Comparación del desempeño de metaheurísticas híbridas para el problema de Flowshop Distribuido y Permutado con Etapa de Ensamble, considerando tiempos de alistamiento dependientes de la secuencia (DAPFSP-SDST) y fábricas heterogéneas.

Silvia Juliana Dagovett Cala, Susana Prada Avellaneda

Trabajo de Grado para Optar el Título de Ingeniero Industrial

Director

Edwin Alberto Garavito Hernández

Magister en Ingeniería Industrial

Codirector

Laura Yeraldín Escobar Rodríguez

Ingeniera Industrial

Universidad Industrial de Santander

Escuela de Estudios Industriales y Empresariales

Bucaramanga

2019

Dedicatoria

A mi familia: mi mami, mi papi, mi hermana, mi hermano y mi sobrina por ser mi motivación diaria y apoyarme a lo largo de este camino, todo lo que soy se lo debo a ustedes, a su ejemplo y a su entero amor, los amo con todo mi corazón.

Silvia Juliana Dagovett Cala

A mis padres, por su amor, enseñanzas y valores durante cada paso de mi vida

Susana Prada Avellaneda

Agradecimientos

Quiero iniciar dándole gracias a Dios, él siempre ha puesto ángeles en mi camino que me han ayudado a lograr mis objetivos de vida. A mi mamita hermosa, mi motivación, mi compañera y mi mejor amiga, fuiste una pieza clave en este proceso. A mi papá por su apoyo. A mi hermana Andrea, quien fue muy alcahueta y la más interesada en que lograra un desarrollo profesional integral, gracias por España. A mi hermano el fastidioso por estar para mí en todo momento y prestarme sus aparatos electrónicos para aprender un poquito más. A mi sobrina por ser una gran hermanita menor.

Por otra parte, quiero agradecerle al profesor Edwin Garavito, por ser un gran mentor y apoyarnos a lo largo del proyecto. A Laura Escobar, por su dedicación, entrega y compromiso. A mi compañera la dramática, Susana, sin tu compañía, lograr este objetivo de vida habría sido muy complicado, eres una amiga de esas que se consideran hermanas.

Finalmente quiero agradecerle a mi bonito, a Jack Black, a Mauri y a todos esos amigos incondicionales que estuvieron apoyándonos en este camino.

Silvia Juliana Dagovett Cala

A Dios, porque es el principio de todo, sin Él nada de esto hubiese sido posible. A mis papás quienes me enseñaron el valor de la educación y el deseo de ser cada día una mejor versión de mí misma. A mi mis hermanos, Rubén, Fernando y Sergio, quienes nunca dudaron de mis esfuerzos y fueron mi apoyo incondicional.

A Edward, por ser parte de mi vida y acompañarme en esta etapa. Al profe Edwin y a Laura, quienes creyeron y me apoyaron en este proyecto, pues su guía terminó haciendo de esto una realidad. Al grupo de investigación OPALO, por estar siempre prestos a brindar apoyo y compartir enseñanzas y consejos. Y a todas y cada una de las personas que de uno u otro modo me ayudaron y apoyaron a que este proyecto se haya transformado de un sueño a un hecho, gracias a todos ellos.

