cerca

de la solución óptima minimizando el tiempo computacional comparado con el método exacto de ramificación y acotamiento (Branch and Bound).

4.3.2 Enfoque Integrado. Por otro lado se muestra el interés por el enfoque integrado. Los primeros investigadores en mostrar su interés hacia este enfoque fueron Choi y Choi (2002) plantearon un modelo de programación entera mixta para el FJSP con tiempos de alistamiento separables y para solucionarlo propusieron un método de búsqueda local, el cual utiliza una propiedad para reducir el tiempo computacional, este mejora significativamente muchas reglas de despacho codiciosas entre las cuales se encuentran tiempo de inicio más temprano (Earliest Start Time, EST), tiempo de finalización más temprano (Earliest Completion Time, ECT) y la mayoría de las operaciones restantes (Most Operations Remaining, MOPR).

En el mismo año Freitas Guimaraes y Aparecida Fernandes (2006) utilizaron un Algoritmo Genético eficiente para el FJSP que incluye tiempos de alistamiento dependientes de la secuencia separables y Optimización Multiobjetivo, introdujeron nuevos operadores que hacen uso de conocimiento específico del problema para mejorar la calidad de las soluciones. Ellos destacan que el aspecto más importante a considerar en el Job Shop Problem es la "factibilidad" y debido a la naturaleza combinatoria puede dificultar la convergencia del algoritmo si se presentan soluciones infactibles en la población. Para ello utilizaron la representación denominada lista de secuencia de tareas (Tasks Sequencing List, TSL) la cual contiene una lista de todas las operaciones de todos los trabajos del problema. Adicionalmente ellos compararon su procedimiento llamado Balanceando la lista de secuencia de tareas (Balancing Task Sequencing List BTSL) sin considerar los tiempos de alistamiento y se demostró que el método iguala a otros encontrados en la literatura. Sin embargo, manifiestan que no fue posible

comparar los resultados considerando los tiempos de alistamiento dependientes de la secuencia porque no se encontraron trabajos relacionados con esta variante.

Posteriormente Bagheri y Zandieh (2011) utilizaron Algoritmo de Búsqueda de Vecindario Variable (VNS) para minimizar la Función Objetivo Agregada (Aggregate Objective Function, AOF)=αF1 + (1-α) F2 y α es el peso respectivo al Makespan (F1) y Retraso Medio (F2). En este algoritmo utilizan dos estructuras de vecindario; una para el problema de asignación y otra para el problema secuenciación. Para la evaluación del algoritmo se generaron aleatoriamente 20 instancias de 4 clases diferentes.

En ese mismo año los autores Oddi, Rasconi, Cesta, y F.Smith (2011) proponen la variante de Búsqueda Aplanadora Iterativa (Iterative Flattening Search, IFS) un procedimiento de búsqueda basado en restricciones centrales como su solucionador (Sheduling Procedure called Precedence Posting, SP-PCP). Los resultados evidenciaron que la heurística de relajación basada en holgura es más rápida en el proceso de resolución del algoritmo. Adicionalmente ellos proponen unas instancias de Benchmark llamadas SDST-HUdata agregando a la instancia original de HUdata la matriz de tiempos de alistamiento, siendo esta la misma para todas las máquinas.

González, Vela, y Varela (2013) integraron un Algoritmo Genético (GA) con Búsqueda Tabú (TS). Para esto diseñaron una estructura de vecindario $N^{SF}{}_1$ la cual es la unión de dos estructuras: $N^{S}{}_1$, diseñada para modificar la secuenciación de las tareas, y $N^{F}{}_1$, diseñado para modificar la asignación de las máquinas. En la fase de selección, la Búsqueda Tabú es aplicada a los descendientes para luego reemplazar los padres por medio de la selección por torneo.

Mediante el estudio experimental concluyen que el método híbrido es mejor que cada método solo.

Recientemente, Abdelmaguid (2015) creó una función de búsqueda de vecindario codiciosa aleatoria (Randomized Greedy Neighborhood Search, RGNS) para resolver el FJSP con tiempos de alistamiento separables. Esta función investiga todas las máquinas en que se puede realizar una operación selecciona la máquina con el tiempo mínimo de procesamiento. A esta máquina se le determina los límites inferior y superior de las posiciones de inserción. Luego se construye la cola finita de la mejor posición de inserción, dentro de los límites determinados excluyendo la posición actual de la operación si la máquina investigada es la misma que la actual en la solución inicial, si la posición de inserción no está disponible, se investiga otra máquina. Por último utilizó un enfoque de Búsqueda Tabú codiciosa-aleatoria (Randomized-Greedy Tabú Search, RGTS) basado en RGNS.

En los últimos dos años los autores Azzouz, Ennigrou, y Ben Said (2016) desarrollaron un Algoritmo Genético para minimizar la Función Objetivo Agregada (Aggregate Objective Function, AOF). Este algoritmo mostró mejores resultados comparado con Búsqueda de Vecindario Variable (VNS) y Búsqueda Tabú (TS).

En el mismo año Azzouz et al. (2016) desarrollaron un Algoritmo Híbrido basado en un Algoritmo Genético (GA) y Búsqueda de Vecindario Variable (VNS) para minimizar la Función Objetivo Agregada (AOF). Ellos compararon el algoritmo con VNS, AIS, PSO, GA, este muestra mejores resultados para valores de α pequeños.

En el año 2017 Azzouz et al., emplearon un algoritmo híbrido auto-adaptativo (SAHA) que integra el algoritmo genético (GA) y la búsqueda local iterativa (Iterative Local Search, ILS). La estrategia auto-adaptativa de este algoritmo permite actualizar los valores de los parámetros de acuerdo a los requerimientos del espacio de búsqueda actual.

5. Marco Teórico

5.1 Optimización Combinatoria

La Optimización combinatoria es una rama de la Optimización, la cual es una disciplina base en campos como la investigación de operaciones, inteligencia artificial e informática. Un problema de Optimización consiste en "encontrar el valor de unas variables de decisión para los que una determinada función objetivo alcanza su valor máximo o mínimo, el valor de las variables en ocasiones está sujeto a unas restricciones". (Martí, 2017, pág. 2)

Los problemas de Optimización se pueden dividir en dos categorías: aquellos con variables continuas y aquellos con variables discretas. En la segunda categoría se encuentran los problemas de Optimización combinatoria. El concepto de un problema de Optimización combinatoria de Blum y Roli (2003) afirman que**:**

Un problema de Optimización combinatoria P(S, f) está definido por:

- Un conjunto de variables $X = \{x_1,\ldots,x_n\}$;
- Dominio de variables $D_1,\ldots,D_n$;
- Restricciones entre variables
- Función objetivo *f* a minimizar, donde *f*: $D_1 \times \ldots \times D_n \rightarrow IR+;$

El conjunto de todas las posibles asignaciones factibles es:

$$S = \{s = \{(x_1, v_1),\ldots, (x_n, v_n)\} \mid v_i\ i \in Di, \text{ s satisfice todas las restricciones}\}$$

S es el espacio de búsqueda, donde cada elemento del conjunto se puede ver como una solución candidata. Para resolver un problema de optimización combinatoria, se debe encontrar una solución $s^* \in S$ que dé el valor mínimo de la función objetivo, es decir, $f(s^*) \leq f(s)\ \forall\ s \in S$. s^* se considera la solución óptima global de (S, f) y el conjunto t $S^* \subseteq S$ es considerado el conjunto de soluciones óptimas globales.

Algunos ejemplos de problemas de Optimización clásicos son: a) problema de la mochila; b) problema del agente viajero (TSP); c) problema de asignación cuadrática (QAP); d) programación entera.

5.1.1 **Tipos de complejidad computacional.** La complejidad computacional estudia la cantidad de recursos computacionales necesarios para resolver un problema; tiempo (número de espacios de ejecución de un algoritmo para resolver un problema) y espacio (cantidad de espacio de memoria para resolver un problema), esto se calcula mediante la cantidad de operaciones que genere el algoritmo para resolver el problema. La máquina de Turing es utilizada como punto de referencia para realizar la siguiente clasificación según Hamalainen (2006) como se muestra en la figura 1.

Clase P: Cuando el tiempo de ejecución de un algoritmo es menor que un cierto valor calculado a partir del largo de la variable de entrada (N) usando una fórmula polinómica.

Clase NP: Es el conjunto de problemas de decisión que pueden ser resueltos por una máquina no determinista en un tiempo polinómico. Los problemas de la clase P son un subconjunto de los de la clase NP.

Clase NP completos: Si el problema es NP y todo otro problema NP es reducible a este, es decir, basta resolver el problema en cuestión para resolver cualquier otro.

Clase NP Duro: Son problemas muy difíciles de resolver que requieren en su mayoría tiempo exponencial para ser solucionado.

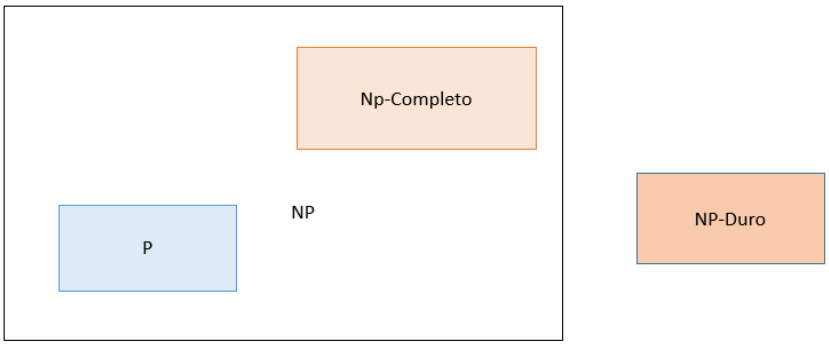

Figura 1. Tipos de complejidad Computacional. Adaptado de Duarte, A., Pantrigo, J. J., Gallego, M. (2008). Introducción a la Optimización. En S. Meléndez (Eds.), Metaheurísticas (pp.11). *Madrid. España: Dykinson*

Según Brandimarte el problema *Job Shop*, desde el punto de vista de la complejidad computacional pertenece a la clase NP Hard. El Flexible Job-shop (FJSP) es una generalización del *Job Shop*, donde se permite procesar operaciones entre cualquiera de las máquinas disponibles. Siendo así, el FJSP más difícil que el JSP clásico, dado que se introduce un nivel adicional de decisión al de la secuencia, es decir, las rutas de trabajo.

5.2 Problema de programación.

Según (Pinedo, 2010, pág. 30) un problema de programación *determinístico* se describe mediante un triplete $\alpha \mid \beta \mid \gamma$. El campo α describe el entorno de la máquina. El campo β proporciona detalles de las características y restricciones de procesamiento, el cual puede tener múltiples entradas. El campo γ describe el objetivo, este debe ser minimizado y puede contener varias entradas. Por lo tanto en un problema de planificación siempre existirán tres componentes distintos.

5.2.1 **Entornos de las Máquinas (α).** Los problemas de planificación y secuenciación pueden dividirse como lo muestra la figura 2. Los posibles entornos especificados en el campo α son de máquinas no especializadas y máquinas especializadas.

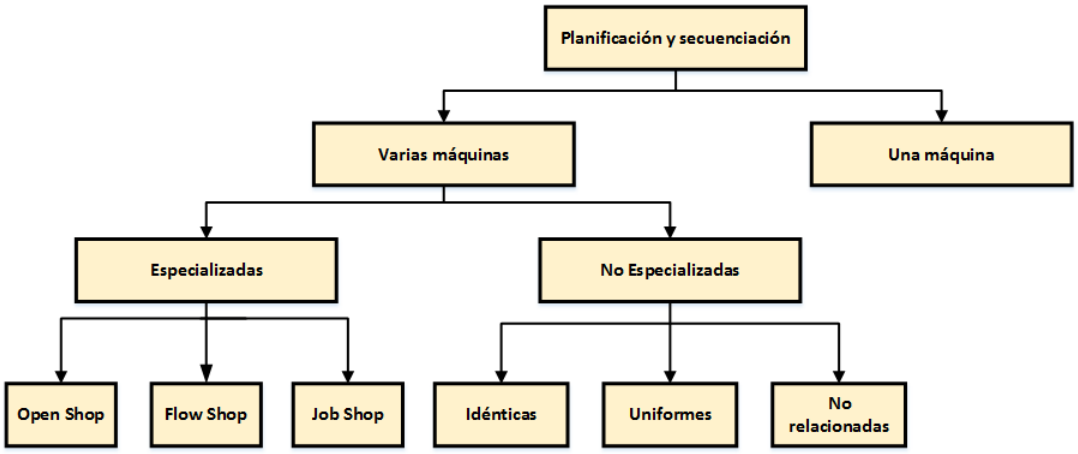

Figura 2. Clasificación de los problemas de planificación y secuenciación.

5.2.1.1 Máquinas no especializadas. Estas máquinas se caracterizan porque las máquinas realizan las mismas funciones.

Una sola máquina (1): El caso de una sola máquina es el más simple de todos los entornos.

Máquinas idénticas en paralelo (Pm): Hay *m* máquinas idénticas en paralelo. El trabajo *j* requiere una sola operación que puede ser procesada en cualquiera de las *m* máquinas y los tiempos de procesamiento son iguales.

Máquinas en paralelo con diferentes velocidades (Qm): Hay *m* máquinas en paralelo con diferentes velocidades. La velocidad de la máquina *i* está denotada por *vi*. El tiempo p_{ij} que el trabajo *j* pasa en la máquina *i* es igual a p_j / v_i.

Máquinas no relacionadas en paralelo (Rm): Este entorno es una generalización más de la anterior. Hay *m* diferentes máquinas en paralelo. Máquina *i* puede procesar el trabajo *j* a la velocidad v_{ij}. El tiempo que el trabajo *j* gasta en la máquina *i* es igual a p_j / v_{ij}.

5.2.1.2 Máquinas especializadas. Estas máquinas se relacionan en problemas en que los trabajos se dividen en operaciones, no todas las máquinas son capaces de realizar todas las operaciones**.**

Flow shop (Fm): Hay *m* máquinas en serie. Cada trabajo tiene que ser procesado en cada una de las *m* máquinas. Todos los trabajos tienen que seguir la misma ruta.

Jop Shop (Jm): En un taller de trabajo con *m* máquinas cada trabajo tiene su propia ruta predeterminada a seguir.

Flexible Job Shop (Féc.): Es una generalización del *Job Shop* y los entornos de máquinas paralelas. Este tiene como característica adicional que las operaciones pueden ser procesadas en una o más máquinas de un conjunto presentes en el taller.

Open Shop (Om): Se debe procesar un conjunto de trabajos en *m* máquinas. Cada trabajo consiste en *m* operaciones, cada una de las cuales debe procesarse en una máquina diferente para un tiempo de procesamiento determinado. Las operaciones de cada trabajo se pueden procesar en cualquier orden.

5.2.2 **Características y restricciones de procesamiento (β).** Especificadas en el campo β pueden incluir varias entradas. Las entradas posibles en el campo β son:

Restricciones de precedencia (prec): Las restricciones de precedencia se refiere a que se debe completar uno o más trabajos antes de que se permita a otro trabajo iniciar su procesamiento.

Tiempos de alistamiento dependientes de la secuencia (S_{ijk}): El S_{ijk} representa el tiempo de alistamiento dependiente de la secuencia que se produce entre el procesamiento de los trabajos j y k en la máquina i S_{ijk}.

Las restricciones de elegibilidad de máquinas (M_j): El símbolo M_j puede aparecer en el campo β cuando el entorno de la máquina hay *m* máquinas en paralelo (Pm). Cuando el M_j está presente, no todas las *m* máquinas son capaces de procesar el trabajo j. Conjunto M_j denota el conjunto de máquinas que pueden procesar el trabajo j.

5.2.3 Función objetivo (γ). A continuación se muestran los ejemplos de las posibles funciones objetivo a ser minimizada.

Makespan (C_{max}): El Makespan, definido como máx. (C_1,..., C_n) es equivalente al tiempo de finalización del último trabajo para salir del sistema.

Retraso Máximo (Lmax): El retraso máximo se define como máximo (L1,..., L_n). Mide la peor violación de las fechas de vencimiento.

Tiempo de finalización total ponderado ($\Sigma w_j C_j$): En la literatura también se conoce como el tiempo de flujo. Este proporciona una indicación de los costos totales de tenencia o de inventario incurridos por el programa.

Retraso ponderado total ($\Sigma w_j T_j$). Esta es también una función de costo más general que el tiempo de finalización total ponderado.

Entre otras medidas de desempeño utilizadas en el FJSP se pueden encontrar; tiempo medio de finalización, tiempo de flujo máximo (fmax), retraso total, retraso promedio, número de trabajos retrasados, carga de trabajo total de las máquinas, carga de trabajo de la máquina crítica.

Basados en la notación anterior el problema a tratar en este proyecto se define de la siguiente manera.

Tabla 2.

Notación del problema en investigación

Notación	**α**	**β**	**γ**
	Féc	*Sjk ó Sijk.*	C_{max}
Descripción	Flexible Job Shop	Tiempos de alistamiento dependientes de la secuencia	Makespan

5.2.4 **Clases de programas.** Un programa o "schedule" se refiere a una asignación de trabajos dentro de un conjunto de máquinas. Dependiendo de las características, un Schedule según (Pinedo, 2010, pág. 25) se puede clasificar en tres tipos; *programa no retraso, programa activo y programa semi-activo* como se puede ver en la figura 3.

Programa no retraso: Es cuando ninguna máquina se mantiene inactiva mientras una operación está esperando para procesar. Requerir un programa para no demora, equivale a prohibir la ociosidad no forzada.

Programa activo: Se considera activo cuando no es posible construir otra programación, a través de cambios en el orden de procesamiento en las máquinas, con al menos una operación terminando antes y ninguna operación terminando más tarde.

Programa semi-activo: Ninguna operación puede comenzar antes sin alterar la secuencia de operación para una asignación de máquina dada. Es decir, las operaciones se programan exactamente en el mismo orden en que aparecen en la secuencia de cromosomas.

Figura 3. Diagrama de Venn de clases de programas no preventivos para Job Shop. Adaptado de Pinedo, M. (2012). En M. L. Pinedo, Scheduling: Theory, algorithms y systems. *(pág. 25). Londres: Springer. doi:10.1007/978-1-4614-2361-4.*

5.3 Flexible Job Shop Scheduling (FJSP)

Según F.T.S. y Wong (2006) hay dos tipos de *Flexible Job Shop.* En el primero, un trabajo puede tener secuencias de operación alternativas y múltiples máquinas capaces de cada operación. En el segundo, los trabajos tienen una secuencia de operación fija, pero todavía hay máquinas alternativas para cada operación. Este trabajo trata el segundo tipo de problema.

El FJSP consiste en dos subproblemas: 1) asignación de máquinas: seleccionar una máquina de un conjunto de máquinas candidatas para las operaciones y 2) la secuenciación: la programación de las operaciones en todas las máquinas, es decir la secuencia con la que se debe ejecutar las operaciones de los trabajos en cada una de las máquinas asignadas.

5.3.1 **Flexibilidad Total y Flexibilidad Parcial**. El FJSP se clasifica en FJSP total y FJSP parcial. Cuando el FJSP es total quiere decir que todas las operaciones se pueden procesar en todas las máquinas. En cambio, el FJSP parcial consiste en que al menos una de las operaciones no se puede procesar en todas las máquinas**.**

A continuación se ilustran ejemplos de FJSP total y FJSP parcial. En la figura 4 se muestra la asignación de dos trabajos a 3 máquinas polivalentes

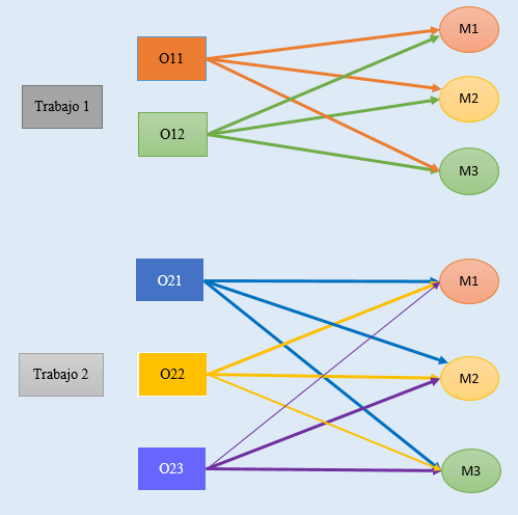

Figura 4. Ejemplo de Flexibilidad total

Como se puede ver en la figura 5 las funcionalidades de las 3 máquinas no son idénticas, esto conlleva a la flexibilidad parcial.

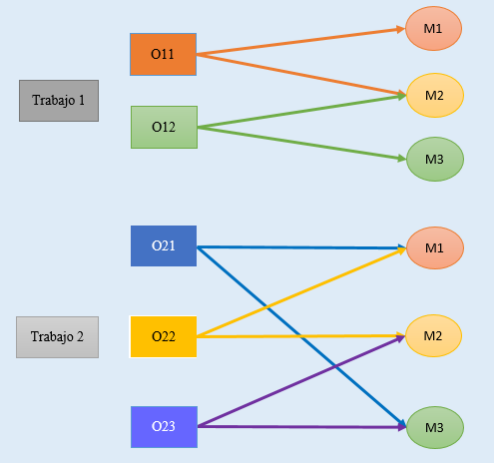

Figura 5. Ejemplo de Flexibilidad Parcial

5.3.2 Consideraciones.

Según el resultado de la revisión de literatura de los autores Imran y Abid (2016), las suposiciones que se consideran en un FJSP general son**:**

- Todas las máquinas están disponibles en el tiempo t = 0.
- Todos los trabajos están disponibles en el momento t = 0.
- Cada operación puede ser procesada por una sola máquina a la vez.

- No existen restricciones de precedencia entre las operaciones de los diferentes trabajos; por lo tanto los trabajos son independientes el uno del otro.
- No se permite la interrupción de operaciones, es decir, una operación una vez iniciada no puede ser interrumpida.
- Tiempo de transporte de trabajos entre las máquinas y tiempo de alistamiento de la máquina para procesar una operación particular se incluyen en el tiempo de procesamiento (p.2).

5.3.3 Representación de la Solución

5.3.3.1 ***Diagrama de Gantt.*** Desde un punto de vista general, un gráfico de Gantt realiza un gráfico de barras horizontal para la visualización basada en el tiempo. Para la programación del taller, la siguiente configuración es típica: La abscisa representa la línea de tiempo mientras que la ordenada distingue las máquinas disponibles como se muestra en la figura 6.