Susana Prada Avellaneda

Tabla de contenido

Introducción 18

1. Planteamiento del Problema 20

1. Justificación 22

2. Objetivos 24

2.1. Objetivo General 24

2.2. Objetivos Específicos 24

3. Revisión de la Literatura 25

3.1. Análisis Bibliométrico 25

3.2. Análisis de la Literatura 31

4. Marco Teórico 42

4.1. Flowshop (FSP) 42

4.2. Flowshop Permutado (PSFP) 43

4.3. Flowshop Permutado y Distribuido (DPFSP) 44

4.4. Flowshop Permutado y Distribuido con etapa de Ensamble (DAPFSP) 45

4.5. Tiempos de alistamiento (Setup Times) 45

4.6. Makespan 46

4.7. Programación lineal entera mixta (MILP) 46

4.8. Heurísticas 47

4.9. Metaheurísticas 48

4.9.1. Algoritmo genético 48

4.10. Complejidad computacional 56

4.11. Optimización 57

4.12. Análisis de varianza ANOVA 60

4.13. Reglas de asignación de trabajos 60

5. Descripción del DAPFSP-SDST con fábricas heterogéneas 62
5.1. Modelo matemático 64
6. Algoritmo Genético (GA) 68
6.1. Representación de la solución para GA1 y GA2 69
6.2. Descripción del algoritmo *GA1* 70
6.3. Descripción del algoritmo GA2 78
7. Algoritmo híbrido entre Algoritmo Genético y VND (*HGA1-VND*) 81
7.1. Descripción del algoritmo 81
8. Algoritmo híbrido entre Algoritmo Genético y Algoritmo Voraz (*HGA1-GR*) 84
8.1. Descripción del algoritmo 84
9. Calibración de parámetros de los algoritmos 86
9.1. Instancias pequeñas 88
9.2. Instancias medianas 94
9.3. Instancias grandes 101
10. Evaluación de desempeño de los Algoritmos 107
10.1. Análisis del desempeño para el Algoritmo Genético GA1 108
10.2. Análisis del desempeño para el Algoritmo híbrido entre Algoritmo Genético y VND (HGA1-VND) 111
10.3. Análisis del desempeño para el Algoritmo híbrido entre Algoritmo Genético y Algoritmo Voraz (HGA1-GR) 114
10.4. Comparación del rendimiento de los tres algoritmos 117
11. Resultados de los algoritmos 118
12. Conclusiones 121
13. Recomendaciones 124
Referencias Bibliográficas 125

Lista de Tablas

Tabla 1.Cantidad de publicaciones 26

Tabla 2.Cantidad de publicaciones después de realizar los filtros 27

Tabla 3. Autores con más publicaciones 28

Tabla 4. Palabras clave 29

Tabla 5. Cantidad de artículos 29

Tabla 6. Algoritmos para solucionar el DPFSP 39

Tabla 7. Métodos de selección. 50

Tabla 8. Tipos de operadores de cruce. 51

Tabla 9. Clases de complejidad computacional 57

Tabla 10. Componentes de un problema de optimización 59

Tabla 11. Reglas de asignación de trabajos 61

Tabla 12. Factores y niveles del diseño de experimentos 86

Tabla 13. Niveles de factores seleccionados para instancias de 8 trabajos 91

Tabla 14. Niveles de factores seleccionados para instancias de 12 trabajos 94

Tabla 15. Niveles de factores seleccionados para instancias de 16 trabajos 96

Tabla 16. Niveles de factores seleccionados para instancias de 20 trabajos 99

Tabla 17. Niveles de factores seleccionados para instancias de 24 trabajos 101

Tabla 18. Niveles de factores seleccionados para instancias de 100 trabajos 104

Tabla 19. Niveles de factores seleccionados para instancias de 200 trabajos 106

Tabla 20. Comparación de instancias GA1 109

Tabla 21. Comparación de instancias HGA1-VND 112

Tabla 22. Comparación instancias HGA1-GR 115

Tabla 23. Porcentaje de instancias mejoradas 117

Tabla 24. Resúmenes resultados GA1 118

Tabla 25. Resúmenes resultados HGA1-VND 119

Tabla 26. Resúmenes resultados HGA1-GR 120

Lista de Figuras

Figura 1. Representación esquemática del DAPFSP 20

Figura 2. Ecuaciones de búsqueda 25

Figura 3. Publicaciones por año. 28

Figura 4. Publicaciones por país 30

Figura 5. Mapa jerárquico de nodos. A 31

Figura 6. Variantes FSP. 32

Figura 7. Representación del proceso en un Flowshop. 42

Figura 8. Representación gráfica de DAPFSP. 45

Figura 9. Etapas del algoritmo genético 49

Figura 10. Etapas del algoritmo voraz. 53

Figura 11. Pseudocódigo del algoritmo voraz iterativo. 54

Figura 12. Estructura general del algoritmo VND. 55

Figura 13. Pseudocódigo algoritmo VND.. 56

Figura 14.Métodos de la optimización. 58

Figura 15. Vector solución. 69

Figura 16. Diagrama de flujo GA1 71

Figura 17. Operador Selección 74

Figura 18. Operador cruce. Primera etapa.. 75

Figura 19. Operador cruce. Segunda etapa. 76

Figura 20. Operador cruce. Etapa final. 76

Figura 21. Operador mutación.. 78

Figura 22 Diagrama de flujo GA2 79

Figura 23. Diagrama de flujo HGA1-VND 83
Figura 24. Diagrama de flujo HGA1-GR 85
Figura 25. Grupos y subgrupos de instancias 87
Figura 26. Efectos principales para CPU time en instancias de 8 trabajos del GA1 89
Figura 27. Efectos principales para CPU time en instancias de 8 trabajos del HGA1-VND ... 90
Figura 28. Efectos principales para CPU time en instancias de 8 trabajos del HGA1-GR 90
Figura 29. Efectos principales para makespan en instancias de 12 trabajos del GA1 92
Figura 30. Efectos principales para makespan en instancias de 12 trabajos del HGA1-VND 93
Figura 31. Efectos principales para makespan en instancias de 12 trabajos del HGA1-GR ... 93
Figura 32. Efectos principales para makespan en instancias de 16 trabajos del GA1 95
Figura 33. Efectos principales para makespan en instancias de 16 trabajos del HGA1-VND 95
Figura 34. Efectos principales para makespan en instancias de 16 trabajos del HGA1-GR ... 96
Figura 35. Efectos principales para makespan en instancias de 20 trabajos del GA1 97
Figura 36. Efectos principales para makespan en instancias de 20 trabajos del HGA1-VND 98
Figura 37. Efectos principales para makespan en instancias de 20 trabajos del HGA1-GR ... 98
Figura 38. Efectos principales para makespan en instancias de 24 trabajos del GA1 100
Figura 39. Efectos principales para makespan en instancias de 24 trabajos del HGA1-VND 100
Figura 40. Efectos principales para makespan en instancias de 24 trabajos del HGA1-GR . 101
Figura 41. Efectos principales para makespan en instancias de 100 trabajos del GA1 103
Figura 42. Efectos principales para makespan en instancias de 100 trabajos del HGA1-VND 103

Figura 43. Efectos principales para makespan en instancias de 100 trabajos del HGA1- GR 104

Figura 44. Efectos principales para makespan en instancias de 200 trabajos del GA1 105

Figura 45. Efectos principales para makespan en instancias de 200 trabajos del HGA1-VND 105

Figura 46. Efectos principales para makespan en instancias de 100 trabajos del HGA1-GR 106

Figura 47. RPD para el GA1 111

Figura 48. RPD para el HGA1-VND 114

Figura 49. RPD para el HGA1-GR 117

Figura 50. Desviación vs Instancias para GA1 119

Figura 51. Desviación vs instancias para HGA1-VND 120

Figura 52. Desviación vs Instancias para HGA1-GR 121

Lista de Apéndices

Los apéndices se encuentran disponibles en el CD anexo:

Apéndice A. Filtro de palabras clave y título

Apéndice B. Filtro de palabras clave y título

Apéndice C. Scripts Algoritmo genético

Apéndice D. Scripts Algoritmo hibrido entre genético y VND

Apéndice E. Scripts Algoritmo hibrido entre genético y voraz

Apéndice F. Instancias calibración GA1

Apéndice G. Instancias calibración HGA1-VND

Apéndice H. Instancias calibración HGA1-VORAZ

Apéndice I. Cálculos de RPD

Apéndice J. Resultados

Apéndice K. Artículo

Resumen

Título: "Comparación Del Desempeño De Metaheurísticas Híbridas Para El Problema De Flowshop Distribuido Y Permutado Con Etapa De Ensamble, Considerando Tiempos De Alistamiento Dependientes De La Secuencia (DAPFSP-SDST) Y Fábricas Heterogéneas"[1]

Autores:

Dagovett Cala Silvia Juliana

Prada Avellaneda Susana[2]

Palabras claves: Flowshop, distribuido, ensamble, permutado, *makespan*, fábricas heterogéneas, algoritmo genético, VND, algoritmo voraz

Descripción:

En esta investigación se aborda el problema de Flowshop Distribuido y Permutado con etapa de ensamble, considerando tiempos de alistamiento dependientes de la secuencia y fábricas heterogéneas. La función objetivo busca minimizar el makespan o tiempo total completamiento. Para dar solución al ((DAPFSP-SDST) con fábricas heterogéneas se proponen tres algoritmos metaheurísticos Algoritmo Genético (GA1), Algoritmo híbrido entre genético con VND (HGA1-VND) y algoritmo hibrido entre genético y voraz (HGA1-GR). Para la calibración de los algoritmos se realiza un diseño de experimentos 3^4 con el objetivo de seleccionar los mejores niveles de los factores: población inicial, probabilidad de cruce, probabilidad de mutación y número de iteraciones. Los algoritmos son implementados en el software MATLAB R2018b y el diseño de experimentos es analizado en el software MINITAB19.