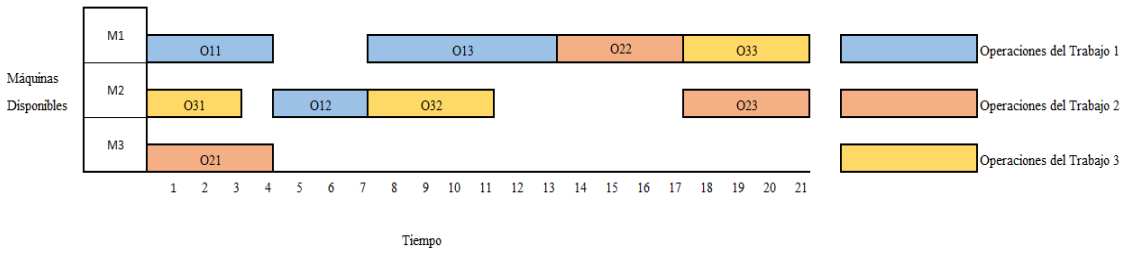

Figura 6. Diagrama de Gantt

5.3.3.2 Grafo Disyuntivo. El modelo de grafo disyuntivo propuesto por Vela et al., (2010) para representar la solución del FJSP consiste en un orden de procesamiento de operación factible σ se representa mediante un grafo acíclico dirigido Gσ = (V, A ∪ E ∪ I). Cada nodo en el conjunto V representa una operación del problema, con la excepción de los nodos ficticios de inicio y fin, que representan operaciones con tiempo de procesamiento 0. Los arcos del conjunto A se llaman arcos conjuntivos y representan restricciones de precedencia y los arcos

del conjunto E se llaman arcos disyuntivos y representan restricciones de capacidad. El conjunto E está dividido en subconjuntos Ei con E = i=1,..., M Ei. Ei corresponde al recurso Ri e incluye un arco (v, w) para cada par de operaciones v, w que requieren ese recurso. Cada arco (v, w) de A se pondera con el tiempo de procesamiento de la operación en el nodo fuente, pv, y cada arco (v, w) de E se pondera con pv + Svw. El conjunto I incluye arcos de la forma (inicio, v) y (v, final) para cada operación v del problema. Estos arcos se ponderan con S0v y pv + Sv0 respectivamente. El Makespan de la programación es el costo de una ruta crítica en Gσ, es decir, un camino dirigido desde nodo de inicio al nodo final que tiene el máximo costo. La figura 7 muestra un ejemplo del grafo disyuntivo.

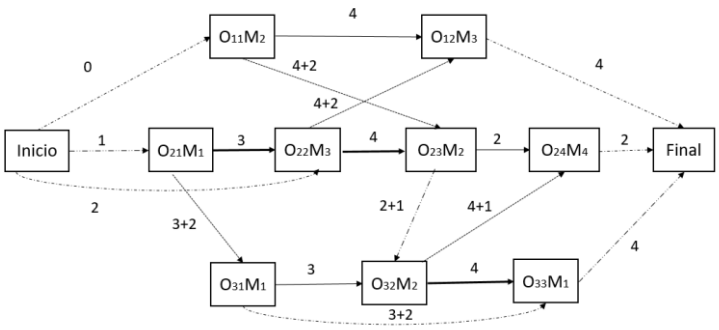

Figura 7. Un programa factible para un problema con 3 trabajos y 3 máquinas. Adaptado de González, M; Vela,C; Varela,R. An efficient Memetic Algorithm for the Flexible Job Shop with Setup Times. En: Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling. p. 91-99.

5.3.4 Tiempos de alistamiento. El tiempo de alistamiento es aquel que se requiere para preparar las máquina, lo cual, permita ejecutar las operaciones. Según la revisión de literatura de Sharma y Jain (2015) en entornos Job Shop los tiempos de alistamiento se clasifican en dos categorías amplias: tiempos de alistamiento no por lotes (trabajo) y tiempos de alistamiento por lotes.

Las dos categorías están subcategorizadas en tiempos de alistamiento dependiente de la secuencia y tiempos de alistamiento independientes de la secuencia, esto se puede evidenciar en la figura 8.

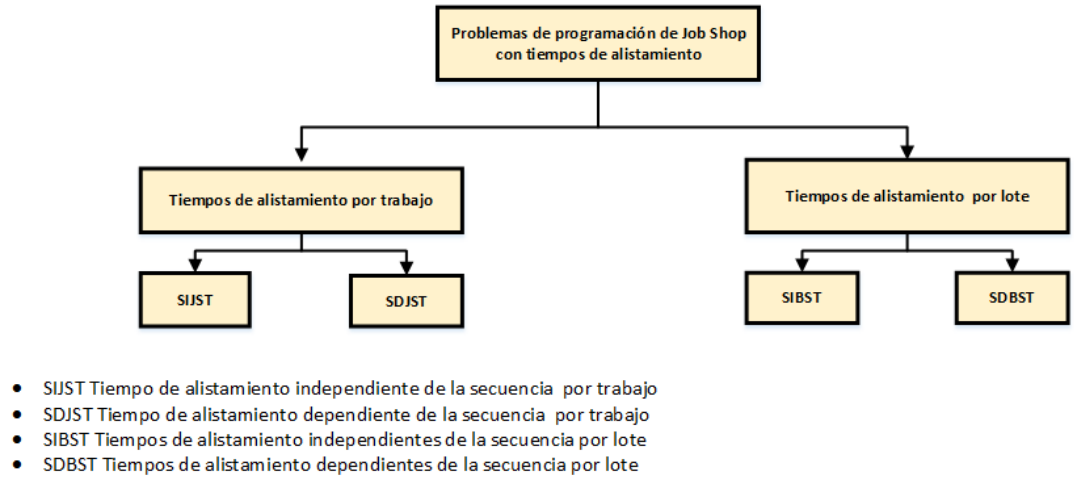

Figura 8. Tiempos de alistamiento en el Job Shop. Adaptado de Sharma, P., y Jain, A. (2015). A review on job shop scheduling with setup times. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 230(3), 517-533. doi:10.1177/0954405414560617

Los tiempos de alistamiento por trabajo /lote. El de trabajo es cuando se cambia de un trabajo a otro en una máquina determinada. Lotes es un conjunto de trabajos que tienen características similares en tiempos de alistamiento, herramientas y secuencia de operación. El número de lotes es fijo así como el número de trabajos en cada lote. Un alistamiento es necesario cuando un trabajo de diferente lote es procesado en una máquina. Es decir, hay un tiempo de alistamiento insignificante para cambiar de un trabajo a otro del mismo lote, pero si hay un tiempo de alistamiento importante entre trabajos de diferentes lotes.

El tiempo de alistamiento independiente de la secuencia es aquel que sólo depende de la operación del trabajo a ser procesada. Este se ha supuesto en numerosos trabajos del Job Shop, esta suposición implica que la gestión de la capacidad solo requiere la asignación de capacidad durante un intervalo de tiempo especificado, debido a que la suma de los tiempos de

alistamiento y procesamiento permanece constante, por lo tanto, la utilización permanece constante.

Por otro lado, el tiempo de alistamiento dependiente de la secuencia es aquel que depende de la operación del trabajo y de la operación inmediatamente precedente .Este último tiene una mirada más realista y completa de la gestión de la capacidad que relaciona la asignación de capacidad y utilización como lo sugieren ejemplos de sistemas industriales.

Los tiempos de alistamiento dependientes de la secuencia (SDST) se pueden utilizar para modelar los tiempos de cambio de una máquina cuando se cambia la producción a otro producto diferente, por ejemplo, en la industria de impresión donde el tiempo de alistamiento (limpieza) de las prensas de impresión dependen del tamaño del papel y del color del uso de la tinta por el trabajo anterior.

A su vez, los tiempos de alistamiento se clasifican en separables e inseparables, basados en los ejemplos mostrados por (Josefowska y Jan weglarz, 2006, pág. 136), se dará una breve explicación.

El tiempo de alistamiento separable, es donde se permite cualquier holgura mayor que cero entre el final del alistamiento y el inicio de la operación correspondiente. Por ejemplo, cuando se necesita trasladar el recurso solicitado de un sitio a otro, por lo tanto, es necesario una holgura para la actividad de transporte.

El tiempo de alistamiento no-separable, es cuando una actividad debe iniciarse inmediatamente después de que el alistamiento haya finalizado. Por ejemplo, el proceso de

alistamiento de la hormigonera para la preparación de hormigón, consiste en verter arena, la grava, y el agua en el mezclador de concreto en cualquier periodo de tiempo, pero el cemento y algunos otros ingredientes específicos tienen que ser vertidos justo antes de iniciarla.

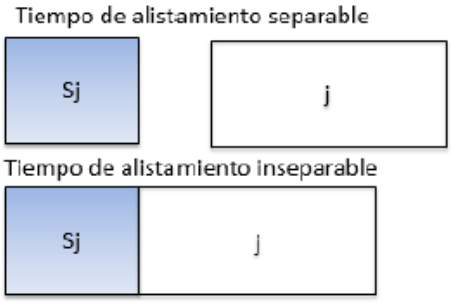

Figura 9. Tiempo de alistamiento separable -inseparable

Por tal motivo en el presente trabajo se ha decidido trabajar con tiempos de alistamiento dependientes de la secuencia y no-separable.

5.3.4.1 Propiedad de desigualdad triangular de los tiempos de alistamiento. De acuerdo a la revisión de literatura los tiempos de alistamiento dependientes de la secuencia tienen la propiedad de desigualdad triangular. Los tiempos de alistamiento dependen de la secuencia, es decir, el tiempo necesario para cambiar el recurso depende de la actividad anterior y la siguiente. Si s^k_{ij} denota el tiempo de alistamiento entre las actividades i, j para el recurso k, Ci es el tiempo de inicio de la actividad i, Sj tiempo de inicio para la actividad j, se debe satisfacer $Ci + s^k_{ij} \leq S_j$ cuando la actividad j se procesa directamente después de la actividad i en el recurso k.

De acuerdo a lo expuesto por Brucker y Knust (2012) a menudo se supone que los tiempos de alistamiento satisfacen la fuerte desigualdad triangular.

$S^k ih + S^k hj \geq S^k ij$ para todo i, j, h $\in$ V, k=1,…, r

O al menos la débil desigualdad triangular

$S^k ih + S^k hj \geq S^k ij$ para todo i, j, h ∈ V, k=1,…, r

5.3.4.2 ***No Simetría.*** Alistar una máquina para procesar un producto de tipo j después de procesar un producto de tipo i puede tomar una cantidad diferente que alistar la máquina para la transición opuesta.

$S^k_{i,j} \neq S^k_{j,i}$

5.4 Modelos Matemáticos para el Flexible Job Shop con Tiempos de Alistamiento Dependientes de la secuencia (SDST-FJSP)

Choi y Choi en el 2002 propusieron un modelo de programación entera mixta donde asumen que los tiempos de alistamiento dependen de las operaciones de cada uno de los trabajos $s_{i,k,l,j,h}$, es decir, que cuando se va a pasar de la operación l del trabajo j a la operación h del trabajo k en la máquina i se tiene un tiempo de alistamiento. Además asumen que los tiempos de alistamiento son separables, esto quiere decir que $O_{i,j,h}$ puede llevarse a cabo incluso antes de que se complete $O_{j,h-1}$ mientras que la máquina i esté inactiva.

Al problema en la formulación matemática lo refieren como Generalizado Flexible Job Shop Scheduling (GFJSSP). En esta formulación un trabajo dummy 0 es usado para indicar el inicio de un trabajo en cada máquina. Además, todo lo relacionado con los tiempos de alistamiento y procesamiento con el trabajo ficticio se establecen en cero.

A continuación se presenta el modelo matemático propuesto por Choi y Choi. El problema tiene m máquinas y n trabajos. Cada trabajo consiste de una secuencia de operaciones $O_{j,h}$, h=1,…,h_j, donde $O_{j,h}$ y h_j denota la enésima operación del trabajo j y el número de operaciones

requeridas para el trabajo j, respectivamente. A menos que se diga lo contrario, el índice i denota la máquina, índice j y k denota trabajos, y h y l denotan los índices de las operaciones.

Dejar $O_{i,j,h}$ denota la operación $O_{j,h}$ realizada en una máquina especifica i. Suponiendo que $O_{i,j,h}$ es programada para seguir inmediatamente después de $O_{i,k,l}$. Luego, $O_{i,j,h}$ requiere un tiempo para un alistamiento, $s_{i,k,l,j,h}$ en adición al tiempo de procesamiento, $p_{i,j,h}$. Adicionalmente se asume que el tiempo de alistamiento es separable. Esto es, $O_{i,j,h}$ se puede llevar a cabo incluso antes $O_{j,h-1}$ se completa mientras la máquina i ($\in M_{j,h}$) está vacía.

La operación $O_{j,h}$ puede ser ejecutada en una máquina que pertenece a un grupo de máquinas alternativas $M_{j,h} = \{m_{j,h,a}\ ; a=1,2,\ldots, |M_{j,h}|\}$, donde $|M_{j,h}|$ y $m_{j,h,a}$ denota la cardinalidad de $M_{j,h}$ y el índice de la máquina alternativa *a* que puede ser usada para $O_{j,h}$ respectivamente. Bajo estas consideraciones y notaciones, el problema es encontrar un programa que minimice el Makespan dado n,m, $O_{j,h}$, $M_{j,h}$, $s_{i,k,l,j,h}$ y $p_{i,j,h}$. Las siguientes notaciones adicionales son usadas en la formulación entera mixta de GJSP.

B_i: un conjunto de índices de operaciones que se pueden procesar en la máquina i, {(j,h); $O_{i,j,h}$ tal que $i \in M_{j,h}$}.

C_{max}: Makespan de un programa.

M: un número muy grande.

$$Y_{i,j,h} = \begin{cases} 1 & \text{Si } O_{j,h} \text{ es realizada en la máquina i} \\ & 0 \text{ de lo contrario} \end{cases}$$

$$X_{i,j,h,k,l} = \begin{cases} 1 & \text{Si } O_{i,j,h} \text{ precede } O_{i,k,l} \text{ inmediatamente} \\ & 0 \text{ de lo contrario} \end{cases}$$

$t_{j,h}$: tiempo de inicio de procesamiento de la operación $O_{j,h}$

Min $C_{máx}$

$t_{j,h} + Y_{i,j,h} * p_{i,j,h} \leq t_{j,h+1}$ para $i \in M_{j,h}$; $h=1,\ldots,h_j-1$; $j=1,\ldots,n$ (1)

$t_{j,hj} + Y_{i,j,h} * p_{i,j,h} \leq C_{max}$ para $i \in M_{j,hj}$; $j=1,\ldots,n$ (2)

$t_{j,h} + p_{i,j,h} + s_{i,k,l,j,h} \leq t_{k,l} + (1- X_{i,j,h,k,l})M$ para $(j,h) \in B_i, (k,l) \in B_i k \geq 1$; $i=1,\ldots,m$ (3)

$\sum_{i \in M_{j,h}} Y_{i,j,h} = 1$ para $h=1,2,\ldots,h_j$; $j=0,1,\ldots,n$; (4)

$\sum_{(j,h) \in B_i} X_{i,j,h,k,l} = Y_{i,k,l}$ para $i \in M_{k,l}$; $l=1,2,\ldots h_k$; $k=0,1,\ldots,n$ (5)

$\sum_{(k,l) \in B_i} X_{i,j,h,k,l} = Y_{i,j,h}$ para $i \in M_{j,h}$; $h=1,2,\ldots,h_j$; $j=0,1,\ldots,n$ (6)

$t_{j,h} \geq 0$ para $h=1,2,\ldots,h_j$; $j=0,1,\ldots,n$; (7)

$Y_{i,j,h} \in \{0,1\}$ para $i \in M_{j,h}$; $h=1,2,\ldots,h_j$; $j=0,1,\ldots,n$ (8)

$X_{i,k,l,j,h} \in \{0,1\}$ para $(j,h) \in B_i, (k,l) \in B_i$; $i=1,2,...,m$ (9)

Las restricciones (1) y (2) hace cumplir que cada trabajo siga una secuencia especificada la Restricción (2) determina el tiempo de finalización de la última operación, es decir, el Makespan (C_{max}). La restricción (3) que cada máquina debe procesar una operación al tiempo y la restricción (4) obliga que una sola máquina debe ser seleccionada de $M_{j,h}$ .Las restricciones (5) y (6) definen permutaciones circulares de operaciones en cada máquina. Elimina operaciones alternativas que se excluyen en un programa final. La restricción (5) selecciona una operación $O_{i,j,h}$ que precede inmediatamente a una operación alternativa programada $O_{i,k,l}$ y la restricción (6) selecciona una operación $O_{i,k,l}$ que sigue inmediatamente a una operación alternativa programada $O_{i,j,h}$. Una permutación circular de operaciones en una máquina produce una secuencia programada de las operaciones en la misma máquina. Las restricciones (3), (5) y (6) prohíbe los subtours, al no permitir que pase más de una vez una operación a una máquina.

Años más tarde Saidi-Mehrabad y Fattahi (2007) realizaron dos modificaciones al modelo matemático propuesto por Choi y Choi. La primera, que el tiempo de alistamiento sólo depende

del trabajo, es decir, $S_{i,k,j}$, este tiempo es necesario cuando se cambia del trabajo k al trabajo j en la máquina i, es decir, asumen que $S_{i,k,l,j,h} = S_{i,k,j}$. La segunda que los tiempos de alistamiento son inseparables.

El problema tiene m máquinas y n trabajos. Cada trabajo consiste de una secuencia de operaciones $O_{j,h}$, h=1,…,h_j, donde $O_{j,h}$ y h_j denota la enésima operación del trabajo j y el número de operaciones requeridas para el trabajo j, respectivamente. A menos que se diga lo contrario, el índice i denota la máquina, índice j y k denota trabajos, y h y l denotan los índices de las operaciones.

Dejar $O_{i,j,h}$ denota la operación $O_{j,h}$ realizada en una máquina especifica i. Suponiendo que $O_{i,j,h}$ es programada para seguir inmediatamente después de $O_{i,k,l}$. Luego, $O_{i,j,h}$ requiere un tiempo para un alistamiento, $s_{i,k,l,j,h}$ en adición al tiempo de procesamiento, $p_{i,j,h}$. En este modelo se asume que $S_{i,k,l,j,h} = S_{i,k,j}$.

$a_{i,j,h}$ Describe el conjunto de máquinas capaces M_{jh} es asignado a la operación O_{jh}

$$a_{i,j,h} = \begin{cases} 1 \text{ si } O_{j,h} \text{ puede ser realizada en la máquina i} \\ 0 \text{ de lo contrario} \end{cases}$$

Bajo estas consideraciones y notaciones, el problema es encontrar un programa que minimice el Makespan dado n,m, $O_{j,h}$, $M_{j,h}$, $s_{i,k,l,j,h}$ y $p_{i,j,h}$. Las siguientes notaciones adicionales son usadas en la formulación entera mixta de GJSP.

C_{max}: Makespan de un programa.

M: un número muy grande.

$$Y_{i,j,h} = \begin{cases} 1 \text{ Si } O_{j,h} \text{ es realizada en la máquina i} \\ 0 \text{ de lo contrario} \end{cases}$$

$$X_{i,j,h,k,l} = \begin{cases} 1 & \text{Si } O_{i,j,h} \text{ precede } O_{i,k,l} \text{ inmediatamente} \\ 0 & \text{de lo contrario} \end{cases}$$

$t_{j,h}$: tiempo de inicio de procesamiento de la operación $O_{j,h}$

$f_{j,h}$: tiempo de finalización de procesamiento de la operación $O_{j,h}$

Min $C_{máx}$

$t_{j,h} + Y_{i,j,h} * p_{i,j,h} \leq f_{j,h}$ para i =1,…,m; j=1,…,n; h=1,2,…,h_j (1)

$f_{j,h} \leq t_{j,h+1}$ para j=1,…,n; h=1,…,h_{j-1} (2)

$f_{j,hj} \leq C_{max}$ para j=1,…,n (3)

$Y_{i,j,h} \leq a_{i,j,h}$ para i =1,…,m; j=0,…,n; h=1,2,…,h_j (4)

$t_{j,h} + p_{i,j,h} + s_{i,j,k} \leq t_{k,l} + (1- X_{i,j,h,k,l})M$ para j=0,…,n; k=1,…,n; h=1,2,…,h_j; l=1,2,…,h_k; i =1,…, m (5)

$f_{j,h} + s_{i,j,k} \leq t_{j,h+1} + (1- X_{i,k,l,j,h+1})M$ para j=1,…,n; k=0,…,n; h=1,2,…,h_{j-1}; l=1,2,…,h_k; i =1,…, m (6)

$\sum_i Y_{i,j,h} = 1$ para h=1,2,…,h_j; j=0,1,…,n; (7)

$\sum_j \sum_h X_{i,j,h,k,l} = Y_{i,k,l}$ para i=1,…,m; k=1,…,n; l=1,2,…,h_k (8)

$\sum_j \sum_h X_{i,j,h,k,l} = Y_{i,j,h}$ para i=1,…,m; j=0,1,…,n; h=1,2,…,h_j (9)

$X_{i,j,h,j,h} = 0$ para i=1,…,m; j=0,1,…,n; h=1,2,…,h_j (10)

$t_{j,h} \geq 0$ para j=0,1,…,n; h=1,2,…,h_j (11)

$f_{j,h} \geq 0$ para j=0,1,…,n; h=1,2,…,h_j (12)

$Y_{i,j,h} \in \{0,1\}$ para i=1,…,m ; j=0,1,…,n; h=1,2,…,h_j (13)

$X_{i,j,h,k,l} \in \{0,1\}$ para i=1,…,m ; j=0,1,…,n; k=1,…,n; h=1,2,…,h_j; l = 1,…,h_k (14)

Las restricciones (1) y (2) hacen referencia a la restricción de precedencia entre dos operaciones del mismo trabajo. La Restricción (3) determina el tiempo de finalización de la última operación, es decir, el Makespan ($C_{max)}$.

La restricción (4) exige que la máquina para cada $O_{j,h}$ debe ser seleccionada de las máquinas alternativas para $O_{j,h}$. Las restricciones (5) y (6) hace cumplir que cada máquina debe procesar una operación al tiempo y considerar los tiempos de alistamiento. La restricción (7) obliga que una sola máquina debe ser seleccionada para $O_{j,h}$.

Las restricciones (8) y (9) definen permutaciones circulares de operaciones en cada máquina. Elimina operaciones alternativas que se excluyen en un programa final. La restricción (8) selecciona una operación $O_{i,j,h}$ que precede inmediatamente a una operación alternativa programada $O_{i,k,l}$ y la restricción (9) selecciona una operación $O_{i,k,l}$ que sigue inmediatamente a una operación alternativa programada $O_{i,j,h}$. Una permutación circular de operaciones en una máquina produce una secuencia programada de las operaciones en la misma máquina.

5.5 Métodos de Solución del SDST-FJSP

Mediante la revisión de literatura se evidenció que los métodos empleados para resolver el FJSP-SDST se dividen en tres grupos: Métodos exactos, Métodos Heurísticos y Meta-heurísticas.

5.5.1 Métodos Exactos. Los algoritmos exactos garantizan que para cada instancia de tamaño finito de un problema de Optimización combinatoria existe una solución óptima en tiempo limitado. Sin embargo, para resolver problemas de programación de taller debido a su

naturaleza Np-Hard no existen algoritmos para resolverlos en tiempo polinomial (Fatos y Ajith, 2008, pág. 11).

5.5.1.1 ***Método de ramificación y acotamiento***. Este método consiste en dos procedimientos: ramificación y acotamiento. La ramificación divide un problema grande en dos o más subproblemas y mutuamente excluyentes. El acotamiento se centra en la relajación lineal del programa entero. Saidi-Mehrabad et al. (2007) utilizan este método para resolver instancias de: tamaño pequeño (SSP1, 15) de 3 trabajos y 3 máquinas, tamaño mediano (MSP1, 5) de 4 trabajos y 5 máquinas y tamaño grande (LSP1, 5) 15 trabajos y 6 máquinas. Las instancias pequeñas son resueltas por este método obteniendo las soluciones óptimas. Para las instancias de tamaño mediano, el tiempo de ejecución fue alto superando los 3600 segundos, por lo que se obtuvo la mejor solución encontrada. Por último para las instancias de tamaño grande el tiempo de ejecución fue muy alto, por lo que no se pudo obtener la solución de los problemas.