El desempeño de los algoritmos se evalúa a través del indicador RPD; para ello, se comparan los resultados de los algoritmos en 150 instancias, las cuales son contrastadas con la mejor solución existente en la literatura. Se encuentra que la metaheurística que mejor makespan obtiene es el hibrido entre algoritmo genético y VND. Asimismo, se concluye que los tres algoritmos presentan un mejor desempeño para la solución de instancias pequeñas y medianas que los existentes en la literatura.

---

[1] Proyecto de grado

[2] Facultad de Ingenierías Físico-mecánicas. Escuela de Estudios Industriales y Empresariales. Programa de Ingeniería Industrial. Director: M. Sc. Edwin Alberto Garavito Hernández. Codirector: Laura Yeraldín Escobar Rodríguez

Abstract

Title: "Comparison of the Performance of Hybrid Metaheuristics for the Distributed Assembly and Permutated Flowshop Scheduling Problem, With Sequence Dependent Setup Times (DAPFSP-SDST) and Heterogeneous Factories"[3]

Autor:s

Dagovett Cala Silvia Juliana

Prada Avellaneda Susana[4]

Key Words: Flowshop, distributed, assembly, permutated, makespan, heterogeneous factories, genetic algorithm, VND, greedy algorithm

Description:

This research addresses the distributed assembled and permuted flowshop problem with sequence-dependent setup times (DAPFSP-SDST) and heterogeneous factories. The objective function seeks to minimize makespan or total completion time. To solve the DAPFSP-SDST with heterogeneous factories, three metaheuristic algorithms are proposed: Genetic algorithm (GA1), a Hybrid genetic algorithm with Variable Neighborhood Descent (HGA1-VND) and a hybrid genetic algorithm and greedy algorithm (HGA1-GR). We made an experimental design 3^4 in order to select the best levels of the factors: initial population, crossover probability, mutation probability, and number of iterations. The algorithms are implemented in the MATLAB R2018b software and the experimental design is analyzed in the MINITAB19 software.

The performance of the algorithms is evaluated through the RPD indicator; for this, the results of the algorithms in 150 instances are compared with the best solution found in the literature. We conclude that the best results of the objective function are obtained with the HGA1-VND metaheuristic. In addition, it is concluded that the three algorithms present a better performance in small and medium instances than those existing in the literature.

[3] Barchelor Thesis

[4] Faculty of Physicomechanical Engineering, Industrial and Business School. Industrial Engineering. Advisor M. Sc. Edwin Alberto Garavito Hernández. Co-advisor Ing. Laura Yeraldín Escobar Rodríguez

Introducción

Para que las organizaciones obtengan una ventaja competitiva en el entorno empresarial actual es necesario establecer estrategias corporativas que permitan gestionar correctamente los procesos y recursos. Entre las estrategias corporativas, la adecuada gestión de la producción tiene una relevancia significativa en las empresas de manufactura debido a que, se enfocan en alinear factores como la calidad, velocidad, flexibilidad y costos de los procesos con los objetivos organizacionales.

De los problemas de optimización de procesos productivos existentes, el Flowshop (Taller de Flujo) se considera relevante por su objetivo de minimizar los tiempos de producción, además presenta una gran cantidad de estudios que muestran las modificaciones generadas como consecuencia de la necesidad de solucionar problemas reales en la manufactura. Asimismo, bajo el planteamiento de una fábrica con dos estaciones de producción, una máquina en cada estación y *n* productos, S.M. Johnson (1954) da a conocer el primer artículo de investigación sobre este problema, generando así la oportunidad abordar situaciones similares que consideran otro tipo de parámetros o variables adicionales.