5.5.2 **Métodos Heurísticos.** En los métodos de aproximación, se sacrifica la garantía de encontrar soluciones óptimas para obtener soluciones casi óptimas en tiempos computacionales razonables y prácticos. De acuerdo con Blum y Roli. (2003) las heurísticas se pueden clasificar en: métodos constructivos y métodos de búsqueda local. Los métodos constructivos generan soluciones desde cero mediante la adición de componentes de una solución parcial inicialmente vacía hasta que la solución esté completa, un ejemplo de ellos son las reglas de secuenciación**.**

Por otro lado, los métodos de búsqueda local parten de alguna solución inicial e intentan sustituir iterativamente la solución actual por una solución mejor en un vecindario adecuadamente definido de la solución actual. Las reglas de secuenciación utilizan diferentes tipos de información. Existen reglas de secuenciación ordinarias y reglas de secuenciación orientadas al alistamiento.

Tabla 3.

Reglas de secuenciación ordinarias

Regla de secuenciación	**Descripción**	**Autores**
Tiempo de procesamiento más corto (SPT).	Selecciona el trabajo que tenga el tiempo de procesamiento más corto.	Choi *et.al*.(2002)
Tiempo de procesamiento más largo (LPT).	Selecciona el trabajo que tenga el tiempo de procesamiento más largo.	Choi *et.al*.(2002)
Mayoría de las operaciones restantes (MOPR).	Selecciona la operación para el trabajo con el mayor número de operaciones restantes para ser procesadas	Choi *et.al*.(2002)
MWKR (la mayoría del trabajo restante).	Selecciona el trabajo que tenga el tiempo de procesamiento total más largo restante	Choi *et.al*.(2002)

Para enfrentar los tiempos de alistamiento se han utilizado diferentes reglas de despacho

Tabla 4.

Reglas de secuenciación orientadas a los tiempos de alistamiento.

Regla de despacho	**Descripción**	**Autores**
Tiempo de alistamiento más corto (SIMSET).	El trabajo con el tiempo de configuración más corto para la operación inminente se selecciona para el procesamiento.	Sharma y Jain (2015)
Menor suma de tiempo de alistamiento y tiempo de procesamiento (SSPT).	El trabajo con el valor más corto de la suma del tiempo de alistamiento y el tiempo de procesamiento se selecciona para el procesamiento.	Sharma y Jain (2015)

Un ejemplo de método de búsqueda local usado para el FJSP-SDST es una heurística para el subproblema de asignación llamada Enfoque por localización propuesta por Kacem et al. (2002) y ha sido utilizada por Freitas Guimaraes y Aparecida Fernandes (2006) para la Optimización Multiobjetivo.

5.5.3 Metaheurísticas. El termino metaheurísticas fue introducido por Fred Glover en 1986, este deriva de la composición de dos palabras griegas. La heurística deriva del verbo heuriskein (²υρισ κ²ιν) que significa "encontrar", mientras que el sufijo meta significa "más allá, en un nivel superior". Osman y Kelly (1996) proponen la siguiente definición: Las metaheurísticas son métodos aproximados diseñados para resolver problemas de Optimización combinatoria en los que las heurísticas clásicas no son efectivas. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos, combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos.

A continuación se presentan las metaheurísticas que han sido utilizadas para resolver el FJSP-SDST.

5.5.3.1 Búsqueda Tabú (TS). La Búsqueda Tabú es una técnica de búsqueda local guiada por el uso de estructuras de memoria adaptativa. La búsqueda comienza con una solución inicial y luego escoge en cada iteración la mejor solución en el vecindario actual incluso si no mejora la calidad de la solución actual. Para escapar de óptimos locales TS utiliza una estructura de memorización temporal en la que realiza un seguimiento de las últimas soluciones visitadas memorizando los últimos movimientos realizados: los tabúes**.** Este algoritmo ha sido aplicado para resolver el FJSP.SDST. Imanipour e el 2006 utilizó TS para encontrar la mejor

secuencia de los trabajos. Luego Saidi-Mehrabad y Fattahi (2007) utilizaron TS para resolver los dos subproblemas: asignación y secuenciación de manera simultánea.

5.5.3.2 Algoritmo Genético (GA). El primer autor John Holland (1975) en utilizar GA para la investigación formal de los mecanismos de adaptación natural. Este algoritmo se ha aplicado en diferentes problemas de Optimización combinatoria arrojando buenos resultados en comparación con otros métodos. El GA consiste en iterar en una serie de pasos o generaciones una población inicial generada aleatoriamente .En cada iteración se crea una nueva generación a partir de la anterior mediante la aplicación de operadores de selección, cruce y mutación. El GA se ha utilizado dar solución al FJSP-SDST, Freitas Guimaraes y Aparecida Fernandes (2006) utilizaron un GA para resolver el problema con optimizacion Multiobjetivo, es decir minimizando 3 objetivos: el Makespan, las cargas de trabajo totales de las máquinas y las cargas de trabajo máximas de las máquinas.

5.5.3.3 Optimización de colonias de hormigas (ACO). Es una metaheurística para problemas de Optimización combinatoria introducida en los años noventa por Dorigo y Gambardella (1997). La ACO se inspira en el comportamiento de forrajeo de algunas especies de hormigas, estas hormigas depositan feromonas en el suelo para marcar un camino favorable que deben seguir otros miembros de la colonia. Rossi y Dini (2007) aplicaron ACO en el FJSP con tiempos de alistamiento dependientes de la secuencia y tiempos de transporte.

5.5.3.4 Búsqueda Aplanadora Iterativa (IFS). Esta consiste en que dada una solución inicial, IFS aplica iterativamente dos pasos: (1) un subconjunto de decisiones para resolver son retiradas aleatoriamente de una solución actual (paso de relajación); (2) una nueva solución se

incrementalmente (paso de aplanamiento). Oddi et al. (2011) propusieron una variante de IFS que se basa en un procedimiento de búsqueda basado en la restricción del núcleo como solucionador (SP-PCP) que genera ordenamientos consistentes de actividades que requieren el mismo recurso al imponer restricciones de precedencia en una solución factible temporalmente, utilizando heurísticas de orden de variables y valores que discriminan sobre la base de la flexibilidad temporal para guiar la búsqueda.

5.5.3.5 Búsqueda de Vecindario Variable (VNS). Fue propuesta por primera vez por Mladenovic (1995). La idea central es usar dos o más estructuras de vecindario y cambiar sistemáticamente el vecindario dentro de un algoritmo de búsqueda local. El VNS fue utilizado bajo un enfoque integrado por Bagheri y Zandieh (2011) para solucionar el FJSP-SDST para minimizar el Makespan y retraso medio.

5.5.3.6 Búsqueda Local Iterativa (ILS). La Búsqueda Local Iterativa se basa en dos componentes principales: uno es la búsqueda local y otro es la perturbación. La búsqueda local tiene una alta capacidad de hacer una búsqueda intensiva alrededor de la solución actual y la perturbación, es decir, el número de componentes de la solución que son modificados. Azzouz et al. (2017) emplearon el ILS junto con el Algoritmo Genético para crear un algoritmo auto-adaptativo para dar solución al FJSP-SDST.

5.5.4 Algoritmos híbridos. El interés por este enfoque ha crecido en la comunidad académica. La fusión de metaheurísticas permite combinar diferentes conceptos con el objetivo de aprovechar la fortaleza de una y eliminar las debilidades de otra.

Talbi en el año 2002 propuso una clasificación de los algoritmos híbridos

- Intercambio de componentes entre métodos metaheurísticas: Un ejemplo es la hibridación de métodos basados en la población con métodos de búsqueda local.
- Búsqueda cooperativa: Implica el intercambio de información entre dos o más algoritmos metaheurísticas diferentes. El nivel de intercambio de información puede variar durante la implementación.
- Integración de algoritmos metaheurísticas y métodos sistemáticos: Ha producido resultados muy prometedores en casos del mundo real. Entre las implementaciones exitosas de esta forma se encuentra la combinación de algoritmos meta-heurísticos y la programación de restricciones.

Para el caso del FJSP los algoritmos híbridos pertenecen a la primera forma de hibridación. Por ejemplo González *et al*. (2013) utilizaron la Búsqueda Tabú para generar descendientes en el Algoritmo Genético, aquí varias características de la solución se comparten entre las metaheurísticas. En el 2017 Azzouz *et al* emplearon un algoritmo híbrido auto-adaptativo (SAHA) que integra el algoritmo genético (GA) y la búsqueda iterativa local (ILS).

6. Descripción del Recocido Simulado

El recocido simulado propuesto por Kirkpatrick, Gelatt, y Vecchi (1983). Este algoritmo se creó a partir de la analogía del proceso de templado de los metales, el cual consiste en 3 fases: La primera fase es de calentamiento a una determinada temperatura. En la segunda se sostiene la temperatura, lo cual permite a las moléculas acomodarse en estados de mínima energía y por último la fase de enfriamiento controlado que permite aumentar el tamaño de sus cristales y así reducir sus defectos.

En términos de optimización, el recocido simulado es un algoritmo de búsqueda local capaz de escapar del óptimo local. Inicia con una solución inicial *s*, mediante una estructura de vecindario genera una solución vecina *s´* de la solución actual. Si la evaluación de la solución vecina *s´* es menor que la solución actual, se cambia la solución actual por la solución vecina.

Por otro lado si la evaluación de *s´* es mayor que la de *s* entonces se puede empeorar eligiendo *s´* en lugar de *s* con una probabilidad de aceptación que depende de las evaluaciones $\Delta f=f(s)-f(s')$ y de temperatura actual del sistema T. Esta probabilidad permite salir de óptimos locales para llegar a óptimos globales. La probabilidad de aceptación está dada por la siguiente fórmula $P(\Delta f,T)=e^{\Delta f/T}$. La analogía entre el proceso térmico y la optimización combinatoria se presenta en la figura 10.

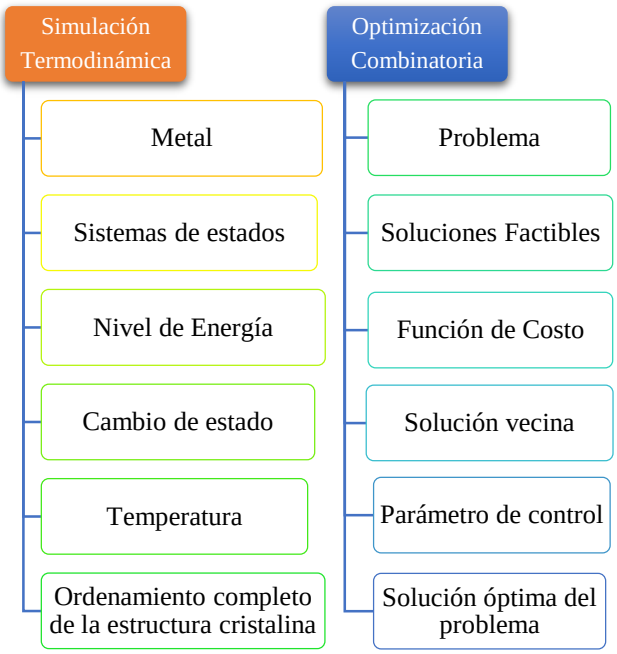

Figura 10. Relación establecida entre los elementos de simulación termodinámica y los de optimización combinatoria. Adaptado de Dowsland, K., y Adenso Díaz, B. (2001). Heuristic design and fundamentals of the Simulated Annealing. Revista Iberoamericana de Inteligencia Artificial, 35. doi:10.4114/ia.v7i19.718

Los principales componentes del Recocido Simulado son: Representación del problema, mecanismo de transición y mecanismo de enfriamiento.

6.1 Representación del problema

El primer paso para implementar este algoritmo es determinar cómo se representan las soluciones y expresar la función de costo que represente adecuadamente el costo de las soluciones.

6.2 Mecanismo de transición

- Estructura de vecindario: El objetivo es generar soluciones vecinas a partir de la solución actual realizando algunos cambios en ella. Específicamente para el *Flexible Job shop* se han empleado varios tipos, por ejemplo estructura de vecindario para secuenciación o asignación.
- Cálculo de la diferencia de costos entre la solución actual y la solución vecina. Este cálculo depende del criterio que se esté optimizando, puede variar dependiendo del tipo de problema en este trabajo el criterio de optimización es el Makespan.
- Aplicación del criterio de aceptación: El algoritmo se basa en el criterio de aceptación de Metrópolis, se genera una solución vecina y se calcula la diferencia de costos: si hay un decremento el cambio se acepta automáticamente; por el contrario, si se produce un incremento, el cambio será aceptado con una probabilidad.

6.3 Mecanismo de enfriamiento

El mecanismo de enfriamiento influye en el rendimiento del SA. En general, es especificado por varios parámetros: temperatura inicial, función de decremento de la temperatura, tamaño de vecindario y condición de parada del algoritmo.

- Temperatura inicial: No existe un método definido para calcular la temperatura inicial. En algunos casos se hace de manera experimental estableciendo un valor fijo. Por otro lado se recurre a fórmulas que se adecuen al problema. El valor de la temperatura inicial debe ser

alto para permitir movimientos hacia nuevas soluciones vecinas, sin embargo, si este valor es excesivamente alto podría moverse a cualquier lugar convirtiéndose en una búsqueda aleatoria y cuando el valor es bajo hay poco espacio de exploración.

- Función de decremento: En la literatura del Recocido Simulado hay tres leyes para reducir la temperatura: lineal, exponencial y logarítmica.

Tabla 5.

Modelos de enfriamiento

Modelo	Fórmula	Parámetros
Lineal	$T_{k+1} = T_k - c$	c
Exponencial o geométrico	$T_k = T_0 \cdot \alpha^k$ $T_{k+1} = T_k \cdot \alpha$	α
Logarítmico	$Tk + 1 = \frac{c}{\log(k+1)}$	c

El modelo exponencial es el más utilizado para resolver problemas de optimización combinatoria porque garantiza la convergencia del algoritmo. Para este modelo con valores de α cercanos a 1 el enfriamiento es suficientemente lento, lo cual permite una convergencia a soluciones óptimas.

- Tamaño de vecindario: Es el número de iteraciones en cada temperatura.
- Velocidad de enfriamiento: Está determinada por el número de iteraciones de cada temperatura, es decir, el tamaño de vecindario y la velocidad α a la que se realizará el enfriamiento.
- Condición de parada del algoritmo: En teoría, la temperatura debería reducirse a cero, pero en la práctica esto conduce a tiempos computacionales mayores. Por tal motivo se prefiere otras opciones: establecer un valor de temperatura bajo, cuando los movimientos no estén generando cambios significativos en la solución actual o un número fijo de iteraciones.

7. Descripción del Algoritmo Genético.

Los principios básicos de los algoritmos genéticos fueron establecidos por Holland (1992). Los algoritmos genéticos se basan en el procedimiento de selección natural y genética natural, para ello se trabaja sobre una población de individuos, cada individuo tiene asociado un ajuste de acuerdo, a la bondad con respecto al problema de la solución que represente. En la naturaleza es equivalente al grado de efectividad de los organismos para competir por recursos. Cuanto mayor sea la adaptación de un individuo del problema, mayor será la probabilidad de que el individuo sea seleccionado para reproducirse, luego se cruza su material genético con otro individuo que fue seleccionado de la misma forma. Este cruce generará nuevos individuos, los cuales comparten material genético de sus padres y también existe la posibilidad de que alguno de estos sufra algún tipo de mutación.

Conceptos importantes de Algoritmo Genético: El conjunto de soluciones se denomina población. Cada población está constituida por un número determinado de individuos representados como cromosomas El cromosoma consiste en una secuencia de genes que puede tomar algunos valores llamados alelos, estos valores deben adaptarse al problema. Los operadores que intervienen en el GA son de: selección, cruce y mutación.

7.1 Representación de la solución

Como se había mencionado anteriormente el *Flexible Job Shop* se compone de dos subproblemas: secuenciación (Job shop) y asignación. Abdelmaguid en el año 2010 clasificó las representaciones del GA para el job shop, como se muestra en la figura 11.

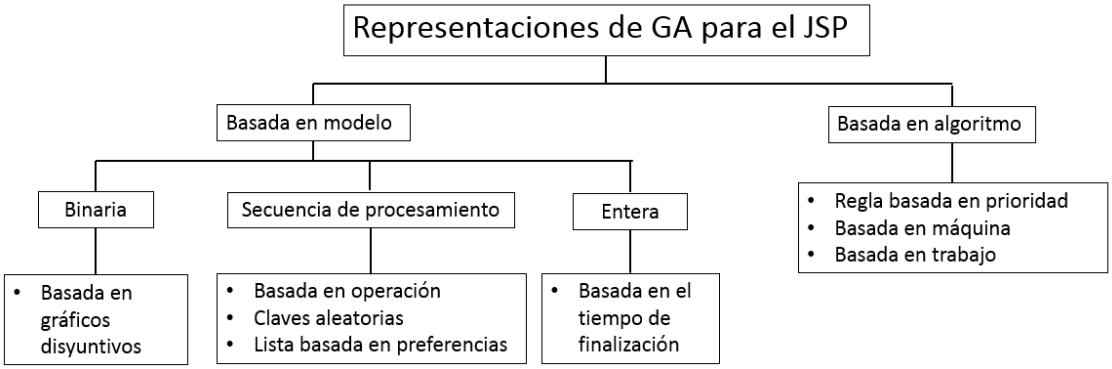

Figura 11. Tipos de representaciones GA para la JSP. Adaptado de Abdelmaguid, Tamer F.(2010) Representations in Genetic Algorithm for the Job Shop Scheduling Problem: A Computational Study[Figura] .Recuperado de https://www.scirp.org/Journal/PaperInformation.aspx?paperID=3561y.

Partiendo de esta clasificación se han propuesto varios tipos de codificaciones para el *Flexible Job Shop* a continuación se muestra las más utilizadas.

7.1.1 Lista de secuencia de tareas (TSL). La lista de secuencia de tareas fue propuesta por Kacem et al. (2002) y utilizada por Guimarães et al. (2006), los anteriores autores coinciden en que tiene en cuenta las restricciones de precedencia con el fin de garantizar soluciones factibles. En la TSL cada operación está formada por 3 indicadores (i, j, k) uno para cada operación donde i es el trabajo al que pertenece la operación; j es el número de esa operación dentro de trabajo i; k es la máquina asignada a esa operación. La longitud de la cadena es igual al número total de operaciones.

$$S= (O_{11}, M_1), (O_{21}, M_3), (O_{12}, M_2), (O_{22}, M_2), (O_{23}, M_3)$$

(1,1,1)	(2,1,3)	(1,2,2)	(2,2,2)	(2,3,3)

Figura 12. Representación de la lista de secuencia de tareas. Adaptado de Lee, K., y Yamakawa, T. (1998). A genetic algorithm for general machine scheduling problems. Int.Conf. on Conventional and knowledge-Based Electronics Systems Australia, 2(2), 60-66. doi:10.1109/KES.1998.725893

7.1.2 Representación de dos vectores. Este tipo de representación fue propuesta por Gen et al. (2009). La representación de dos vectores está conformada por un vector de secuencia de operación y el otro vector de asignación de máquina. Para el vector de secuencia la representación basada en operación, la longitud del vector es igual al número total de operaciones. Para el vector de asignación, los números representan las máquinas asignadas a las operaciones. Según Gen et al. (2009) las principales ventajas de la representación de dos vectores son;, que cada posible cromosoma siempre representa una solución factible candidata, y que el espacio de codificación es menor que el de la representación de permutación, la cual genera más espacio de soluciones, esta es quizás la representación más natural de las secuencias de operación pero debido a la existencia de restricciones de precedencia, no todas las permutaciones de las operaciones definen secuencias factibles.

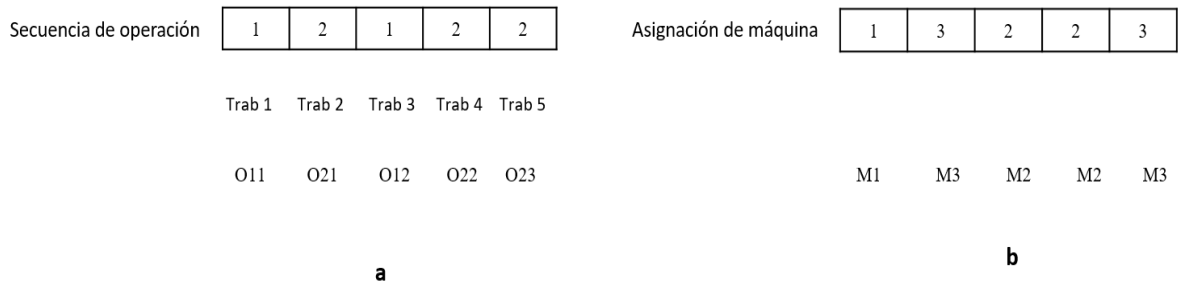

Figura 13. La representación de dos vectores: a) vector de secuencia de operación. b) vector de asignación de máquina. Adaptado de Gen, M., Gao, J., y Lin, L. (2009). Multistage-based genetic algorithm for flexible job-shop scheduling problem. Studies in Computational Intelligence, 183-196.*Recuperado de https://doi.org/10.1007/978-3-540-95978-6_13*

7.1.3 Representación cromosómica MSOS. Li y Gao (2016) proponen una representación mejorada del FJSP la cual no requiere mecanismo de descodificación, la representación cromosómica tiene dos componentes: Selección de máquina y secuencia de operación.

Selección Maquina(MS)					Secuencia de operación (OS)				
1	2	1	2	2	1	3	2	2	3

Figura 14. Representación cromosómica MSOS. Adaptado de Li, X., y Gao, L. (2016). An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. International Journal of Production Economics, 174, *93–110. Recuperado de https://doi.org/10.1016/j.ijpe.2016.01.016*

7.1.4 Matriz Binaria. El diseño del cromosoma es una matriz binaria donde: las filas corresponden a todas las operaciones de los trabajos. Además, el orden en que aparece en el cromosoma describe la secuencia de operaciones presente en la solución. Las columnas corresponden a todas las máquinas. En esta representación se presenta una restricción $\sum Xijk = 1$, donde $X_{ijk}=1$, si *Oij* es asignada al recurso *Mk*, y $X_{ijk}=0$ de lo contrario. La suma de cada fila debe ser igual a 1, para garantizar que cada operación es asignada a solo una máquina.

	M1	M2	M3
O_{11}	0	0	1
O_{21}	1	0	0
O_{22}	0	1	0
O_{12}	0	0	1
O_{13}	1	0	0
O_{23}	0	0	1
O_{31}	0	1	0
O_{32}	0	1	0
O_{33}	1	0	0

Figura 15. Matriz binaria

7.2 Criterio de selección

La selección se encarga de la elección de los individuos para la reproducción. Según la revisión de literatura se encontraron diferentes criterios de selección: selección por ruleta, selección por torneo y selección lineal.