El Flowshop Distribuido y Permutado con Etapa de Ensamble considerando tiempos de alistamiento dependientes de la secuencia (DAPFSP-SDST) es una adaptación del Flowshop, donde se busca encontrar una secuencia de fabricación y ensamble que minimice el tiempo total de producción, *makespan*, para *n* trabajos, *j* productos y *f* fábricas heterogéneas (Hatami, Ruiz, & Romano, 2015). De esta manera, existe la necesidad de proponer soluciones que contribuyan a la administración efectiva de los procesos de manufactura.

Esta investigación pretende dar un aporte a la optimización de los procesos productivos de las compañías manufactureras, para contribuir a la minimización del tiempo de producción y asimismo reducir el tiempo de alistamiento, acumulación de inventario, pedidos retrasados e incluso la pérdida de productos por fecha de caducidad considerando como problema el DAPFSP-SDST. Para la solución de este problema se desarrollan y comparar tres métodos de solución: un algoritmo genético (*Genetic Algorithm*, *GA*), un algoritmo genético híbrido con algoritmo voraz y un algoritmo genético híbrido con algoritmo de vecindario en búsqueda variable (*Variable Neighbourhood Descent, VND*). Estos algoritmos deben dar solución al problema en tiempos computacionales aceptables, después son evaluados y comparados con pruebas estadísticas para realizar la identificación de aquel cuya eficiencia sea la mejor.

Dentro de este documento en los capítulos 4 y 5, se presenta la revisión de literatura conformada por el análisis bibliométrico y bibliográfico, donde se pretende ahondar sobre los últimos estudios del DAPFSP-SDST y los métodos de solución más usados para este problema. Posteriormente, en el capítulo 6 se realiza una definición del problema, enunciando las consideraciones que se deben tener en cuenta para su análisis. A continuación, en los capítulos 7, 8 y 9 se describen los detalles del diseño de cada uno de los algoritmos propuestos. Para la calibración de estos, se lleva a cabo un diseño de experimentos con el fin de identificar la combinación de parámetros que generan mejor desempeño, esta calibración es descrita en el capítulo 10.

Finalmente, en el capítulo 11 se lleva a cabo una comparación de las eficiencias de cada algoritmo, considerando diferentes factores que intervienen durante el problema. Asimismo, se realizan los análisis complementarios pertinentes, con el fin de dar paso a las conclusiones y recomendaciones para tener en cuenta en investigaciones futuras.

1. Planteamiento del Problema

La optimización de procesos enfocada a la minimización de los tiempos de producción es un factor clave en la industria actual, ya que esto permite que las compañías maximicen sus beneficios, reduzcan costos y aumenten su producción, mientras aprovechan al máximo la capacidad de la planta. Ahora bien, con el objetivo de lograr esto, la programación de trabajos dentro de la producción, se vuelve un tema de estudio pertinente. Considerando lo anterior, en la presente investigación se estudia una variación del problema de Flowshop Distribuido y Permutado con etapa de ensamble, en él, se asume la relación entre *f* fábricas, *m* máquinas, *n* trabajos y *p* productos. Para su solución, se debe encontrar la secuencia optima de producción, la cual será la misma para todas las máquinas. La secuencia de procesamiento se encuentra con el objetivo de minimizar el *makespan* o tiempo de completamiento. Este problema se caracteriza por tener dos etapas, en la primera etapa o etapa de procesamiento, las *m* máquinas procesan los *n* trabajos y elaboran todos los componentes que se van a enviar a la segunda etapa, donde una máquina *M* es la encargada de ensamblar estos componentes con el fin de obtener el producto final *p*. En la Figura 1 se muestran las dos fases del problema.