7.2.1 Selección por ruleta. En la selección por ruleta los individuos son distribuidos de acuerdo a su valor de adaptabilidad donde los mejores recibirán una mayor porción en la ruleta y por ende tendrán mayor probabilidad de ser seleccionados. Se genera un número aleatorio de 0 a 1 y devuelve el individuo que corresponda a la posición, esta se obtiene recorriendo los individuos de la población y acumulando sus porciones de ruleta hasta que la suma exceda el valor obtenido.

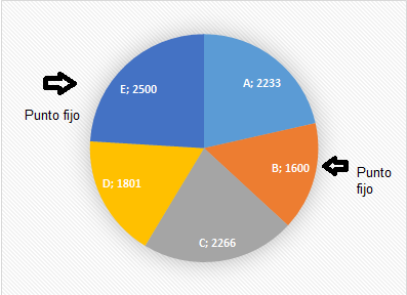

Figura 16. Selección por ruleta

7.2.2 Selección por torneo tamaño n. El individuo para la reproducción se elige entre un número aleatorio de individuos. La selección por torneo es muy utilizado en algoritmos evolutivos, funciona muy bien para una amplia gama de problemas.

Existen 2 tipos de selección mediante torneo: torneo determinístico y torneo probabilístico. El torneo determinístico consiste en elegir de manera aleatoria un número n de individuos, la forma más simple es cuando n=2. Dentro de los individuos seleccionados se elige el más apto para que continúe en la siguiente generación.

Por otro lado, el torneo probabilístico para seleccionar un individuo genera un número aleatorio entre 0 y 1 y se compara con una probabilidad de selección predeterminada. Si el valor aleatorio es menor o igual que la probabilidad de selección, entonces se selecciona el

candidato más ajustado, de lo contrario se elige el candidato más débil. Usualmente la probabilidad de selección es mayor a 0.5.

7.2.3 Selección lineal. Los individuos se clasifican de acuerdo a su aptitud, un rango r ∈ {1,2,..., N} es asignado a cada uno, donde N es el tamaño de la población. El mejor individuo obtiene un rango N mientras que el peor individuo un rango 1.

$$p(i) = \frac{2\mathrm{ri}}{\mathrm{N(N+1)}}, (i = 1,2, \dots N)$$

P(i) es la probabilidad de escoger el individuo i en la clasificación.

7.3 Operadores

Según lo encontrado en la literatura se distinguen dos tipos de operadores para el FJSP. Los operadores de asignación y los operadores de secuenciación.

Los operadores de asignación solo cambian la propiedad de asignación de los cromosomas, es decir, la secuenciación de las operaciones se conserva en la descendencia. Por otro lado, los operadores de secuenciación sólo cambian la secuencia de las operaciones en los cromosomas de la descendencia, es decir, la asignación de operaciones a máquinas se conserva. Estos operadores deben respetar las restricciones de precedencia.

Los operadores de asignación y secuenciación que se han aplicado con mayor frecuencia en el FJSP debido a las características del problema se muestran en la siguiente tabla.

Tabla 6.

Operadores de cruce y mutación

Tipo de operador / proceso	De asignación	De secuenciación
Cruce	Cruce de dos puntos. Operador de cruce uniforme Operador de cruce de un punto	PPS JOX
Mutación	Mutación inteligente. POX	Selección aleatoria Búsqueda de vecindario

A continuación se mostrará cómo funcionan los diferentes operadores nombrados en la tabla anterior.

7.3.1 **Cruce.** La operación de cruce selecciona los genes de los cromosomas progenitores y crea una nueva descendencia.

7.3.1.1 ***Operador de cruce uniforme*.** El operador de cruce uniforme consiste en crear una máscara de manera aleatoria y se llama alfa, es decir, un vector binario uniforme en la longitud del cromosoma. Luego se copian los genes del padre 1 en el hijo 1 en las posiciones donde alfa es 1, y si alfa es igual a cero copiar genes del padre 2 al hijo 1.

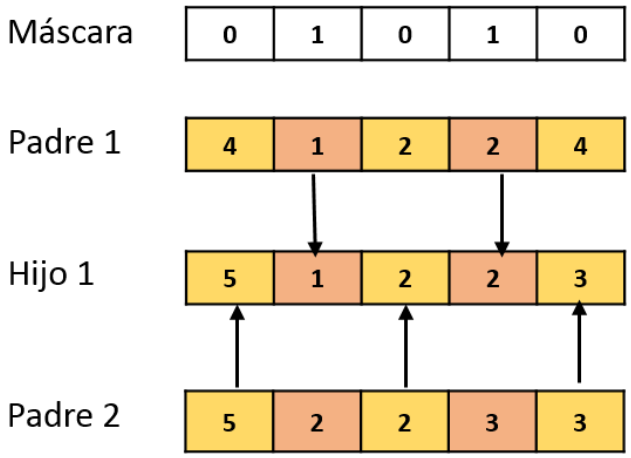

Figura 17. Operador de cruce uniforme

7.3.1.2 Operador de cruce de un punto. En el operador de cruce de un punto se elige un punto de corte para crear dos descendientes, los genes de los dos padres después del punto de corte se intercambian .En la siguiente figura se muestra un ejemplo, el punto de cruce es después del gen 4. El primer hijo copia los genes de antes del punto de cruce del padre 1, el resto las copias del padre 2. Para el segundo hijo copia los genes de antes del punto de cruce del padre 2, el resto los copia del padre 1.

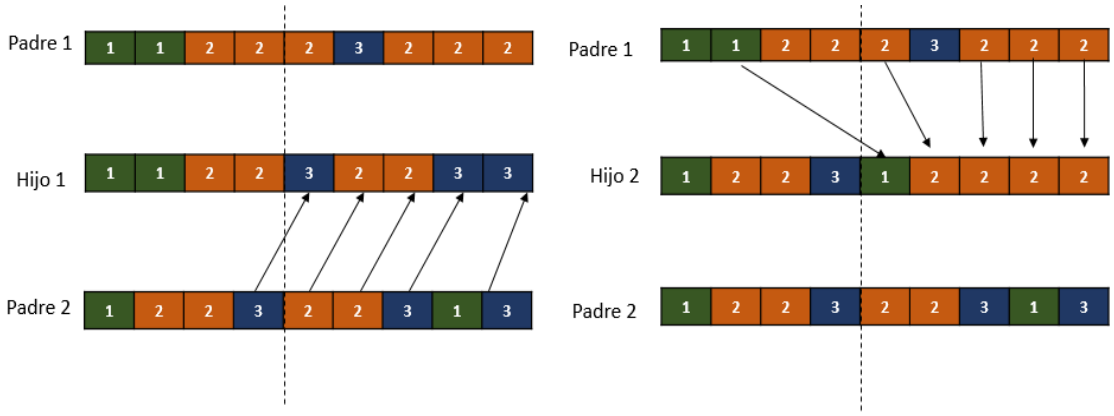

Figura 18. Operador de cruce de un punto

7.3.1.3 Operador de cruce de dos puntos. El operador de cruce de dos puntos consiste en copiar los genes del primer padre que están entre los dos puntos de cruce (a y b) y se rellenan los faltantes con los genes del segundo padre como se representa en la figura 19.

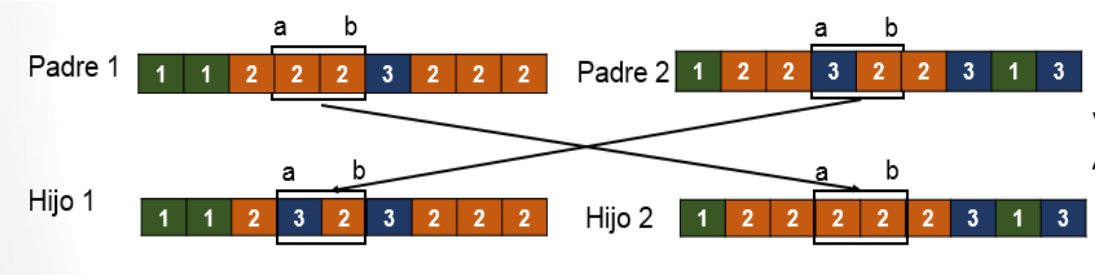

Figura 19. Operador de cruce de dos puntos

7.3.1.4 Operador POX (Precedence Preserving Order-based). Este operador fue desarrollado por Lee y Yamakawa (1998). El cual genera dos hijos a partir de dos padres respetando las restricciones de precedencia.

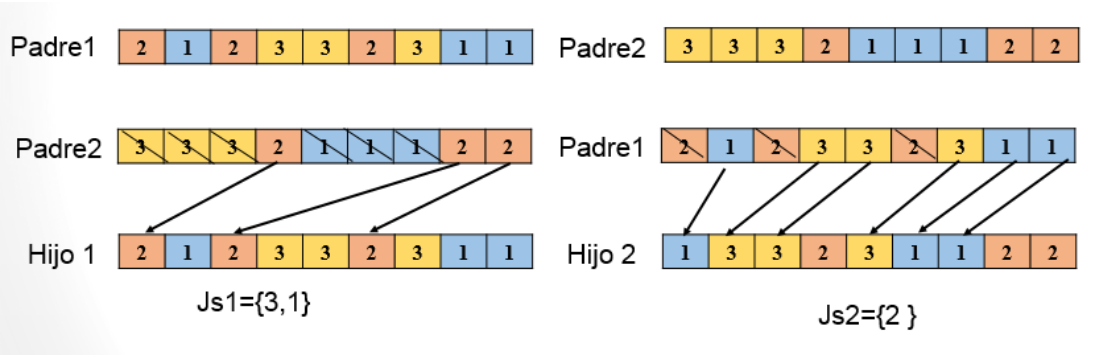

Figura 20. Operador de cruce POX

El operador POX funciona de la siguiente manera:

1) Se selecciona el padre1 y el padre 2

2) Determinar para cada padre, el conjunto de trabajos que serán heredados a su correspondiente hijo. Para el padre 1, los trabajos {3,1} y para el padre 2, el trabajo 2.

3) Los trabajos del padre 1, toman la misma posición en el hijo 1, igualmente para el trabajo del padre 2, toma la misma posición en el hijo2.

4) Luego se eliminan del padre 1 los trabajos obtenidos del padre 2 y se coloca el resultado en el primer hijo según el mismo orden de aparición y viceversa para el segundo hijo.

5) Finalmente se crean dos hijos factibles.

7.3.2 **Mutación.** La mutación introduce diversidad en la población, generando variaciones aleatorias en un individuo.

7.3.2.1 ***Operador PPS (Precedence Preserving Shift).*** Tomando como referencia el operador PPS que utilizaron He, Weng, y Fujimura (2017) en su investigación. El operador consiste en seleccionar un gen del vector de secuenciación de manera aleatoria, luego, el gen se desplaza a otra posición, mientras que la operación que este gen representa en su trabajo no cambia. En la siguiente figura se selecciona el tercer 3,que representa la tercera operación del trabajo 3, por lo tanto solo se puede desplazar entre el rango del segundo 3 y el tercer 3,los cuales representan la segunda y tercera operación del trabajo 3.Como mencionan los autores He et.al,para este caso si el 3 se selecciona fuera de este rango, antes del segundo 3 ,entonces el 3 se convertiria en el segundo 3, o en el primer 3,y ya no representaria la tercera operación del trabajo 3,esta situacion influiría mucho en la estructura del cromosoma original.

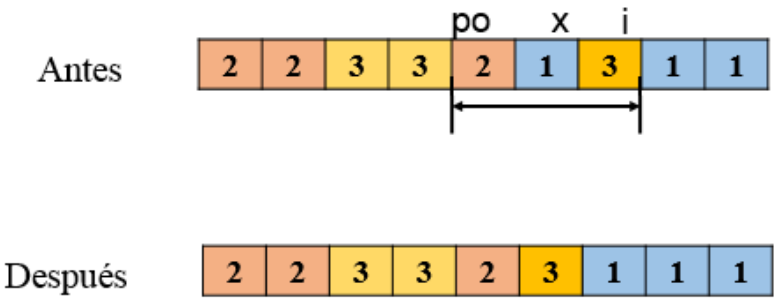

Figura 21. Operador de mutación PPS

7.3.2.2 Mutación inteligente. Se selecciona aleatoriamente una operación en la máquina con la carga de trabajo máxima y se le asigna a la máquina con la carga de trabajo mínima. Este operador solo cambia la asignación de la máquina de una sola operación en el individuo. En la investigación de Wan, Zhang, Fan, y Bai (2010) describen el operador en los siguientes pasos:

1) Se selecciona una máquina Mk en el vector de máquina y se identifica su correspondiente trabajo y operación.
2) Borrar del vector de máquina la operación identificada en el paso anterior.
3) Seleccionar un vector de máquina diferente e insertar la operación en este vector.

7.3.2.3 ***Operador de mutación MS.*** El operador de mutación MS propuesto por Zhang, Gao, y Shi (2011) actúa de la siguiente manera. Cambia la propiedad de asignación de los cromosomas, selecciona el tiempo de procesamiento más corto de la máquina alternativa para equilibrar la carga de trabajo de las máquinas.

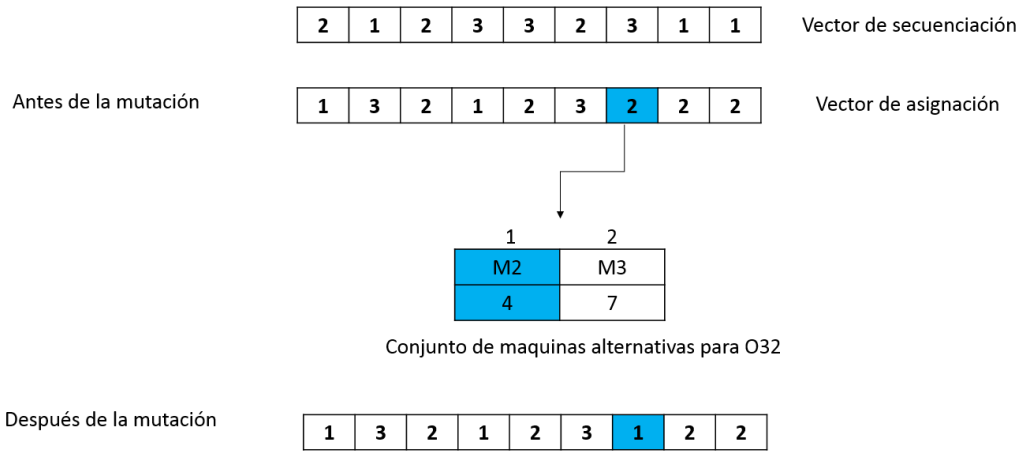

Figura 22. Operador de mutación MS

8. Problema "Flexible Job Shop Con Tiempos de Alistamiento Dependientes de La Secuencia (FJSP-SDST).

Se realizó una revisión de los modelos propuestos en la literatura para representar el problema y el que más se ajusta es el propuesto por Choi y Choi(2002) y modificado por Saidi-Mehrabad y Fattahi (2007) asumiendo que los tiempos de alistamiento solo depende de los trabajos.

8.1 Descripción del problema

El *Flexible Job Shop* es una extensión del Job Shop, el cual implica un nivel más de decisión (la asignación) debido a que se cuenta con un número determinado de máquinas disponibles que pueden realizar cada operación. Adicionalmente se consideran los tiempos de alistamiento dependientes de la secuencia, es decir, el tiempo de alistamiento de una máquina para procesar un trabajo k después de procesar un trabajo j, puede tomar una cantidad de tiempo diferente que alistar la máquina para la transición contraria.

Las condiciones y restricciones del problema son las siguientes:

- Las máquinas son independientes una de otra.
- Los trabajos son independientes de uno a otro. No hay restricciones de precedencia entre operaciones de diferentes trabajos. No obstante, existen restricciones de precedencia entre las operaciones del mismo trabajo.
- Todos los trabajos y máquinas están disponibles en el tiempo cero.
- En un momento dado, una máquina sólo puede ejecutar una operación.
- No se permite la anticipación. Es decir, cada operación debe ser completada sin interrupción, una vez que se inicia.
- El tiempo de procesamiento de la $O_{j,h}$ en la máquina i es $p_{i,j,h.}$
- Los tiempos de alistamiento dependen de la secuencia.

8.2 Formulación del problema

El siguiente modelo es el propuesto por Saidi-Mehrabad y Fattahi (2007). El problema bajo consideración tiene *m* máquinas y *n* trabajos. Cada trabajo consiste de una secuencia de operaciones $O_{j,h}$, h=1,…,h_j, donde $O_{j,h}$ y h_j denota la h operación del trabajo j y el número de operaciones requerido para el trabajo j, respectivamente. El índice i denota la máquina, índices j y k denotan los trabajos, y h y l denota los índices de las operaciones.

Dejar $O_{i,j,h}$ denota la operación $O_{j,h}$ realizada en una máquina especifica i. Suponer que $O_{i,j,h}$ es programado para seguir inmediatamente después de $O_{i,k,l}$. Luego, $O_{i,j,h}$ requiere tiempo para un alistamiento, $s_{i,k,l,j,h}$ en adición al tiempo de procesamiento, $p_{i,j,h.}$. En este modelo se asume que $S_{i,k,l,j,h} = S_{i,k,j}$. El problema consiste en encontrar un programa que minimice el Makespan dado n, m, $O_{j,h,}$ $M_{j,h,}$ $S_{i,k,j}$ y $p_{i,j,h..}$

Índices

i máquinas

j y k trabajos

h y l operaciones

Parámetros

n número de trabajos

m número de máquinas

M_{jh} Conjunto de máquinas alternativas que pueden procesar la operación h del trabajo j.

$O_{j,h}$ denota la operación h del trabajo j.

$a_{i,j,h}$ Describe el conjunto de máquinas capaces M_{jh} para cada operación O_{jh}

$$a_{i,j,h} = \begin{cases} 1 \text{ si } O_{j,h} \text{ puede ser realizada en la máquina i} \\ 0 \text{ de lo contrario} \end{cases}$$

$p_{i,j,h.}$ tiempo de procesamiento de la operación h del trabajo j en la máquina i.

$S_{i,k,j}$ tiempo de alistamiento en la máquina i que requiere la operación del trabajo k después de ser procesada la operación del trabajo j.

M: un número grande

8.2.1 Variables

$$Y_{i,j,h} = \begin{cases} 1 \text{ Si } O_{j,h} \text{ es realizada en la máquina i} \\ 0 \text{ de lo contrario} \end{cases}$$

$$X_{i,j,h,k,l} = \begin{cases} 1 \text{ Si } O_{i,j,h} \text{ precede } O_{i,k,l} \text{ inmediatamente} \\ 0 \text{ de lo contrario} \end{cases}$$

$t_{j,h}$:tiempo de inicio de procesamiento de la operación $O_{j,h}$

$f_{j,h}$: tiempo de finalización de procesamiento de la operación $O_{j,h}$

C_{max}: Makespan del programa

8.2.2 Restricciones

$t_{j,h} + Y_{i,j,h} * p_{i,j,h} \leq f_{j,h}$ para i =1,…,m; j=1,…,n; h=1,2,…,h_j (1)

$f_{j,h} \leq t_{j,h+1}$ para j=1,…,n; h=1,…,h_{j-1} (2)

$f_{j,hj} \leq C_{max}$ para j=1,…,n (3)

$Y_{i,j,h} \leq a_{i,j,h}$ para i =1,…,m; j=0,…,n; h=1,2,…,h_j (4)

$t_{j,h} + p_{i,j,h} + s_{i,j,k} \leq t_{k,l} + (1- X_{i,j,h,k,l})M$ para j=0,…,n; k=1,…,n; h=1,2,…,h_j; l=1,2,…,h_k; i =1,…, m (5)

$f_{j,h} + s_{i,j,k} \leq t_{j,h+1} + (1- X_{i,k,l,j,h+1})M$ para j=1,…,n; k=0,…,n; h=1,2,…,h_{j-1}; l=1,2,…,h_k; i =1,…, m (6)

$\sum_i Y_{i,j,h} = 1$ para h=1,2,…,h_j; j=0,1,…,n; (7)

$\sum_j \sum_h X_{i,j,h,k,l} = Y_{i,k,l}$ para i=1,…,m; k=1,…,n; l=1,2,…,h_k (8)

$\sum_j \sum_h X_{i,j,h,k,l} = Y_{i,j,h}$ para i=1,…,m; j=0,1,…,n; h=1,2,…,h_j (9)

$X_{i,j,h,j,h} = 0$ para i=1,…,m; j=0,1,…,n; h=1,2,…,h_j (10)

$t_{j,h} \geq 0$ para j=0,1,…,n; h=1,2,…,h_j (11)

$f_{j,h} \geq 0$ para j=0,1,…,n; h=1,2,…,h_j (12)

$Y_{i,j,h} \in \{0,1\}$ para i=1,…,m ; j=0,1,…,n; h=1,2,…,h_j (13)

$X_{i,j,h,k,l} \in \{0,1\}$ para i=1,…,m ; j=0,1,…,n; k=1,…,n; h=1,2,…,h_j; l = 1,…,h_k (14)

Las restricciones (1) y (2) hacen referencia a la restricción de precedencia entre dos operaciones del mismo trabajo. La Restricción (3) determina el tiempo de finalización de la última operación, es decir, el Makespan (C_{max}).

La restricción (4) exige que la máquina para cada $O_{j,h}$ debe ser seleccionada de las máquinas alternativas para $O_{j,h}$. Las restricciones (5) y (6) hace cumplir que cada máquina debe procesar

una operación al tiempo y considerar los tiempos de alistamiento. La restricción (7) obliga que una sola máquina debe ser seleccionada para $O_{j,h}$.

Las restricciones (8) y (9) definen permutaciones circulares de operaciones en cada máquina. Elimina operaciones alternativas que se excluyen en un programa final. La restricción (8) selecciona una operación $O_{i,j,h}$ que precede inmediatamente a una operación alternativa programada $O_{i,k,l}$ y la restricción (9) selecciona una operación $O_{i,k,l}$ que sigue inmediatamente a una operación alternativa programada $O_{i,j,h}$. Una permutación circular de operaciones en una máquina produce una secuencia programada de las operaciones en la misma máquina.

Las restricciones (10), (11) y (12) definen las restricciones de no negatividad. Por último las restricciones (13) y (14) definen las variables binarias.

8.2.3 Función de costo. La función de costo es el criterio para medir la calidad de una solución propuesta. En este trabajo se utiliza el Makespan, es decir, el tiempo de finalización de la última operación.

Min C_{max}

8.2.4 Ejemplo. Para ilustrar el problema se utilizó un ejemplo descrito por Azzouz et al. (2016) este cuenta con 3 trabajos, la tabla 7 muestra los tiempos de procesamiento de las operaciones en las máquinas alternativas y la tabla 8 muestra los tiempos de alistamiento de los trabajos en cada una de las máquinas, incluye el tiempo de alistamiento Dummy, es decir, el tiempo de alistamiento de la primera operación en cada máquina.

Tabla 7.

Tiempos de procesamiento

Trabajo	Operación	M1	M2	M3
1	O11	4	-	5
	O12	-	3	4
	O13	6	5	-
2	O21	3	-	4
	O22	4	5	-
	O23	-	4	7
3	O31	5	3	-
	O32	-	4	-
	O33	4	5	3

Tabla 8.

Tiempos de Alistamiento dependientes de la secuencia

	Máquina 1			Máquina 2			Máquina 3		
	T1	T2	T3	T1	T2	T3	T1	T2	T3
Dummy	3	2	1	2	4	1	4	3	2
T1	0	1	3	0	2	4	0	2	3
T2	1	0	2	4	0	3	2	0	4
T3	1	3	0	2	1	0	3	2	0

Nota: T1, representa el trabajo 1, T2 el trabajo 2 y así sucesivamente.

9. Algoritmo Híbrido propuesto al FJSP-SDST

En el algoritmo híbrido se utiliza el Recocido Simulado para la construcción de una porción de la población inicial, generando buenas soluciones iniciales para el Algoritmo Genético.

9.1 Parámetros de entrada

El FJSP-SDST tiene los siguientes parámetros:

- Número de trabajos: Cantidad de trabajos que deben ser procesados en las diferentes máquinas.
- Número de operaciones: Cantidad de operaciones que conforman un trabajo. Para el algoritmo, el número de operaciones es representado por medio de un vector llamado v, cada columna del vector representa el número de operaciones del correspondiente trabajo, por ejemplo en el siguiente vector cada trabajo tiene 3 operaciones.

  V= {3, 3, 3} => {O11 O12 O13 O21 O22 O23 O31 O32 O33}

- Número de máquinas: Cantidad de máquinas en las que podrán ser procesadas las operaciones.

  mx={2 2 2 2 2 2 2 1 3 }

- Matriz de máquinas disponibles por operación (con_maq): Siguiendo con el ejemplo mencionado en la sección 8.2.4 se muestra la tabla 9, cada número representa el número de máquina donde se puede procesar la operación. Esta matriz está relacionada con el vector mx, el cual, indica la cantidad de máquinas donde se puede procesar la operación.

  Tabla 9.

  Matriz de máquinas disponibles por operación

Trabajo 1			Trabajo 2			Trabajo 3		
O11	O12	O13	O21	O21	O23	O31	O32	O33
1	2	1	1	1	2	1	2	1
3	3	2	3	2	3	2	0	2
0	0	0	0	0	0	0	0	3

Basados en que, algunos autores como Sun *et al*. (2010) manifiestan que para problemas de flexibilidad parcial cuando una máquina no puede realizar una determinada operación el tiempo de procesamiento lo cambian artificialmente a un valor relativamente grande, lo cual puede generar soluciones infactibles. Teniendo en cuenta lo anterior, para que las soluciones sean factibles al problema, fue necesario realizar modificaciones en la forma de representar para extraer los datos de entrada. Para el proceso de asignación de máquinas, se genera un número aleatorio entre 1 y el total de máquinas para cada operación indicadas en el vector mx, este número indica la posición de la máquina elegida en la Matriz de máquinas disponibles por operación (con_maq), por ejemplo para l O11 el número aleatorio es 2 entonces la máquina seleccionada es la 3 porque está en la posición 2. Esto permite soluciones factibles porque, solo se escogen las máquinas disponibles para cada operación.

Luego, teniendo seleccionada la máquina 3 de igual manera el número de la máquina en este caso 3 indica la posición donde se busca el tiempo de procesamiento en la matriz de tiempos de procesamiento. En la matriz de tiempos de procesamiento se coloca cero en la posición de la máquina donde no se puede realizar cada operación

- Matriz de tiempos de procesamiento (tmo): Tiempos de procesamiento de las operaciones en cada una de las máquinas.

Tabla 10.

Matriz de tiempos de procesamiento

Trabajo 1			Trabajo 2			Trabajo 3		
O11	O12	O13	O21	O21	O23	O31	O32	O33
4	0	6	3	4	0	5	0	4
0	3	5	0	5	4	3	4	5
5	4	0	4	0	7	0	0	3

- Matriz de tiempos de alistamiento (talist): Tiempos de alistamiento que se requiere cuando se cambia de un trabajo a otro en cada máquina. Es necesario decir que los tiempos de alistamiento dependen solo del trabajo y no de la operación en sí.

Tabla 11.

Matriz de tiempos de alistamiento

	Máquina 1			Máquina 2			Máquina 3		
	T1	T2	T3	T1	T2	T3	T1	T2	T3
T1	0	1	3	0	2	4	0	2	3
T2	1	0	2	4	0	3	2	0	4
T3	1	3	0	2	1	0	3	2	0

- Matriz de tiempos de alistamiento Dummy: Tiempos de alistamiento iniciales que se requieren para la primera operación de cada máquina.

Tabla 12.

Matriz de tiempos de alistamiento Dummy

	T1	T2	T3
Máquina 1	3	2	1
Máquina 2	2	4	1
Máquina 3	4	3	2

Como el problema se soluciona mediante un AHG, es necesario nombrar los parámetros del Recocido Simulado y el Algoritmo Genético. El recocido simulado está compuesto por los siguientes parámetros: Temperatura inicial, función de decremento de temperatura, tamaño de

vecindario y criterio de parada. Para el Algoritmo Genético los parámetros son: Tamaño de población, número de generaciones, probabilidad de cruce y probabilidad de mutación.

9.2 Representación de la solución

La manera de representar los individuos influye en la eficiencia del algoritmo propuesto. En este trabajo se utiliza la representación de dos vectores, el primero representa la secuencia de operación en las máquinas, y el segundo la asignación de la máquina. Para la primera representación, se usa la *representación basada en operaciones*; este tipo de representación cubre todo el espacio de la solución y cualquier permutación de los operadores puede corresponder a un calendario factible. Por otro lado, para la representación de asignación de máquinas, se asigna la máquina dependiendo del conjunto de máquinas disponibles para realizar cada operación.

Para ilustrar la solución del problema mediante las dos representaciones cromosómicas, se utiliza el mismo ejemplo presentado en la sección 8.2.4.

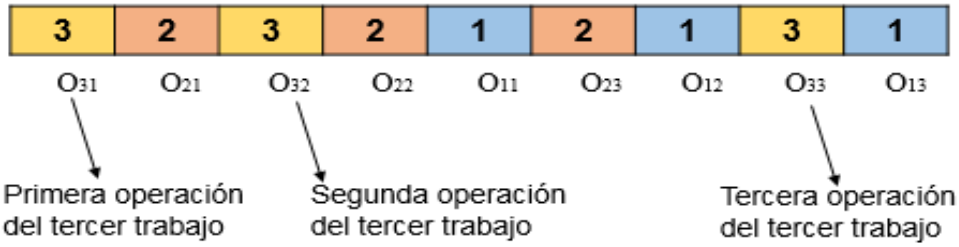

Figura 23. Cromosoma de secuenciación

En el cromosoma de secuenciación se evidencia que la primera operación a realizar es $O_{3,1}$

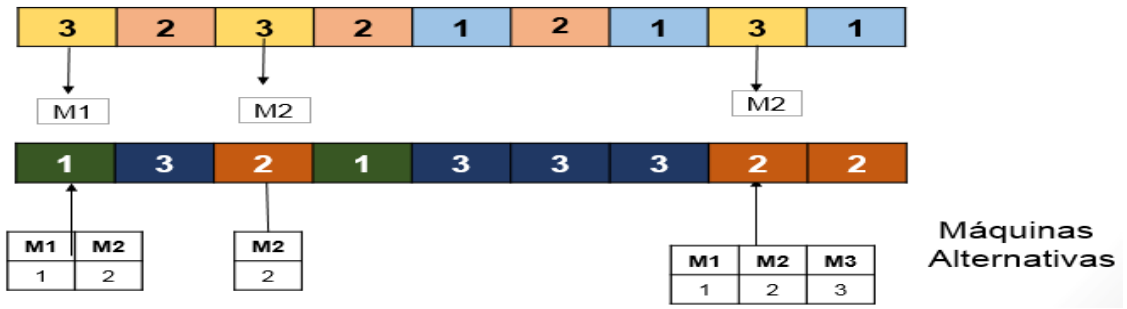

Figura 24. Cromosoma de asignación

De acuerdo con los dos cromosomas: el de secuenciación y asignación, la primera operación a realizar es la O_{31}, la cual es asignada a la máquina 1; la segunda operación es la O_{21}, la cual es asignada a la máquina 3; la tercera operación es la O_{32}, la cual es asignada a la máquina 2 y así sucesivamente hasta completar el total de operaciones.

También se evidencia en la figura 24 que la máquina 1 se selecciona para realizar la operación O_{31}, este valor en el vector puede cambiar a 2 porque las máquinas que pueden realizar esta operación son la M1 y M2.

9.3 Algoritmo Híbrido Genético

El algoritmo Híbrido Genético está compuesto principalmente por un algoritmo genético con una modificación en la población inicial, porque parte de ella es generada por el recocido simulado, después, se busca mejorar la población inicial mediante criterios de selección, operadores de cruce y mutación.

9.3.1 Población inicial.

El cálculo de la población inicial es importante para el Algoritmo Genético porque influye directamente en la tasa de convergencia y en la calidad de las soluciones óptimas. La población inicial es obtenida por una combinación de dos métodos: 1) aleatorio y 2) Recocido Simulado.

9.3.1.1 Recocido Simulado.

Para definir el porcentaje de la población inicial que se generaría por Recocido Simulado se realizó un análisis de varianza (ver apéndices: C e I.) obteniendo que el mejor porcentaje de población generado con el Recocido simulado es de 20% por la calidad de la solución y el tiempo computacional, si bien sus resultados de Makespan eran similares a los del 30% y 50%

el tiempo computacional de estos últimos era mayor por ese motivo se descartaron. En la figura 25 se puede evidenciar la diferencia que existe en los tiempos computacionales para el 20%,30% ,50% de Recocido Simulado en la población inicial para la instancia AEB01 de 10 trabajos y 5 máquinas, el mismo comportamiento se presenta en la figura 26 con la instancia AEB16 de 15 trabajos y 10 máquinas.

El otro 80% de la población inicial se genera de manera aleatoria.

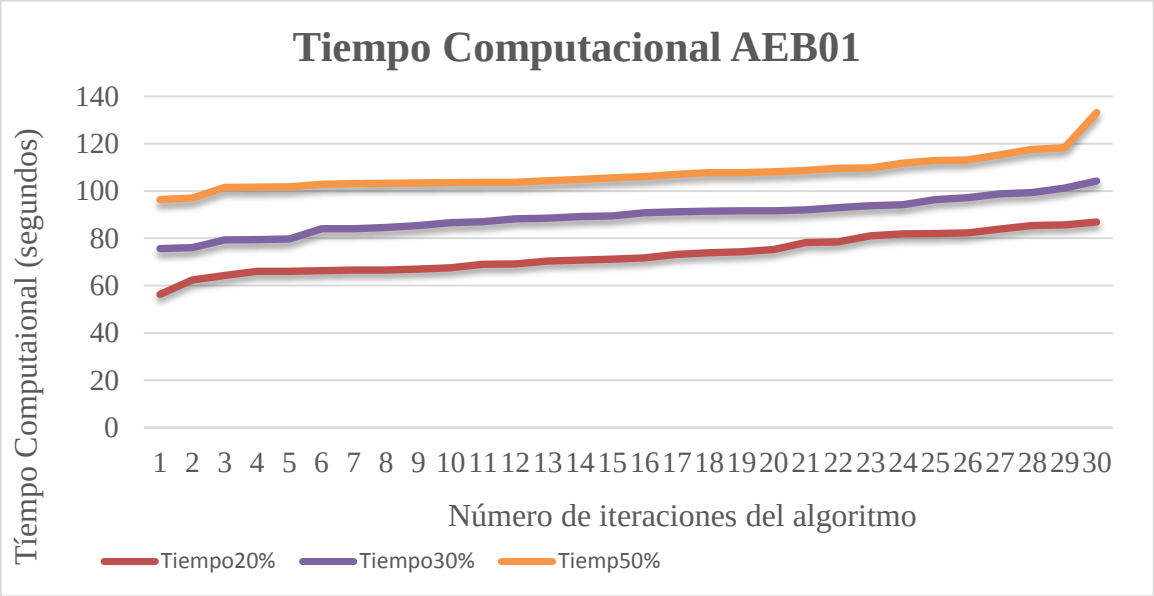

Figura 25. Tiempo computacional AEB01

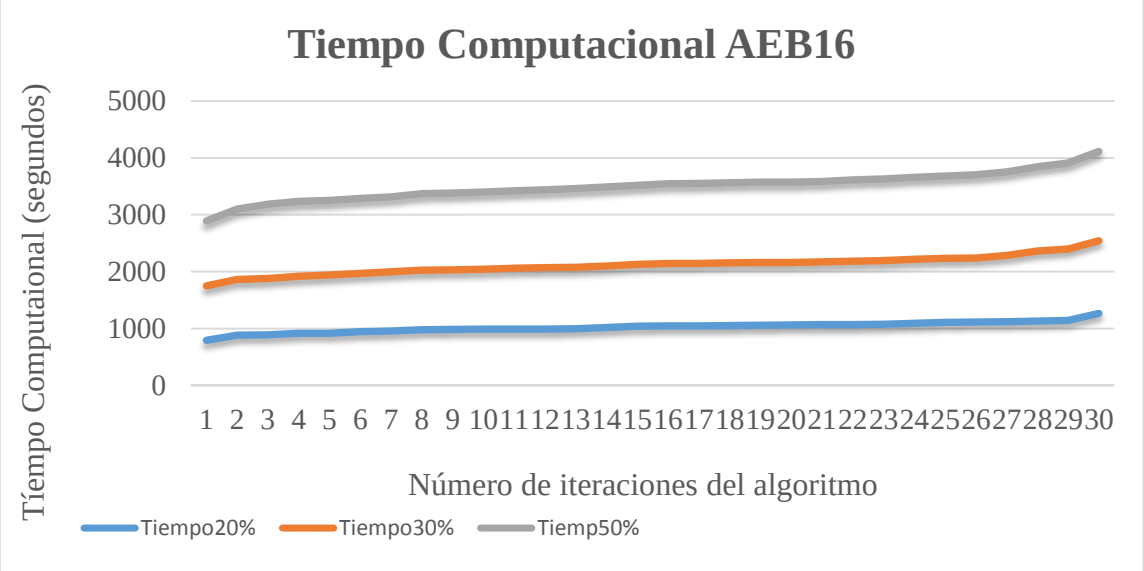

Figura 26. Tiempo computacional AEB16

9.3.1.1.1 Cálculo de los parámetros del mecanismo de enfriamiento. Los parámetros son los siguientes

- Temperatura inicial*:* Se calculó mediante la siguiente fórmula

$$Ti = e^{-\frac{\alpha.fo}{To}} \geq 0.9 \; para \; valores \; de \; \alpha \leq 0.2$$

La temperatura de inicio se establece para que haya una probabilidad de 0.9 de aceptar soluciones que desmejoren la solución inicial hasta en un 20% del valor inicial como lo propuso Bula (2004).

- Criterio de parada: Este se determinó que pasados 3 cambios de temperatura al realizarse la última iteración de la temperatura final el algoritmo termina. Porque al aumentar el número de cambios de temperatura, aumenta también el número de iteraciones, lo cual influye directamente en el tiempo computacional.
- Función de decremento: Se utilizó un modelo exponencial para realizar el enfriamiento con la fórmula $T_{k+1} = T_k \cdot \alpha$, con α de 0.5 (ver apéndice: E y K) el cual permite observar mejores soluciones ajustándose de mejor manera al criterio de parada.
- Número de vecindario: Para determinar el valor del tamaño de vecindario se usó un diseño de experimentos, en el cual se definieron dos niveles: nivel bajo 10 y nivel alto 50.Se tomaron estos valores para medir qué impacto tenía la cantidad de vecinos en cada temperatura, dando como resultado de este diseño que el nivel bajo no mostraba diferencia significativa con respecto al nivel alto. Para tomar la decisión del valor 10 como vecindario final se comparó los tiempos computacionales donde este obtuvo un menor tiempo de ejecución.

9.3.1.1.2 *Estructura de vecindario.* Estructura de vecindario de secuenciación entre operaciones adyacentes.

Los vecinos de una solución se generan intercambiando las posiciones de dos operaciones adyacentes, es decir, dos operaciones que pertenecen a la misma máquina, respetando las restricciones de precedencia.

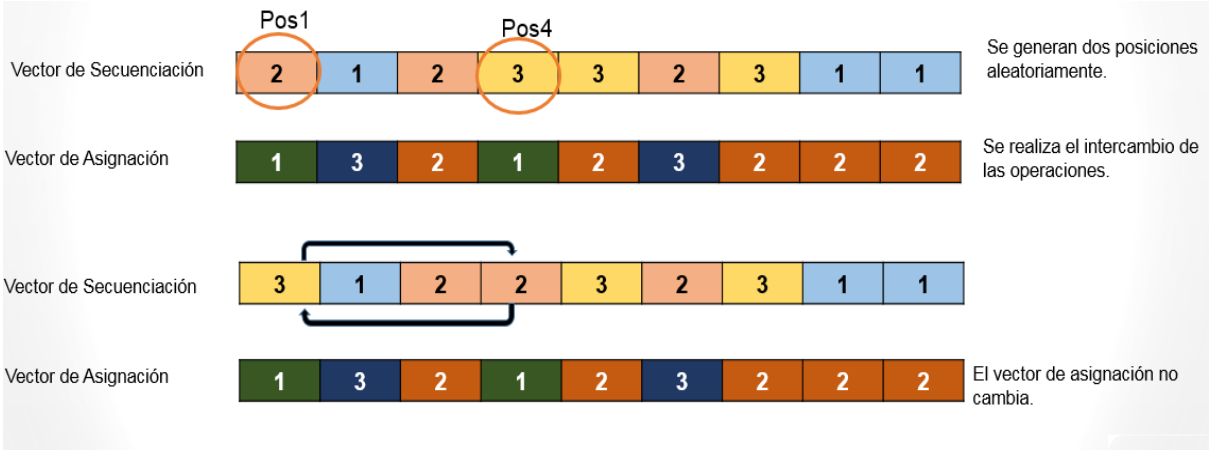

Figura 27. Ejemplo de la Estructura de vecindario entre operaciones adyacentes.

El algoritmo Genético se utiliza para mejorar las soluciones de la población inicial, esto se realiza mediante operadores de selección, cruce y mutación.

9.3.2 Criterio de selección. Para seleccionar los mejores individuos de la población se utilizó el criterio de selección de torneo determinístico de tamaño n. Aleatoriamente se escogen individuos de la población, los cuales participan en un torneo de tamaño n en el que compiten, evaluando que solución es mejor a otra basándose en el fitness de cada uno, en este caso es el Makespan y el ganador tendrá la opción de reproducirse, es decir, según el número de torneos ganados por un individuo será el número de descendientes. Esto se realiza hasta que el número de descendientes sea igual al tamaño de la población.

9.3.3 Operadores de cruce. Para generar nuevos individuos a través de las diferentes generaciones, se utilizaron operadores específicos de acuerdo al subproblema a solucionar. Para el caso de la asignación, se utilizó el operador de cruce de dos puntos porque según el estudio realizado por DeJong y Spears (1999), el cruce de dos puntos arroja mejores resultados en comparación con 3 o más puntos de cruce, los cuales disminuyen el rendimiento del algoritmo Genético al no conservar la información genética en los descendientes. Por otro lado, para la secuenciación se utilizó el operador POX porque está diseñado de tal manera que respeta las restricciones de precedencia de las operaciones de un mismo trabajo.

Para la selección de los operadores de cruce se realizó una ANOVA de un solo factor (ver Apéndice D) donde se evaluó de manera separada el rendimiento de cruce de dos puntos para el vector de asignación y el del cruce POX para el vector de secuenciación. Luego, se evaluó el comportamiento utilizando de manera conjunta el cruce de dos puntos y el cruce POX dando una probabilidad del 50% de que ocurra uno o el otro. Dando como resultado que al utilizarlos de manera conjunta pueden llegar a mejores soluciones comparado si se hicieran por separado.

La probabilidad de cruce es del 80%, fue determinada con base en la revisión de literatura autores como Azzouz *et al*. (2016) donde utilizaron algoritmo genético para resolver el FJSP-SDST en el cual arrojaba buenos resultados.

9.3.4 Operadores de mutación. Con el fin de obtener una mayor diversidad en el espacio de soluciones, se utilizó el operador PPS en el vector de secuenciación el cual respeta las restricciones de precedencia entre las operaciones de cada trabajo.

Utilizando este tipo de operador se evita un proceso de corrección que sería muy costoso en términos de tiempo computacional.

El operador PPS elige un gen del vector de manera aleatoria, luego el gen se desplaza a otra posición, mientras que la operación que este gen representa en su trabajo no cambia. La probabilidad de mutación es del 2% se tomó como referencia de la investigación realizada por Pezzella *et al*. (2008) donde utilizaban el mismo operador PPS.

9.4 Procedimiento para el Cálculo del Makespan

Para el cálculo del Makespan del Flexible Job Shop con tiempos de alistamiento dependientes de la secuencia. La asignación de los tiempos de alistamiento se realizó de acuerdo a la secuencia generada después de realizar el diagrama de Gantt en la evaluación del Makespan sin tiempos de alistamiento, dicha secuencia es un programa activo porque no es posible construir otra programación, a través de cambios en el orden de procesamiento en las máquinas, con al menos una operación terminando antes y ninguna operación terminando más tarde.

Puesto que si se utilizaba la secuencia de la *solución de referencia*, es decir, la que arroja el cromosoma, en la mayoría de los casos hay tiempos disponibles entre la finalización de una operación y el inicio de la siguiente operación en cada máquina, ocasionando el aumento del Makespan, a esto se le denomina un programa semi-activo porque las operaciones se programan exactamente en el mismo orden en que aparecen en la secuencia de cromosomas.

Por tal motivo se permitió que sí, una operación puede realizarse en el tiempo disponible se asignara en ese intervalo de tiempo, esto convierte el programa semi-activo en activo. Como

resultado, se produce un cambio en la secuencia de las operaciones de las máquinas, es decir, se genera una *solución ajustada*, la cual es utilizada para asignar los tiempos de alistamiento. Con el siguiente ejemplo se muestra el procedimiento que se realizó para calcular el Makespan con Tiempos de Alistamiento dependientes de la secuencia. La figura 28 es la *solución de referencia* al problema presentado anteriormente en la sección 8.2.4.

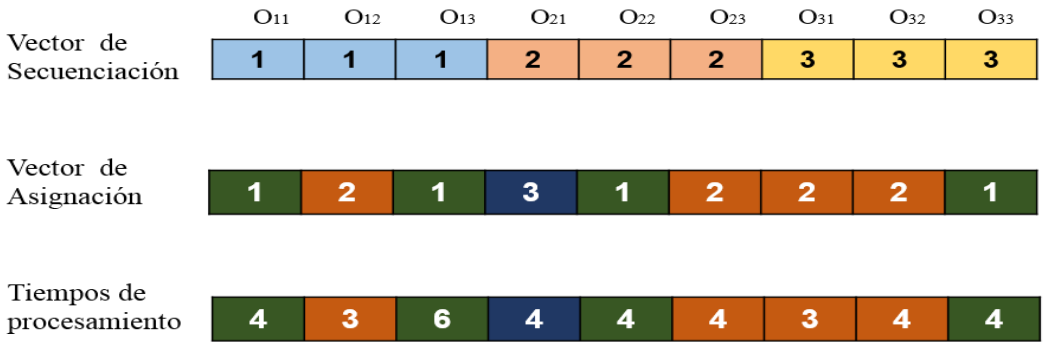

Figura 28. Solución de referencia al problema presentado en la sección 8.2.4

En la figura 29 se muestra el diagrama de Gantt correspondiente al cálculo del Makespan sin tiempos de alistamiento permitiendo tiempos ociosos y siguiendo estrictamente la secuencia de la *solución de referencia*.