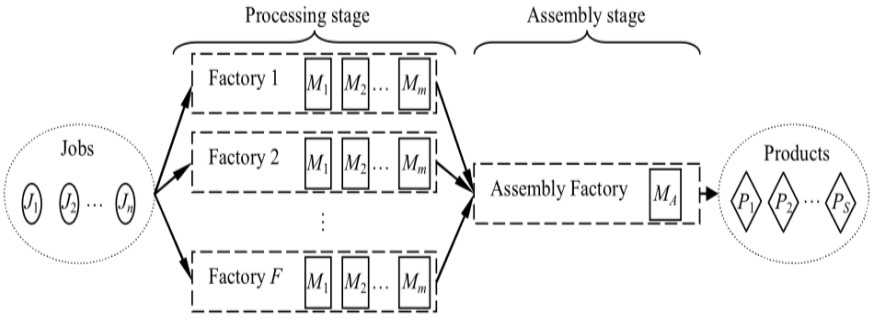


Figura 1. Representación esquemática del DAPFSP. Tomado de Wang, S. Y., & Wang, L. (2016). An Estimation of Distribution Algorithm-Based Memetic Algorithm for the Distributed Assembly

Permutation Flow-Shop Scheduling Problem. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 46(1), 139–149. https://doi.org/10.1109/TSMC.2015.2416127

El DAPFSP-SDST con fábricas heterogéneas, tiene en cuenta algunas consideraciones para su solución, como que un trabajo se procesa en solo una fábrica, donde cada fábrica tiene la misma cantidad de máquinas y cada máquina procesa un solo trabajo a la vez. Este trabajo, solo avanza a la siguiente máquina si ha terminado su procesamiento en la máquina actual, se consideran trabajos ficticios cero, que representan el estado inicial de la máquina. Los tiempos de alistamiento y procesamiento son distintos entre máquinas y fábricas, su valor se conoce y es determinístico y, finalmente, para poder ensamblar un producto *p* todos los trabajos *n* que lo componen deben haber terminado el proceso de producción. Considerando la complejidad computacional del problema a tratar, se plantea como oportunidad de estudio, el desarrollo de heurísticas y metaheurísticas para encontrar buenas soluciones al problema, en tiempos computacionales razonables.

Por otra parte, la comparación de metaheurísticas para problemas de optimización se realiza con el propósito de encontrar aquella que es mejor respecto a un criterio establecido, este puede ser el tiempo computacional, la eficiencia, entre otros. Teniendo en cuenta que en la literatura se proponen múltiples enfoques de solución para problemas asociados al Flowshop, se propone realizar la comparación de tres metaheurísticas: algoritmo genético, algoritmo híbrido entre los algoritmos genético y voraz y algoritmo híbrido entre los algoritmos genético y VND, con el fin de evaluar su desempeño para el DAPFSP-SDST con fábricas heterogéneas.

Teniendo en cuenta lo anterior se plantean tres metaheurísticas para dar solución al DAPFSP-SDST, lo cual resulta pertinente porque este problema busca implantar la programación optima de trabajos en máquinas, dentro de un proceso de producción. En Colombia, se encuentran numerosas

empresas manufactureras, que deben ser competitivas tanto con la industria nacional como con las compañías internacionales que logran captar mercado en el territorio colombiano. Por tal motivo la escuela de Estudios Industriales y Empresariales y en especial el grupo de investigación OPALO, ha estudiado el DAPFSP-SDST y otros tipos de problemas relacionados con la optimización de procesos, para dar un aporte a la planeación la producción de las industrias colombianas.

1. Justificación

En la actualidad el entorno económico es cada vez más competitivo, las empresas se ven obligadas a compartir la demanda no solo con compañías nacionales, sino que también con compañías extranjeras. Algunas de estas por su avanzada tecnología, estrategias corporativas y adecuada gestión de los procesos, logran importar bienes y servicios de calidad a precios más asequibles y atractivos para el consumidor. Por tal motivo, es importante implementar tácticas que permitan a las empresas adquirir un posicionamiento significativo en el mercado internacional. Dentro de estas tácticas, se encuentra la expansión de la infraestructura de las empresas, en países donde los impuestos y costos son menores, aumentando así, las posibilidades de captar una mayor porción de mercado mundial.

Cuando una organización, cuenta con fábricas posicionadas en diferentes países, se presentan múltiples problemas logísticos y operacionales, que obstaculizan el cumplimento de los objetivos organizacionales y limitan el crecimiento de la empresa en el mercado. Por lo tanto, a lo largo de

la historia han sido usados múltiples algoritmos, modelos matemáticos y lenguajes de programación, para hacer más sencilla la toma de decisiones.