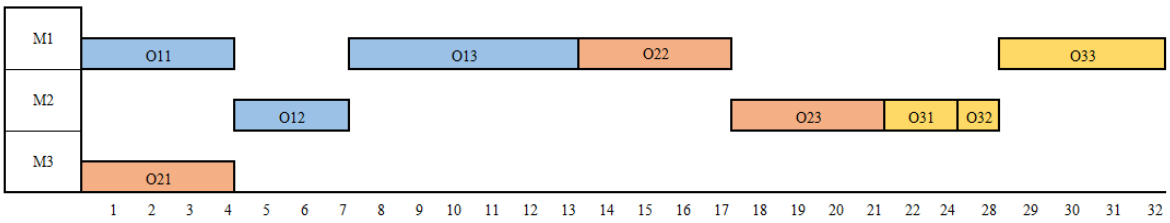

Figura 29. Diagrama de Gantt de la solución de referencia

Luego se calcula el Makespan con los tiempos de alistamiento, los cuales son dependientes de la secuencia, estos se suman al tiempo de procesamiento de la operación. En la figura 30 (a) se muestra la secuencia de las operaciones en cada máquina y los correspondientes tiempos de

alistamiento en cada una de ellas (b), cabe resaltar que el primer tiempo en cada máquina es el tiempo de alistamiento dummy, es decir, el tiempo de alistamiento que se requiere realizar para la primera operación en cada máquina. En la figura 31 se observa el diagrama de Gantt resultante luego de adicionar los tiempos de alistamiento.

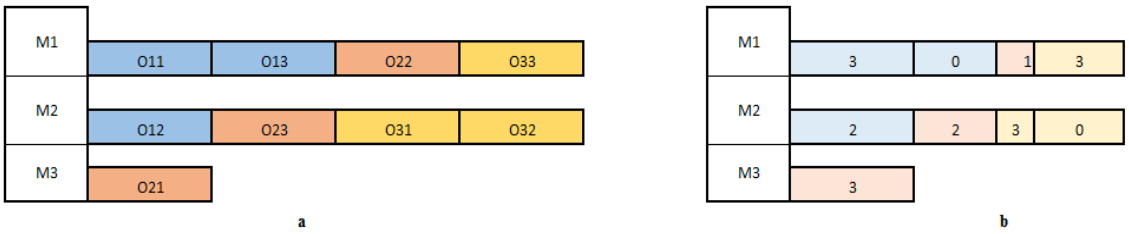

Figura 30. a) Secuencia de las operaciones en cada máquina b) Tiempos de alistamiento en cada máquina.

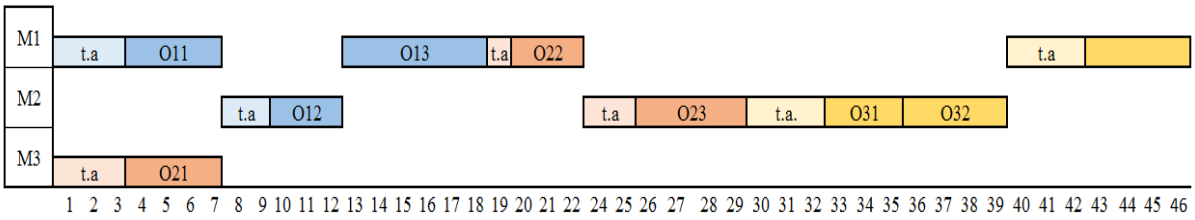

Figura 31. Diagrama de Gantt de la solución de referencia con Tiempos de Alistamiento Dependientes de la secuencia.

Sin embargo, se debe tener en cuenta la disponibilidad de las máquinas, por lo tanto se permiten adelantamientos de ciertas operaciones en las máquinas, es decir, se convierte el programa semi-activo en activo esto genera un cambio en la *solución de referencia* (ver figura 28) y se genera la *solución ajustada* (ver figura 33).

Para el ejemplo dado se observa un tiempo disponible de 4 unidades de tiempo (u.t) entre t (0) y la primera operación de la máquina 2 (ver figura 29), la primera operación del trabajo 3 se puede realizar allí porque su tiempo de procesamiento es menor al tiempo disponible

(ocioso), esto permite que el Makespan disminuya de 32 a 21 u.t (ver figura 32).Adicionalmente cambia la secuencia de las operaciones en la máquina 2.

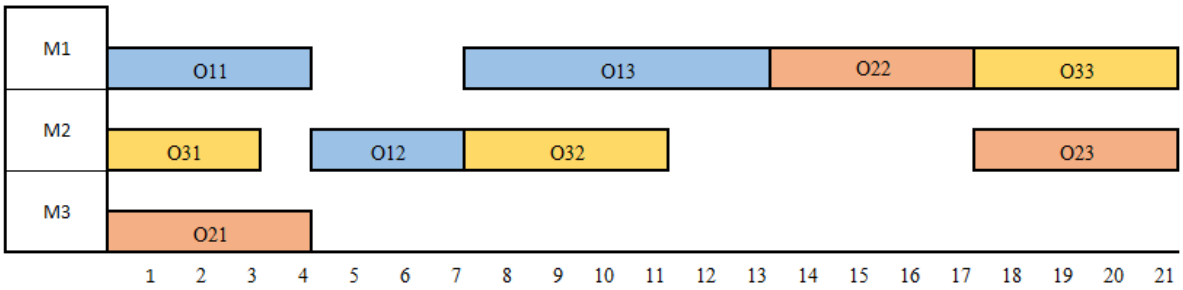

Figura 32. Cálculo del Makespan permitiendo ocupar tiempos ociosos

Debido a que se produce un cambio de secuencia en la máquina 2, se genera una *solución ajustada*.

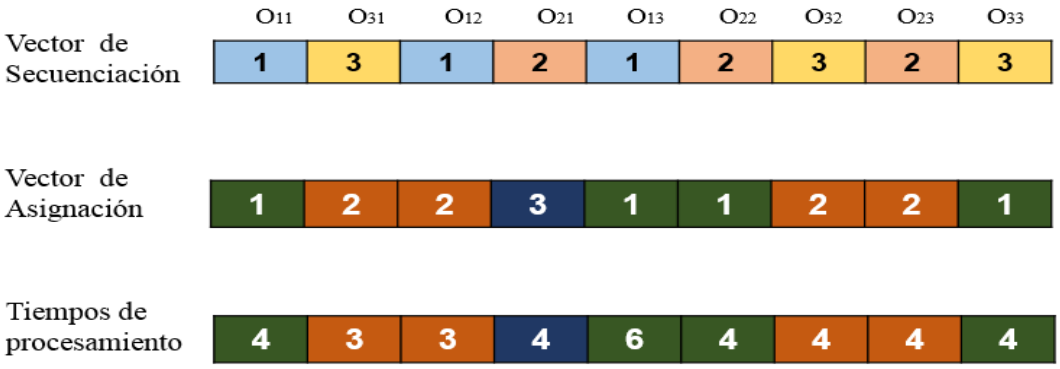

Figura 33. Solución ajustada

A partir de la solución ajustada se asignan los tiempos de alistamiento dependiendo de la secuencia de los trabajos en cada una de las máquinas. Para más detalle ver las figuras 34 y 35.

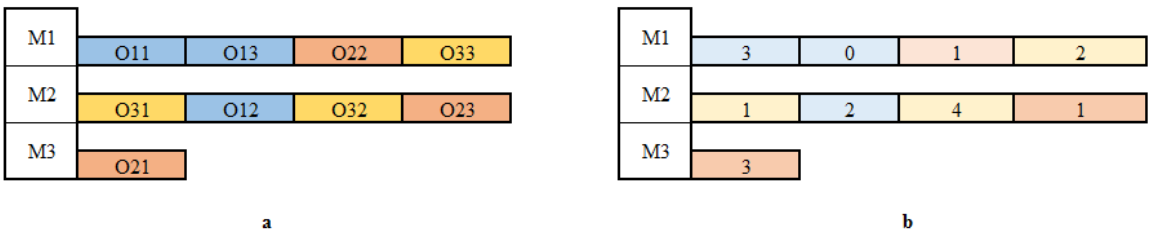

Figura 34. a) Secuencia de las operaciones en cada máquina b) Tiempos de alistamiento en cada máquina.

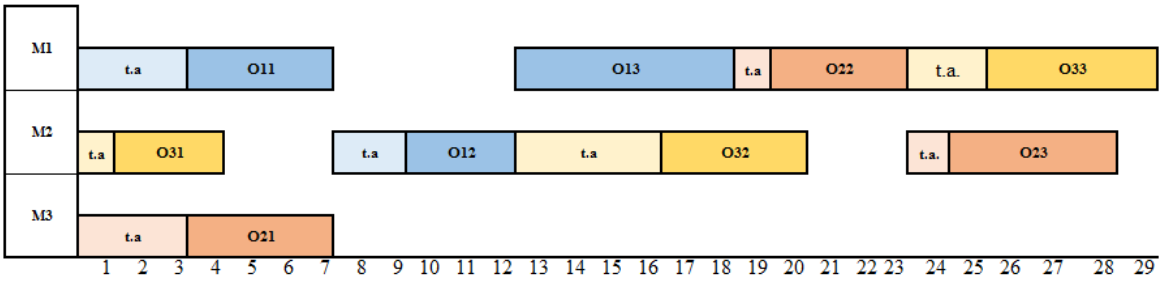

Figura 35. Diagrama de Gantt asignando los tiempos de alistamiento

Finalmente el Makespan es 29 u.t. asignándole los tiempos de alistamiento.

9.5 Diagrama de flujo del Algoritmo Híbrido (AG-SA)

La descripción de la estructura del Algoritmo Híbrido genético es de la siguiente manera

1. Codificación: El tipo de codificación utilizada es la representación de dos vectores.
2. Población inicial: La población inicial es obtenida mediante la combinación del Recocido Simulado (20%) y aleatorio (80%).
3. Evaluación fitness: El Makespan es calculado para cada cromosoma de cada generación, el procedimiento es descrito en la sección 9.4.
4. Selección: En cada generación se seleccionan los individuos por el método de selección por torneo.
5. Cruce: La nueva generación se obtiene cambiando la asignación de las máquinas, mediante un operador de cruce de dos puntos y se cambia la secuenciación mediante el operador POX.
6. Mutación: Para generar diversidad en la población se utilizó el operador PPS.
7. Criterio de parada: Numero de generaciones.

En la figura 36 se evidencia el Diagrama de flujo del Algoritmo Híbrido Genético y en la figura 37 el diagrama de flujo del Recocido Simulado.

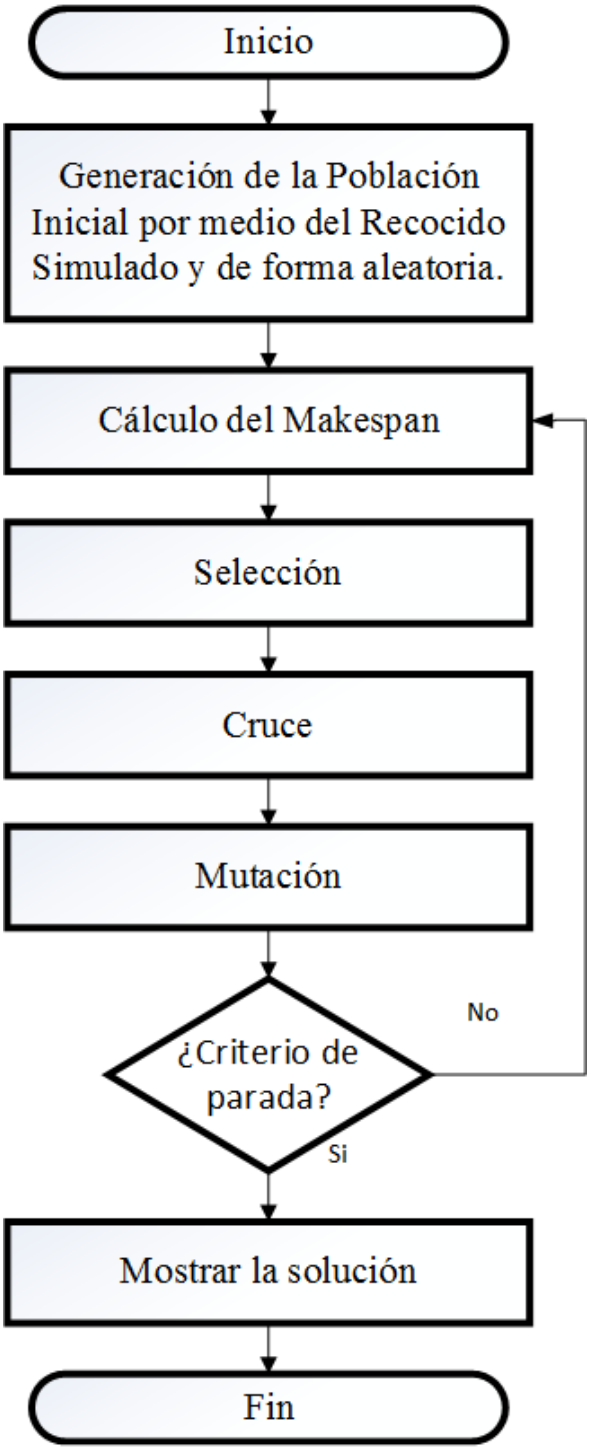

Figura 36. Diagrama de Flujo del Algoritmo Híbrido Genético.

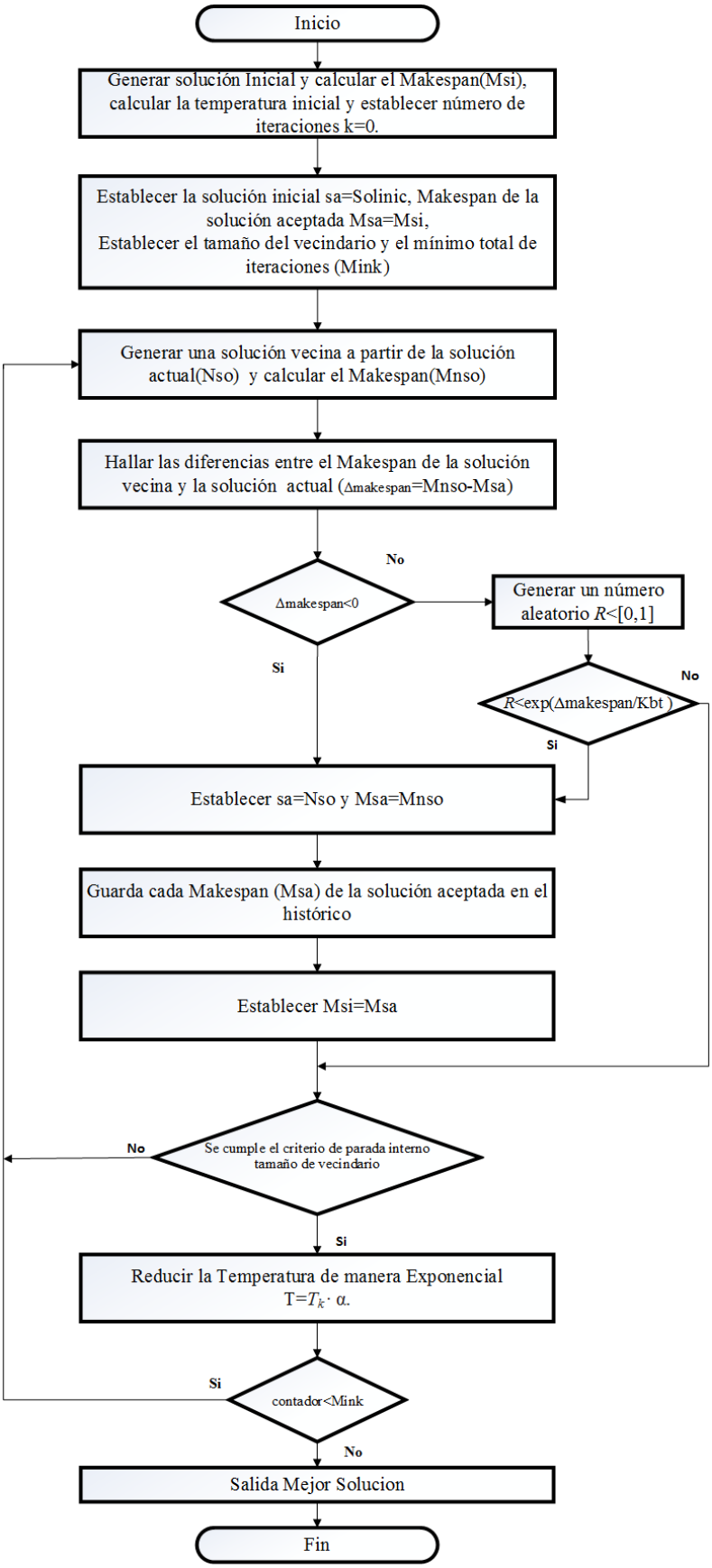

Figura 37. Diagrama de Flujo del Recocido Simulado

10. Validación del Algoritmo Propuesto

Para analizar el comportamiento del Algoritmo Híbrido Genético (AHG) en el Flexible Job Shop se realizó un diseño de experimentos 2^k (ver Apéndices: F y L). El algoritmo se desarrolló en Matlab ® versión 2017a y el diseño factorial se ejecutó en el software estadístico Minitab ® 18. Las instancias utilizadas fueron las de Bagheri y Zandieh (2011), Ver Apéndice O.

Tabla 13.

Instancias de Bagheri y Zandieh (2011)

	nxnixm	**Nombre instancia**
Clase1	10x5x5	AEB01-AEB05
Clase2	15x5x8	AEB06-AEB10
Clase3	10x10x5	AEB11-AEB15
Clase4	15x10x10	AEB16-AEB20

El diseño experimental fue un 2^3, con 5 réplicas, los factores y niveles se muestran en la tabla 14.

Tabla 14.

Factores y niveles del diseño experimental 2

Factores	**Niveles**	**Valores**
	Bajo(-)	Alto(+)
Pob: Población	10	100
Vec: Vecindario	10	50
Gen: Generaciones	100	400

A continuación se presenta el Análisis de Varianza de una instancia de cada clase, que representa el comportamiento en común de cada una de ellas.

10.1 Análisis de Varianza para la instancia AEB05

Tabla 15.

Análisis de Varianza para la instancia AEB05

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	602406	86058	23,48	0,000
Lineal	3	584675	194892	53,18	0,000
Pob	1	583464	583464	159,20	0,000
Vec	1	119	119	0,03	0,858
Gen	1	1092	1092	0,30	0,589
Interacciones de 2 términos	3	11455	3818	1,04	0,387
Pob*Vec	1	207	207	0,06	0,814
Pob*Gen	1	10465	10465	2,86	0,101
Vec*Gen	1	783	783	0,21	0,647
Interacciones de 3 términos	1	6275	6275	1,71	0,200
Pob*Vec*Gen	1	6275	6275	1,71	0,200
Error	32	117277	3665		
Total	39	719683			

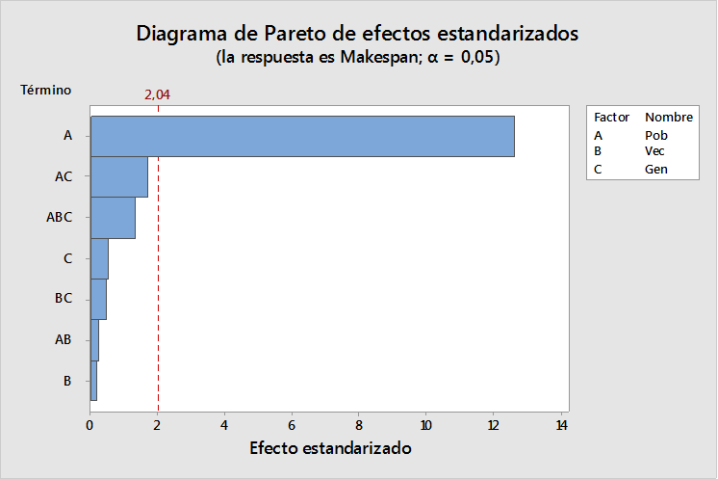

Figura 38. Diagrama de Pareto de efectos estandarizados de la instancia AEB05

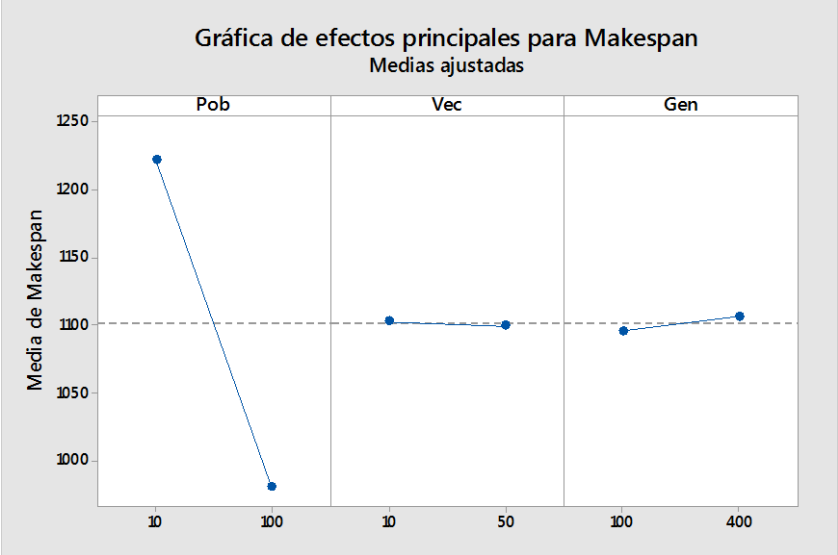

Figura 39. Gráfica de Efectos principales para la instancia AEB05

De la tabla del Análisis de Varianza de la instancia AEB05 se evidencia que el efecto principal está asociado al factor, tamaño de la población con un valor p 0,000. Para verificar los resultados del Análisis de Varianza se muestra el Diagrama de Pareto de efectos estandarizados.

Analizando la gráfica de efectos principales para Makespan, se puede evidenciar que los niveles de factores que minimizan el Makespan son: tamaño de la población inicial 100, número de generaciones 100, dado que el tamaño de vecindario no es significativo se toma el valor 10, porque con este se consume menos tiempo computacional.

10.2 Análisis de Varianza para la instancia AEB10

Tabla 16.

Análisis de Varianza para la instancia AEB10

Fuente	GL	SC Ajust.	MC Ajust.		Valor F	Valor p
Modelo	7	788726	112675		35,80	0,000
Lineal	3	781510	260503		82,77	0,000
Pob	1	778131	778131		247,24	0,000
Vec	1	403	403		0,13	0,723
Gen	1	2976	2976		0,95	0,338
Interacciones de 2 términos	3	5270	1757		0,56	0,646
Pob*Vec	1	2941	2941		0,93	0,341
Pob*Gen	1	3	3		0,00	0,975
Vec*Gen	1	2326	2326		0,74	0,396
Interacciones de 3 términos	1	1946	1946		0,62	0,437
Pob*Vec*Gen	1	1946	1946		0,62	0,437
Error	32	100712	3147			
Total	39	889438				

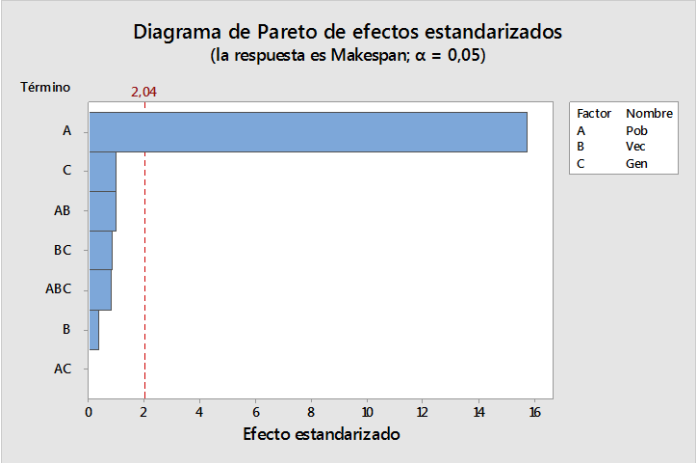

Figura 40. Diagrama de Pareto de efectos estandarizados de la instancia AEB10

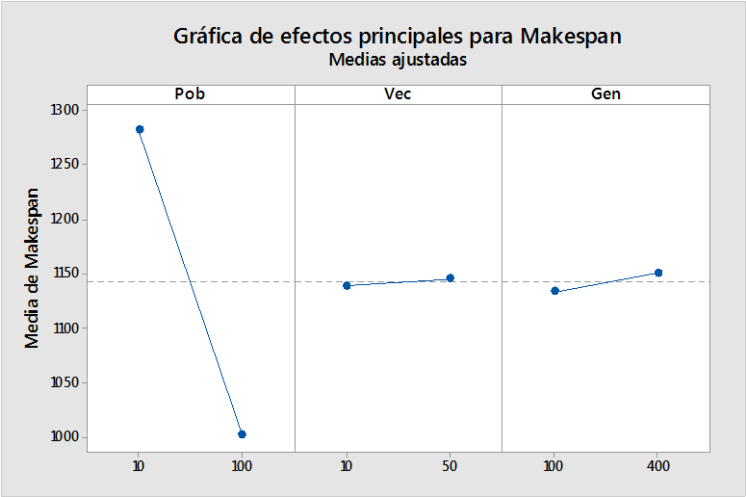

Figura 41. Gráfica de Efectos principales para la instancia AEB10

Para esta instancia el efecto principal es el tamaño de la población con un valor p 0,000. Por otro lado los niveles de los factores: tamaño de vecindario y número de generaciones no influyen de manera significativa en el Makespan.

Analizando la gráfica de efectos principales para Makespan, se puede evidenciar que los niveles de factores que minimizan el Makespan son: tamaño de la población inicial 100, número de generaciones 100, tamaño de vecindario 10, dado que los dos últimos factores no

tienen efecto significativo se toman los niveles bajos de cada factor, porque se consume menos tiempo computacional.

10.3 Análisis de Varianza para la instancia AEB11

Tabla 17.

Análisis de Varianza para la instancia AEB11

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	1410282	201469	40,75	0,000
Lineal	3	1388827	462942	93,65	0,000
Pob	**1**	1374556	1374556	278,06	0,000
Vec	1	5832	5832	1,18	0,286
Gen	1	8439	8439	1,71	0,201
Interacciones de 2 términos	3	16940	5647	1,14	0,347
Pob*Vec	1	2205	2205	0,45	0,509
Pob*Gen	1	14707	14707	2,98	0,094
Vec*Gen	1	27	27	0,01	0,941
Interacciones de 3 términos	1	4516	4516	0,91	0,346
Pob*Vec*Gen	1	4516	4516	0,91	0,346
Error	32	158190	4943		
Total	39	1568472			

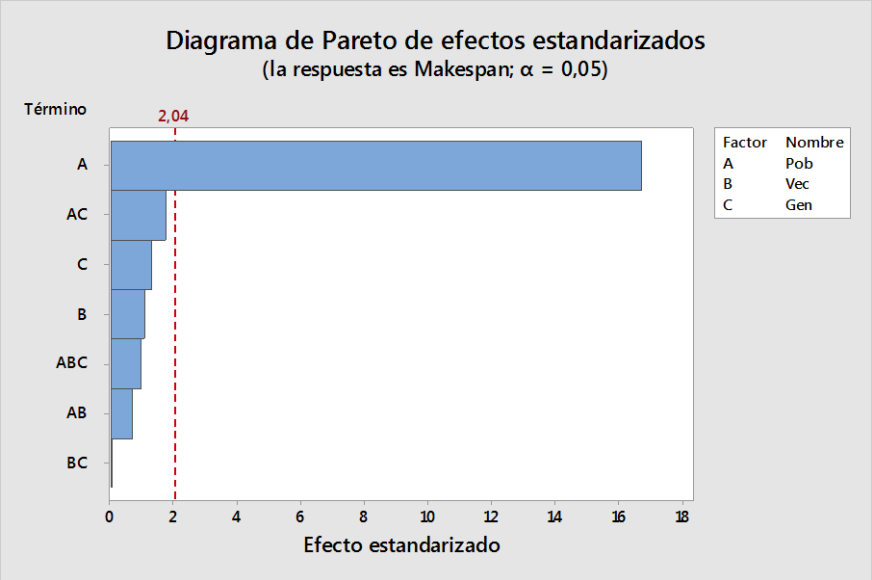

Figura 42. Diagrama de Pareto de efectos estandarizados de la instancia AEB11

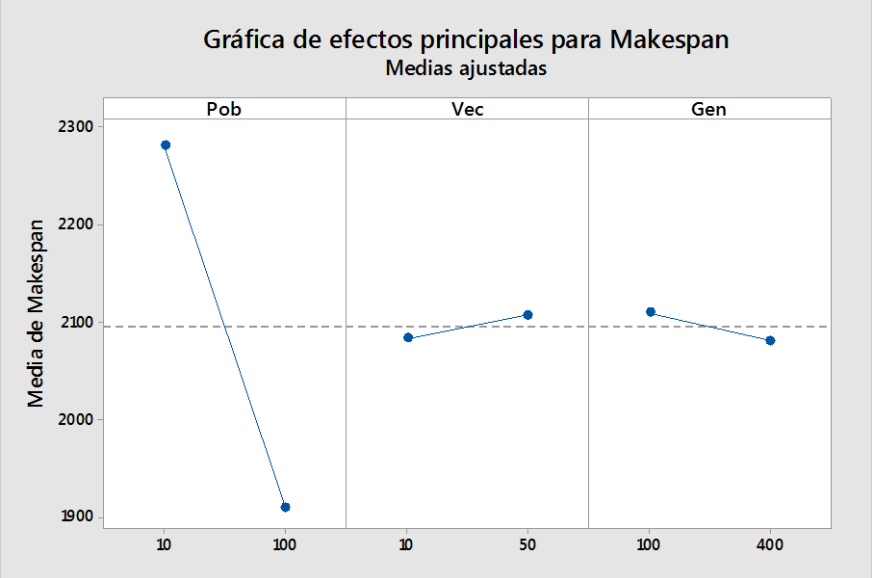

Figura 43. Gráfica de Efectos principales para la instancia AEB11

Para esta instancia el efecto principal es el tamaño de la población con un valor p 0,000. Por otro lado, los niveles de los factores: tamaño de vecindario y número de generaciones no influyen de manera significativa en el Makespan.

Analizando la gráfica de efectos principales para Makespan, se puede evidenciar que los niveles de factores que minimizan el Makespan son: tamaño de la población inicial 100,

número de generaciones 100, tamaño de vecindario 10, dado que los dos últimos factores no tienen efecto significativo se toman los niveles bajos de cada factor, porque se consume menos tiempo computacional. Para las instancias AEB11-AEB12-AEB13-AEB14 el comportamiento es similar, a diferencia de la instancia AEB15, por esta razón se menciona a continuación.

10.4 Análisis de Varianza para la instancia AEB15

Tabla 18.

Análisis de Varianza para la instancia AEB15

Fuente	**GL**	**SC Ajust.**	**MC Ajust.**	**Valor F**	**Valor p**
Modelo	7	1878349	268336	41,88	0,000
Lineal	3	1751131	583710	91,10	0,000
Pob	1	1748076	1748076	272,82	0,000
Vec	1	2190	2190	0,34	0,563
Gen	1	865	865	0,13	0,716
Interacciones de 2 términos	3	127216	42405	6,62	0,001
Pob*Vec	1	27773	27773	4,33	0,045
Pob*Gen	1	36240	36240	5,66	0,024
Vec*Gen	1	63202	63202	9,86	0,004
Interacciones de 3 términos	1	2	2	0,00	0,987
Pob*Vec*Gen	1	2	2	0,00	0,987
Error	32	205037	6407		
Total	39	2083386			

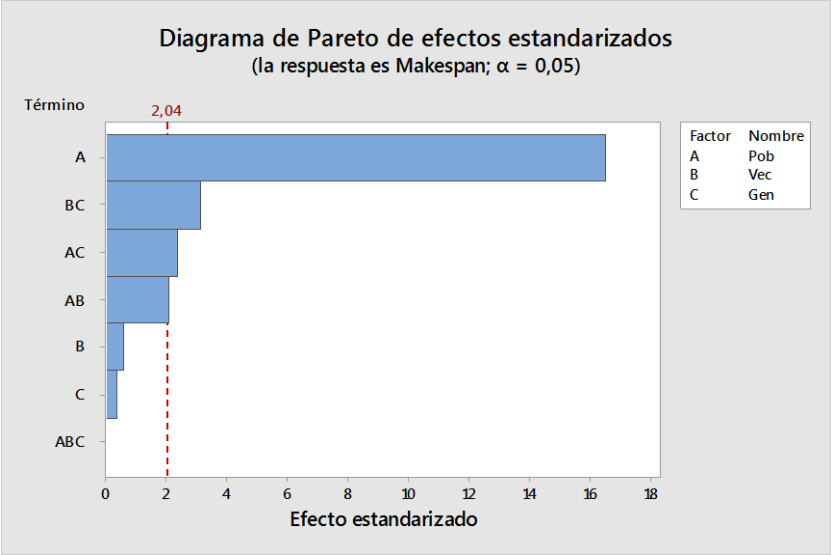

Figura 44. Diagrama de Pareto de efectos estandarizados de la instancia AEB15

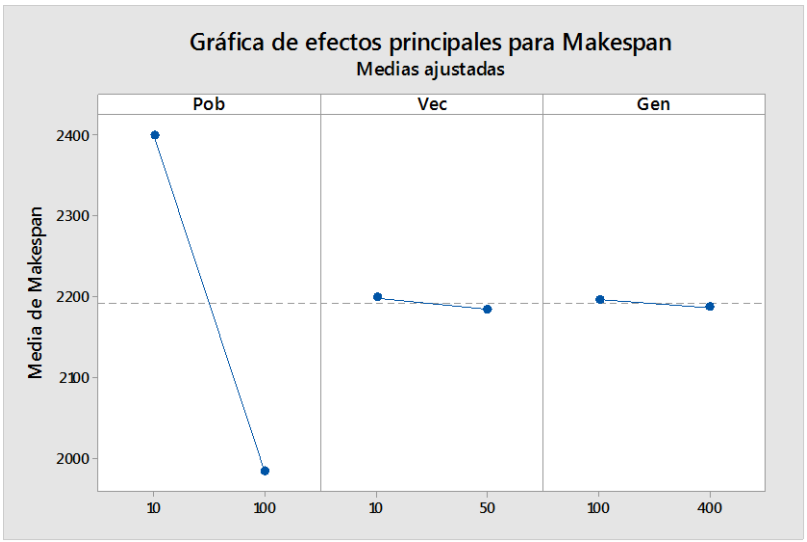

Figura 45. Gráfica de Efectos principales para la instancia AEB15

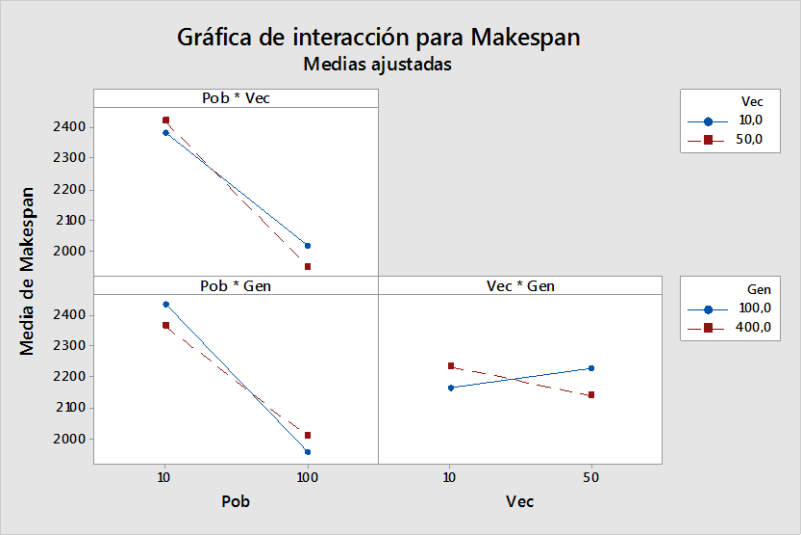

Figura 46. Gráfica de interacción para Makespan de la instancia AEB15

Para esta instancia tienen efecto significativo el tamaño de la población y la interacción entre los factores: tamaño de vecindario y número de generaciones (Vec*Gen) con un valor p de 0,004, el tamaño de la población y número de generaciones (Pob*Gen) con un valor p de 0,024. En la gráfica de interacción para Makespan se puede evidenciar que para Vec*Gen con 100 generaciones y 10 vecinos contra 400 generaciones y 50 vecinos tienen efectos similares.

En la interacción Pob*Gen con 100 tamaño de población y 100 generaciones contra 100 tamaño de población y 400 generaciones tienen efectos similares.

Teniendo en cuenta el efecto del número de población y la interacciones entre dos factores: Vec*Gen y Pob*Gen, la mejor combinación es: 100 tamaño de población, 10 tamaño de vecindario y 100 generaciones por menor tiempo computacional.

10.5 Análisis de Varianza para la instancia AEB20

Tabla 19.

Análisis de Varianza para la instancia AEB20

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	1552771	221824	34,77	0,000
Lineal	3	1546818	515606	80,81	0,000
Pob	1	1538208	1538208	241,09	0,000
Vec	1	1588	1588	0,25	0,621
Gen	1	7023	7023	1,10	0,302
Interacciones de 2 términos	3	5195	1732	0,27	0,846
Pob*Vec	1	202	202	0,03	0,860
Pob*Gen	1	2496	2496	0,39	0,536
Vec*Gen	1	2496	2496	0,39	0,536
Interacciones de 3 términos	1	757	757	0,12	0,733
Pob*Vec*Gen	1	757	757	0,12	0,733
Error	32	204165	6380		
Total	39	1756936			

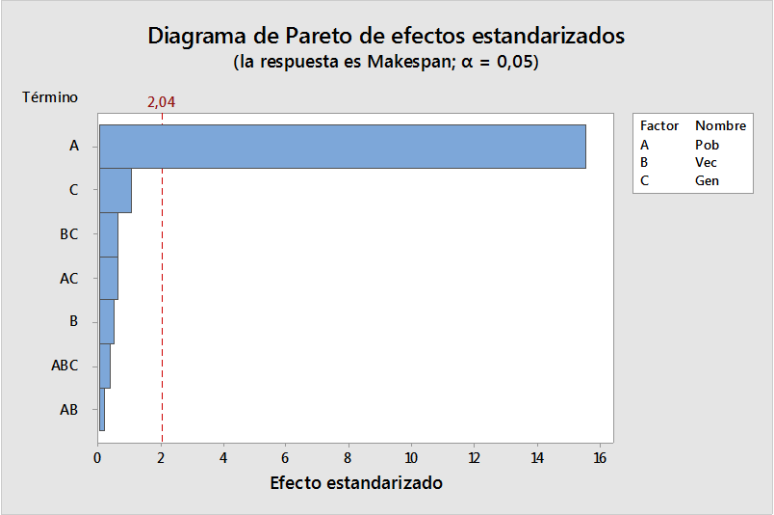

Figura 47. Diagrama de Pareto de efectos estandarizados de la instancia AEB20

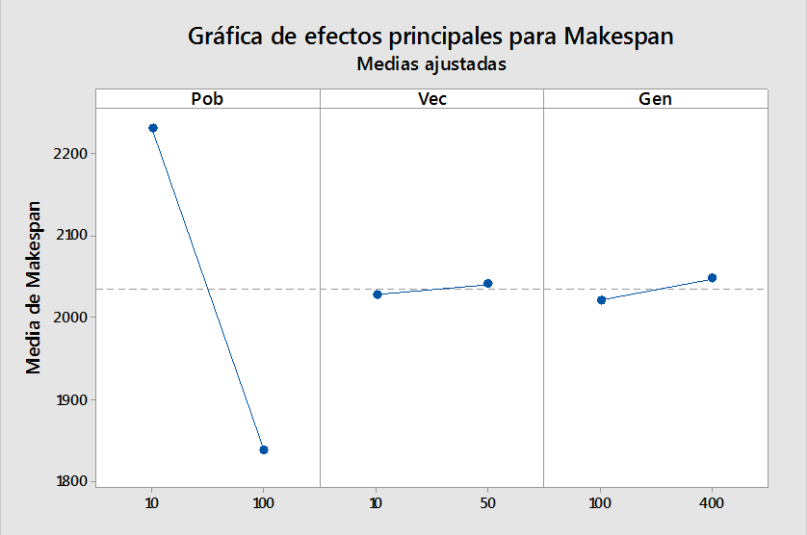

Figura 48 . Gráfica de Efectos principales para la instancia AEB20

De la tabla del Análisis de Varianza de la instancia AEB20 se evidencia que el efecto principal está asociado al factor, tamaño de la población con un valor p 0,000. Para verificar los resultados del Análisis de Varianza se muestra el Diagrama de Pareto de efectos estandarizados

Analizando la gráfica de efectos principales para Makespan, se puede evidenciar que los niveles de factores que minimizan el Makespan son: tamaño de la población inicial 100,

número de generaciones 100, tamaño de vecindario 10, dado que los dos últimos factores no tienen efecto significativo se toman los niveles bajos de cada factor. Sin embargo, con el valor de 100 en las generaciones el tiempo computacional es demasiado alto, por lo que se decidió comparar si era significativo el GAP cuando el algoritmo tenía 100 tamaño de población, 10 tamaño de vecindario y 100 número de generaciones comparado con 100 tamaño de población, 10 tamaño de vecindario y 50 número de generaciones.

Tabla 20.

Comparación del Makespan 100 generaciones vs 50 generaciones para las instancias de tamaño 15x10x10

Instancia	**M.M(100)**	**T.C. (100)**	**M.M (50)**	**T.C(50)**	**GAP**	**Diferencia T.C**
AEB16	1742	4500,50	1727	2738,27	0.86%	64.35%
AEB17	1690	4125,72	1744	2537,37	3.19%	62.59%
AEB18	1697	3807,96	1745	2993,06	2.82%	27.22%
AEB19	1730	5364,23	1820	2939,98	5.20%	82.45%
AEB20	1730	4877,11	1723	2978,43	0.40%	63.74%

Nota: M.M (100) Mejor Makespan de 10 iteraciones del algoritmo con 100 generaciones, T.C (100) Tiempo computacional de 10 iteraciones del algoritmo con 100 generaciones, M.M (100) Mejor Makespan de 10 iteraciones del algoritmo con 50 generaciones, T.C (50) Tiempo computacional de 10 iteraciones del algoritmo con 50 generaciones

De los resultados de la tabla 21 se puede evidenciar que el GAP para el Makespan cuando el algoritmo tiene 100 generaciones y 50 generaciones no es significativo, porque el porcentaje más alto es de 5.20% pero en cuestión de tiempo computacional la diferencia es notable desde

un 27.22% hasta 82.45%, por lo tanto se escoge la opción de 50 generaciones porque llega a mejores respuestas en un tiempo computacional menor (Ver Apéndice M).

Parámetros utilizados en el Algoritmo Híbrido Genético para cada clase de instancia

Tabla 21.

Parámetros utilizados en el Algoritmo Híbrido Genético para cada clase de instancia.

Instancia	**nxnixm**	**Parámetros SA**		**Parámetros AG**					
		α	Vec	Psa	Pal	Pob	Pc	Pm	Gen
AEB01-AEB05	10x5x5	0.5	10	0.2	0.8	100	0.8	0.02	100
AEB06-AEB10	15x5x8	0.5	10	0.2	0.8	100	0.8	0.02	100
AEB11-AEB15	10x10x5	0.5	10	0.2	0.8	100	0.8	0.02	100
AEB16-AEB20	15x10x10	0.5	10	0.2	0.8	100	0.8	0.02	50

Nota: α: Factor de enfriamiento, Vec: Tamaño del vecindario, Psa: Porcentaje de recocido simulado en la población inicial, Pal: Porcentaje de recocido simulado en la población inicial, Pob: Tamaño de población, Pc: Probabilidad de cruce, Pm: probabilidad de mutación, Gen: Generaciones.

11. Resultados

En la tabla 23 se muestra la mejor respuesta encontrada (Makespan) después de 10 corridas y el tiempo computacional empleado en cada instancia (Ver Apéndice N). Sin embargo, no es posible comparar en términos de tiempo computacional porque los autores Azzouz et al. (2017) no lo presentan en su trabajo de investigación.

Tabla 22.

Tiempo computacional en segundos empleado en conseguir la mejor respuesta para el algoritmo propuesto

Instancia	Tamaño	Makespan	TC [s]
AEB01		957	275,09
AEB02		891	194,65
AEB03	10x5x5	841	243,00
AEB04		931	314,26
AEB05		917	199,05
AEB06		981	753,19
AEB07		979	837,18
AEB08	15x5x8	972	762,55
AEB09		982	879,23
AEB10		968	730,59
AEB11		1826	1110,56
AEB12		1849	1565,61
AEB13	10x10x5	1882	993,37
AEB14		1874	1056,92
AEB15		1922	1323,85
AEB16		1727	2738,27
AEB17		1744	2537,47
AEB18	15x10x10	1745	3249,53
AEB19		1820	2939,98
AEB20		1723	2978,43

En la tabla 23 se muestran los resultados obtenidos de los algoritmos: Búsqueda Local Iterativa (ILS), Búsqueda de vecindario Variable (VNS), Algoritmo Genético (GA), Algoritmo Híbrido Genético (HGA) este algoritmo es la hibridación entre GA y VNS. Adicionalmente se compara con el algoritmo Híbrido SAHA que combina el Algoritmo Genético (GA) y la Búsqueda Local Iterativa (ILS), cabe resaltar que se compara con dos tipos de algoritmo SAHA: SAHA (EE) y SAHA(AS), el último utiliza una estrategia auto-adaptativa basada en:

(1) la especificidad actual del espacio de búsqueda, (2) los resultados anteriores de algoritmos ya aplicados (GA e ILS) y (3) sus ajustes de los parámetros asociados. Los anteriores algoritmos también fueron corridos 10 veces.

Tabla 23.

Comparación de los resultados obtenidos por el AHG propuesto vs ILS, VNS, GA, HGA, SAHA (EE), SAHA (AS).