Normalmente, existen organizaciones que ubican sus diferentes fábricas en lugares distanciados. Situaciones así se conocen como problemas de Flowshop Distribuido y Permutado con Etapa de Ensamble (DAPFSP). Estos problemas se caracterizan por contar con dos etapas, una de procesamiento y otra de ensamble, que están representadas por fábricas diferentes, dentro de las que se deben encontrar las secuencias adecuadas de procesamiento y ensamble, mientras se minimizan los tiempos de completamiento de los productos.

La presente investigación centra su atención en el DAPFSP, considerando la presencia de tiempos de alistamiento dependientes de la secuencia SDST, debido a que, en la literatura, no existe evidencia que ratifique la existencia de métodos de solución óptimos para este problema y por el contrario se encuentran múltiples recomendaciones sobre posibles parámetros, variables y factores para tener en cuenta a la hora de estudiar métodos de solución que se adapten al DAPFSP-SDST.

2. Objetivos

2.1. Objetivo General

Comparar el desempeño de metaheurísticas basadas en algoritmos genéticos e híbridos para el problema de Flowshop Distribuido y Permutado con Etapa de Ensamble considerando tiempos de alistamiento dependientes de la secuencia (DAPFSP-SDST) y fábricas heterogéneas.

2.2. Objetivos Específicos

- Realizar una revisión bibliográfica sobre problema DAPFSP-SDST y sus métodos de solución para encontrar las heurísticas y metaheurísticas más utilizadas.
- Diseñar tres métodos de solución basados en el algoritmo genético e implementarlos en el software Matlab.
- Evaluar el desempeño de los algoritmos mediante la comparación con instancias de la literatura o a través de un análisis estadístico de los resultados.
- Elaborar un artículo de carácter publicable con el análisis, resultados y conclusiones del trabajo de investigación.

3. Revisión de la Literatura

3.1. Análisis Bibliométrico

Con la intención de estar al tanto de las últimas investigaciones acerca del problema de Flowshop Distribuido y Permutado con etapa de ensamble (DAPFSP-SDST) y de las condiciones principales manejadas en esta investigación , como lo son el uso de fábricas heterogéneas y tiempos de alistamiento, se crean ecuaciones de búsqueda, las cuales son ingresadas en las bases de datos disponibles de la Universidad Industrial de Santander, SCOPUS, SCIENCE DIRECT y WEB OF SCIENCE (Figura 2).

Web of Science: *Tema:* ((Flowshop) AND (Distributed OR Permutation) AND (Setup times))

Scopus: (*TITLE-ABS-KEY* **(**(Flowshop) AND (Distributed OR Permutation) AND (Setup times))

Science Direct: *TITLE-ABSTR-KEY* ((**(**Flowshop) AND (Distributed AND Permutation) AND (Setup times))

Figura 2. Ecuaciones de búsqueda

Debido a que se va a manejar una pequeña ventana de búsqueda, comprendida desde el año 2016 en adelante, se decide utilizar las tres bases de datos, con la intención de abarcar gran parte de los artículos publicados desde el 2016. A todas estas bases de datos se les aplica el filtro de artículos de investigación. La cantidad de publicaciones encontradas en las respectivas bases de datos se puede observar en la

Tabla 1.

Cantidad de publicaciones

Bases de datos	***Número de documentos encontrados***
Scopus	15
Science Direct	105
Web of Science	33
Total	153

Después de obtener estos resultados y observar que existe una diferencia significativa entre la cantidad de publicaciones de la base de datos Science Direct respecto a las otras dos; se encontró que esta base de datos ha abarcado una gran cantidad de tipos de publicaciones que no están relacionadas con nuestro proceso de investigación, por tal motivo se decide aplicar los siguientes filtros:

- Research Articles
- Computers & Industrial Engineering
- Applied Soft Computing
- Computer & Operations Research
- European Journal of Operational Research
- International Journal of Production Economics
- Journal of Cleaner Production
- Procedia Computer Science

Finalmente, las bases de datos arrojan la cantidad de publicaciones presentada en la Tabla 2

Tabla 2.

Cantidad de publicaciones después de realizar los filtros

Bases de datos	***Número de documentos encontrados***
Scopus	15
Science Direct	73
Web of Science	33
Total	121

Ahora bien, con la intención de hacer un análisis de estos artículos, detectar tendencias o comportamientos, descartar documentos que se encuentran en más de una base de datos, se hace uso del software VANTAGE POINT. De esta manera, el software elimina diez artículos de la base de datos Web of Science que se encontraban repetidos.