Instancia	Tamaño	ILS	VNS	GA	HGA	SAHA(EE)	SAHA(AS)
AEB01	10x5x5	1183	1157	1157	865	912	**846**
AEB02		1118	1023	981	818	881	**807**
AEB03		1062	1028	886	762	810	**753**
AEB04		1125	1031	917	840	854	**807**
AEB05		1110	1085	961	898	907	**902**
AEB06	15x5x8	1121	1018	959	870	836	**831**
AEB07		1233	1025	884	896	870	**820**
AEB08		1288	1096	927	886	853	**810**
AEB09		1262	1060	871	870	857	**818**
AEB10		1276	986	986	875	851	**825**
AEB11	10x10x5	2416	2132	1771	1769	1765	**1720**
AEB12		2464	2062	1983	1742	1843	**1708**
AEB13		2509	2036	1910	1740	1769	**1658**
AEB14		2576	2213	1940	1793	1896	**1743**
AEB15		2575	2333	1894	1810	1783	**1698**
AEB16	15x10x10	2380	1892	1335	1321	1319	**1319**
AEB17		2574	1946	1245	1240	1245	**1240**
AEB18		2380	2026	1282	1268	1268	**1268**
AEB19		2196	2019	1208	**1199**	1208	1201
AEB20		2413	1943	1307	1307	1307	**1307**

Instancia	Tamaño	ILS	VNS	GA	HGA	SAHA(EE)	SAHA(AS)
AEB01	10x5x5	23.61%	20.89%	13.06%	10.63%	4.93%	13,12%
AEB02		25.47%	14.81%	10.10%	8.92%	1.13%	10,40%
AEB03		26.27%	22.23%	5.35%	10.36%	3.82%	11,68%
AEB04		20.83%	10.74%	1.52%	10.83%	9.01%	15,36%
AEB05		21.04%	18.32%	4.80%	2.11%	1.10%	1,66%
AEB06	15x5x8	14.27%	3.77%	2.29%	12.75%	17.34%	18,05%
AEB07		25.94%	4.69%	10.74%	9.26%	12.52%	19,39%
AEB08		32.92%	13.10%	4.53%	9.36%	13.59%	19,62%
AEB09		28.51%	7.94%	12.74%	12.87%	14.58%	20,04%
AEB10		31.81%	1.86%	8.39%	10.62%	13.74%	17,33%
AEB11	10x10x5	32.31%	16.75%	3.10%	3.22%	3.45%	6,16%
AEB12		33.26%	11.51%	7.24%	6.14%	0.32%	8,25%
AEB13		33.31%	8.18%	1.48%	8.16%	6.38%	13,51%
AEB14		37.46%	18.09%	3.52%	4.51%	1.17%	7,51%
AEB15		33.97%	21.38%	1.47%	6.18%	7.79%	13,19%
AEB16	15x10x10	37.81%	9.55%	29.36%	30.73%	30.93%	30,93%
AEB17		47.59%	11.58%	40.08%	40.64%	40.08%	40,64%
AEB18		36.38%	16.10%	36.11%	37.61%	37.61%	37,61%
AEB19		20.66%	10.93%	50.66%	51.79%	50.66%	51,54%
AEB20		40.04%	12.76%	31.82%	31.82%	31.82%	31,82%

Figura 49. Comparación de los resultados obtenidos por el AHG propuesto vs ILS, VNS, GA, HGA, SAHA (EE), SAHA (AS)

Nota: Las celdas en azul son los resultados que el algoritmo propuesto (AHG) mejora a los demás algoritmos, las celdas en amarillo son en las que el algoritmo está cerca a la mejor respuesta entre un 0,32% hasta un 15%, las celdas en naranja son las que el algoritmo está entre un 15% hasta un 51.79% de la mejor respuesta encontrada.

De los resultados obtenidos, el algoritmo propuesto para las instancias de tamaño 10 trabajos, 5 operaciones por cada trabajo y 5 máquinas tiene un porcentaje de diferencia con la mejor solución obtenida que está entre el 1,66% y 15,36%.

Se observa que cuando se aumenta el número de trabajos de 10 a 15 y el número de máquinas de 5 a 8, el porcentaje de diferencia aumenta hasta en un 20% respecto a la mejor solución.

Por otro lado, cuando se deja el mismo tamaño de trabajos 10 y máquinas 5 pero aumentando las operaciones por trabajo de 5 a 10, el porcentaje de diferencia está entre el 6,16% y 13,51%.Para las instancias de mayor tamaño: 15 trabajos, 10 operaciones por cada trabajo y 10 máquinas el porcentaje de diferencia está entre 30,93% y 51,54%.

En la tabla 24 se muestra el porcentaje del total de las instancias donde el algoritmo propuesto (AHG) mejoralos resultados obtenidos y el GAP, respecto a los algoritmos propuestos por Azzouz, Ennigrou, y Ben Said (2017).

Tabla 24.

Comparación Algoritmo Propuesto vs algoritmos de Azzouz et al. (2017).

		GAP		
Algoritmo	**Mejora**	**0,32% - 15%**	**15%- 20.04%**	**29.36% -51.79%**
Búsqueda Local Iterativa (ILS)	(20) 100%	_	_	_
Búsqueda de Vecindario Variable (VNS)	(20) 100%	_	_	_
Algoritmo Genético (GA)	(7) 35%	(8) 40%	_	(5) 25%
Algoritmo Híbrido Genético(HGA)-GA y VNS	_	(15) 75%	_	(5) 25%
Algoritmo Híbrido Auto adaptativo (SAHA(EE))- GA y ILS	(1) 5%	(13) 65%	(1) 5%	(5) 25%
Algoritmo Híbrido Auto adaptativo SAHA(AS)- GA y ILS Con estrategia auto adaptativa.	_	(9) 45%	(6) 30%	(5) 25%

12. Conclusiones

Según la revisión de literatura pocos autores han abordado el Flexible Job Shop con tiempos de Alistamiento dependientes de la secuencia (FJSP-SDST), lo cual permite que este proyecto sea una referente para futuras investigaciones en esta temática.

Como resultado del análisis de medias presentado en el Apéndice C se observó que utilizar el Recocido Simulado para generar parte de la población inicial del Algoritmo Genético mejora el desempeño del mismo en los cuatro tamaños de instancias, siendo una buena alternativa para la solución de problemas complejos.

El algoritmo propuesto (AHG) mejora en el 100% de las instancias los resultados obtenidos por los algoritmos ILS, VNS; un 35% y 5% al GA y SAHA (EE) respectivamente propuestos por los autores Azzouz, Ennigrou, y Ben Said, lo cual demuestra que el algoritmo mejoró los resultados obtenidos por otros algoritmos

Comparando el algoritmo propuesto (AHG) con los algoritmos GA, HGA, SAHA (EE), SAHA(AS) los resultados muestran que el 40%, 75%, 65%, 45% de sus resultados respectivamente está entre un 0,32% hasta un 15% de los mejores resultados encontrados por cada uno de ellos. Sin embargo, para las instancias de mayor tamaño el algoritmo presenta dificultades para encontrar resultados cercanos a los mejores encontrados por el GA, HGA, SAHA (EE), SAHA(AS) estando entre un 29,36% hasta un 51.79% de la mejor respuesta.

En el análisis del diseño experimental para todas las clases de instancias se concluye que, el factor tamaño de la población tiene efecto significativo en la variable Makespan, porque al

aumentar el tamaño de la población se pueden obtener mejores soluciones debido a una mayor exploración en el espacio de soluciones.

13. Recomendaciones

Para futuras investigaciones considerar restricciones adicionales tales como: tiempos de transporte, averías en las máquinas, interrupciones, ventanas de tiempos de procesamiento, entre otras de tal manera que se pueda aproximar a entornos reales.

Al momento de plantear un algoritmo Híbrido se debe tener en cuenta el tiempo de procesamiento de los algoritmos por separado y el papel que va a ocupar dentro de la estrategia de solución, esto es importante debido a que, permite explotar las capacidades de cada uno de manera estratégica con tiempos computacionales razonables para que el funcionamiento de uno no afecte la calidad de soluciones en conjunto.

Emplear una estrategia auto-adaptativa en el Algoritmo Hibrido Genético para resolver el problema, esta permite detectar la próxima dirección de búsqueda prometedora y mantener el equilibrio entre exploración y explotación, como la proponen varios autores de la literatura.

Motivar a los estudiantes de Ingeniería Industrial para que adquieran conocimientos en diferentes lenguajes de programación con el fin de incentivar el desarrollo de este tipo de proyectos.

Profundizar en el estudio de heurísticas y Metaheurísticas para diversificar las herramientas disponibles que permitan afrontar los diferentes problemas de investigación de operaciones presentes en la Ingeniería Industrial.

Referencias Bibliográficas

Abdelmaguid , T. (2010). Representations in Genetic Algorithm for the Job Shop Scheduling Problem: A Computational Study. *Journal of Software Engineering and Applications, 3*, 1155-1162 . doi:10.4236/jsea.2010.312135

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research , 246*(2), 345-378. doi:https://doi.org/10.1016/j.ejor.2015.04.004

Alvarez Valdes, R., Fuertes, A., Tamarit, J., Gimenez, G., y Ramos, R. (2005). A heuristic to schedule flexible job shop in a glass factory. *European Journal of Operational Research, 165*, 525-534. doi:https://doi.org/10.1016/j.ejor.2004.04.020

Azzouz, A., Ennigrou, M., y Ben Said , L. (2016). A hybrid algorithm for flexible job-shop scheduling problem with setup times. *International Journal of Production Management and Engineering, 5*(1), 23-30. doi:https://doi.org/10.4995/ijpme.2017.6618

Azzouz, A., Ennigrou, M., y Ben Said, L. (2016). Flexible Job-shop Scheduling Problem with Sequence-dependent Setup times using genetic algorithm. *18th International Conference on Enterprise Information Systems (ICEIS 2016)*, (pp. 47-53). Rome. doi:10.5220/0005821900470053

Azzouz, A., Ennigrou, M., y Ben Said, L. (2016). Flexible Job-shop Scheduling Problem with Sequence-dependent Setup Times using Genetic Algorithm. *18th International Conference on Enterprise Information Systems (ICEIS 2016), 2*, pp. 47-53. Roma. doi:10.5220/0005821900470053

Azzouz, A., Ennigrou, M., y Ben Said, L. (2017). A self-adaptive hybrid algorithm for solving flexible job-shop problem with sequence dependent setup time. *Procedia Computer Science, 112*, 457-466. doi:https://doi.org/10.1016/j.procs.2017.08.023

Bagheri, A., y Zandieh, M. (2011). Bi-criteria flexible job-shop scheduling with sequence-dependent setup times-Variable neighborhood search approach. *Journal of Manufacturing Systems, 30*(1), 8-15. doi:doi:10.1016/j.jmsy.2011.02.004

Ballicu, M., Giua, A., y Seatzu, C. (2002). Job-shop scheduling models with set-up times. *IEEE International Conference on Systems, Man and Cybernetics, 5*, 1-6. doi:10.1109/ICSMC.2002.1176335

Blum, C., y Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys, 35*(3), 268-308. Retrieved from https://www.iiia.csic.es/~christian.blum/.../blum_roli_2003.pdf

Blum, C., y Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys, 35*(3), 268-308. doi:10.1145/937503.937505

Brandimarte, P. (1993). Routing and Scheduling in a flexible job shop by tabu search. *Annals of Operations Research, 41*(3), 157-183. doi:10.1007/BF02023073

Brucker, P., y Knust, S. (2012). *Complex Scheduling.* Berlin: Springer.

Brucker, P., y R. Schlie. (Diciembre de 1990). Job-shop scheduling with multi-purpose machines. *Computing, 45*(4), 369-375. Obtenido de https://link.springer.com/article/10.1007/BF02238804

Bula, G. (2004). Programación de operaciones en el taller de trabajo utilizando la Meta-Heurística de Recocido Simulado. *Revista de la facultad de Ingenierías Fisicomecánicas, 3*(1), 21-28. Retrieved from http://revistas.uis.edu.co/index.php/revistauisingenierias/article/view/2269

Chang, H. C., Chen, Y. P., Liu, T. K., y Chou, J. H. (2015). Solving the Flexible Job Shop Scheduling Problem with Makespan Optimization by Using a Hybrid Taguchi-Genetic Algorithm. *IEEE Access, 3*, 1740-1754. doi:10.1109/ACCESS.2015.2481463

Choi, I.-C., y Choi, D.-s. (2002). A local search algorithm for shop scheduling problems with alternative operations and sequence dependent setups. *Computer and Industrial Engineering*, 43-58. doi:10.1016/S0360-8352(02)00002-5

Cruz-Chávez, M., Martínez-Rangel, M., y Cruz-Rosales, M. (2017). Accelerated simulated annealing algorithm applied to the flexible job shop scheduling problem. *International Transaction in Operational Research, 24*(5), 1119-1137. doi:10.1111/itor.12195

Dorigo, M., y Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation, 1*(1), 53-66. doi:10.1109/4235.585892

Dowsland, K., y Adenso Díaz, B. (2001). Heuristic design and fundamentals of the Simulated Annealing. *Revista Iberoamericana de Inteligencia Artificial*, 34-52. doi:10.4114/ia.v7i19.718

Driss, I., Mouss, K., y Laggoun, A. (2015). A new genetic algorithm for flexible job-shop scheduling problems. *Journal of Mechanical Science and Technology 29, 29*(3), 1273-1281. doi:10.1007/s12206-015-0242-7

Duarte, A., Pantrigo, J., y Gallego, M. (2008). Introducción a la optimización. In *Metaheurísticas* (pp. 1-14). Madrid.

F.T.S., C., y Wong, T. (2006). Flexible job-shop scheduling problem under resource constraints. *International Journal of Production Research, 44*(11), 2071-2089. doi:10.1080/00207540500386012

Fatos, X., y Ajith, A. (2008). *Metaheuristic for scheduling in industrial and Manufacturing applications.* Berlin: Springer.

Freitas Guimaraes, K., y Aparecida Fernandes, M. (2006). An approach for Flexible Job Shop scheduling with separable sequence dependent setup time . *IEEE International Conference on Systems, Man and Cybernetics*, (pp. 3727-3731). Taipei. doi:10.1109/ICSMC.2006.384709

Gao, J., Gen, M., Sun, L., y Zhao, X. (2007). A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems. *Computers y Industrial Engineering, 53*(1), 149-162. doi:https://doi.org/10.1016/j.cie.2007.04.010

Gao, J., Sun, L., y Gen, M. (2008). A hybrid genetic and variable neighborhood descent algrorithm for flexible job shop scheduling problems. *Computer and Operations research, 35*(9), 2892-2907. doi:https://doi.org/10.1016/j.cor.2007.01.001

Gen, M., Gao, J., y Lin, L. (2009). Multistage based genetic algorithm for flexible job shop scheduling problem. *Intelligent and Evolutionary Systems, 187*, 183-196. Retrieved from https://rd.springer.com/chapter/10.1007/978-3-540-95978-6_13

González, M. A., Vela, C. R., y Varela, R. (2013). An Efficient Memetic Algorithm for the Flexible Job Shop with Setup Times. *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling*, (pp. 91-99). Roma. Retrieved from https://www.aaai.org/ocs/index.php/ICAPS/ICAPS13/.../6167

Hamalainen, W. (2006, Noviembre 6). *https://cs.joensuu.fi*. Retrieved from https://cs.joensuu.fi/pages/whamalai/daa/npsession.pdf

He, Y., Weng, W., y Fujimura, S. (2017). Improvements to Genetic Algorithm for Flexible Job Shop Scheduling with Overlapping in Operations. *Computer and Information Science (ICIS), 2017 IEEE/ACIS 16th International Conference on* (pp. 791-796). Wuhan, China: IEEE. doi:10.1109/ICSMC.2003.1244425

Hurink, J., Jurisch, B., y Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum, 15*(4), 205-215. doi:10.1007/BF01719451

Imanipour, N. (2006). Modeling and solving flexible job-shop problem with sequence dependent setup times. *2006 International Conference on Service Systems and Service Management,*, 1205–1210. doi:10.1109/ICSSSM.2006.320680

Josefowska, J., y Jan weglarz. (2006). *Perspectives in modern project scheduling.* United States of America: Springer. doi:10.1007/978-0-387-33768-5

Kacem, I., Hammadi, S., y Borne, P. (2002). Approach by Localization and Multiobjective Evolutionary Optimization for Flexible Job Shop Scheduling Problems. *IEEE Transactions on Systems,man,and Cybernetics-Part C*, 1-13. doi:10.1109/TSMCC.2002.1009117

Kacem, I., Hammadi, S., y Borne, P. (2002b). Pareto optimality approach for flexible job shop scheduling problems:hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computer in Simulation 60, 60*(3-5), 245-276. doi:https://doi.org/10.1016/S0378-4754(02)00019-8

Kirkpatrick, S., Gelatt, C., y Vecchi, M. (1983). Optimization by Simulated Annealing. *Science, New Series, 220*(4598), 671-680. Retrieved from http://links.jstor.org/sici?sici=0036-8075%2819830513%293%3A220%3A4598%3C671%3AOBSA%3E2.0.CO%3B2-8

Lee, K., y Yamakawa, T. (1998). A genetic algorithm for general machine scheduling problems. *Int.Conf. on Conventional and knowledge-Based Electronics Systems Australia, 2*(2), 60-66. doi:10.1109/KES.1998.725893

Li, X., y Gao, L. (2016). An effective hybrid genetic algorithm and tabu search forfl exible job shop scheduling problem. *Int. J Production economics 174, 174*, 93-110. doi:http://dx.doi.org/10.1016/j.ijpe.2016.01.016

Martí, R. (2017, Febrero). *http://citeseerx.ist.psu.edu.* Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.419.3447yrank=1

Mastrolilli, M., y Gambardella, L. (2000). Effective neighborhood Functions for the flexible job shop problem. *Journal of Scheduling, 3*(1), 3-20. doi:10.1002/(SICI)1099-1425(200001/02)3:1<3::AID-JOS32>3.0.CO;2-Y

Najid, N., Dauzere-Pérés, S., y Zaidat, A. (2002). A modified simulated annealing methd for Flexible Job Shop Scheduling Problem. *IEEE International Conference on Systems, Man and Cybernetics*, (pp. 1-6). doi:10.1109/ICSMC.2002.1176334

Oddi, A., Rasconi, R., Cesta, A., y F.Smith, S. (2011). Applying Iterative Flattening Search to the Job Shop Scheduling Problem with alternatives resources and Sequence Dependent Setup Times. *ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*, 15-22. Retrieved from https://www.semanticscholar.org/paper/Applying-Iterative-Flattening-Search-to-the-Job-Sh-Oddi-Rasconi/d6fd7cba2c6111b7aed7fcb9d60fbfa9838d8206

Pan, Q.-k., Li, J.-q., Xie, S.-x., Jia , B.-x., y Yu-ting, W. (2010). A hybrid particle swarm optimization and tabu search algorithm for flexible job shop scheduling problem. *International Journal of Computer Theory and Engneering, 2*(2), 189-194. doi:https://doi.org/10.1016/j.proeng.2011.08.689

Paulli, J., y Dauzere-Peres, S. (1997). An integrated approach for modeling and solving the general multiprocessor job shop scheduling problem using tabu search. *Annals of Operations Research, 70*(0), 281-306. doi:https://rd.springer.com/article/10.1023/A:1018930406487

Pezzella, P., Morganti, G., y Ciaschetti, G. (2008). Problem, A genetic algorithm for the Flexible Job-shop Scheduling problem. *Computers y Operations Research, 35*(10), 3202-3212. doi:https://doi.org/10.1016/j.cor.2007.02.014

Pinedo, M. (2010). *Scheduling: Theory, Algorithms, and Systems* (Cuarta ed.). New York: En Springer New York Dordrecht Heidelberg London. doi:10.1007/978-1-4614-2361-4

Rossi, A., y Dini, G. (2007). Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. *Robotics and Computer Integrated Manufacturing 23,* 503-516. doi:https://doi.org/10.1016/j.rcim.2006.06.004

Saidi- Mehrabad, M., Fattahi, P., y Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *J. Intell. Manuf*(3), 331-342. doi:10.1007/s10845-007-0026-8

Saidi-Mehrabad , M., y Fattahi, P. (2007). Flexible job shop scheduling with tabu search algorithms. *The International Journal of Advanced Manufacturing Technology, 32*(5-6), 563-570. doi:0.1007/s00170-005-0375-4

Sharma, P., y Jain, A. (2015). A review on job shop scheduling with setup times. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 230*(3), 517-533. doi:10.1177/0954405414560617

Sharma, P., y Jain, A. (2015). Stochastic Dynamic Job Shop Scheduling with Sequence Dependent Setup Tines:Simulation Experimentation. *Journal of Engineering and Technology, 5*(1), 19-25. doi:10.4103/0976-8580.149475

Sotskov, Y., Tautenhahn, T., y Werner, F. (1999). On the application of insertion techniques for job shop problems with setup times. *RAIRO.Recherche opérationnelle, 33*(2), 209-245. Retrieved from http://www.numdam.org/item?id=RO_1999__33_2_209_0

Sun, W., Pan, Y., Lu, X., y Ma, Q. (2010). Research on flexible job shop scheduling problem based on a modified genetic algorithmn. *Journal of Mechanical Science and techology, 24*(10), 2119-2125. doi:https://doi.org/10.1007/s12206-010-0526-x

Tang, J., Zhanj, G., Lin, B., y Zhang, B. (2011). A hybrid Algorithm for flexible Job Shop scheduling problem. *Procedia Engineering, 15*, 3678-3683. doi:https://doi.org/10.1016/j.proeng.2011.08.689

Tejada Muñoz, G. (2016). Enrutamiento y secuenciación óptimos en un Flexible Job Shop Multiobjetivo mediante Algoritmos Genéticos. *Revista Industrial Data 19(2)*, 124-133. doi:http://dx.doi.org/10.15381/idata.v19i2.12846

Wan, M., Zhang, F., Fan, X., y Bai, C. (2010). An integrated Genetic Algorithm for Flexible Job Shop Scheduling Problem. *2010 International Conference on Computational Intelligence and Software Engineering,*, (pp. 1-4). Wuhan. doi:10.1109/CISE.2010.5676961

Wu, Z., y Weng, M. (2005). Multiagent scheduling method with earliness and tardiness objectives in flexible job shops. *IEEE Transactions on systems, Man, and Cybernetics,Part B 35, 35*(2), 293-301. doi:10.1109/TSMCB.2004.842412

Yazdani, M., Gholami, M., Zandieh, M., y Mousakhani, M. (2009). A simulated Annealing Algorithm for Flexible Job-Shop Scheduling Problem. *Journal of Applied Sciences 9*, 662-670. doi:10.3923/jas.2009.662.670

Zäpfel, G., Braune, R., y Bögl, M. (2010). *Metaheuristic Search Concepts:A tutorial with applications to production and logistics.* Berlin, Alemania: Springer Heidelberg Dordrecht London New York. doi:10.1007/978-3-642-11343-7

Zhang, G., Gao, L., y Shi, Y. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications, 38*(4), 3563–3573. doi:10.1016/j.eswa.2010.08.145

Zribi, N., Kacem, I., El Kamel,, A., y Borne, P. (2007, Julio). Assignment and Scheduling in Flexible Job-Shops by Hierarchical Optimization. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews, 37*(4), 652-661. doi:10.1109/TSMCC.2007.897494