A continuación, en la Figura 3 se presenta una gráfica que muestra el número de publicaciones hechas por año. Los porcentajes de publicaciones por año para el 2016, 2017 y 2018 son 32%, 38% y 29%, respectivamente, donde se encontró que no hay diferencia porcentual relevante.

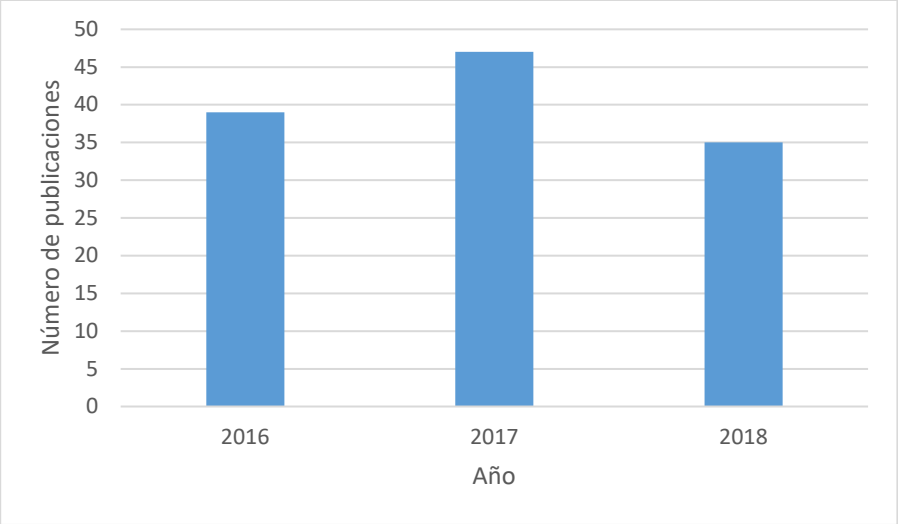


Figura 3. Publicaciones por año.

Es también importante identificar los autores que han estado más activos durante este tiempo en el desarrollo de investigaciones sobre el problema planteado en el presente trabajo, ya que sus publicaciones pueden presentar una continuidad o estar relacionadas, por tal motivo se encontraría una investigación más profunda acerca de una temática o un método de solución específico (ver Tabla 3).

Tabla 3.

Autores con más publicaciones

Autores	**Número de publicaciones**
Pi, Dechang	4
Shao, Weishi	4
Shao, Zhongshi	4
Dawal, Siti Zawiah Md	3
Fernandez-Viagas, Victor	3
Gao, Liang	3

Por otra parte, haciendo un análisis de las palabras claves en cada artículo, se presentan en la Tabla 4 las más frecuentes, esta tabla ayuda a validar qué tan acertadas son las ecuaciones de búsqueda según los criterios deseados.

Tabla 4.

Palabras clave

Palabras clave	***Número de publicaciones***
Scheduling	21
Makespan	9
Genetic algorithm	6
Multi-objective optimiation	6
Local search	5

Siguiendo con la revisión bibliográfica, se realiza un nuevo filtro analizando el título de los artículos y las palabras claves. Se descartan en esta fase artículos que no se relacionan directamente con el tema de Flowshop, obteniendo así la siguiente cantidad de publicaciones (ver Tabla 5). Por otra parte, el listado de artículos se encuentra en el Apéndice A.

Tabla 5.

Cantidad de artículos

Bases de datos	***Número de documentos encontrados***
Scopus	13
Science Direct	50
Web of Science	20
Total	83

Posterior a esto, se procede a hacer un análisis de países en donde se realizan las publicaciones. Se puede observar en la Figura 4 que los países donde más se efectuaron publicaciones son China, Inglaterra, Estados Unidos, Irán y Holanda con 25%, 13%, 10%, 10% y 9% de los documentos publicados respectivamente. Es importante resaltar que, de los veintiún países, los cinco mencionados acumulan el 67% de todos los artículos publicados, ya que más 50% de los países solo realizaron entre una y dos publicaciones en este periodo.

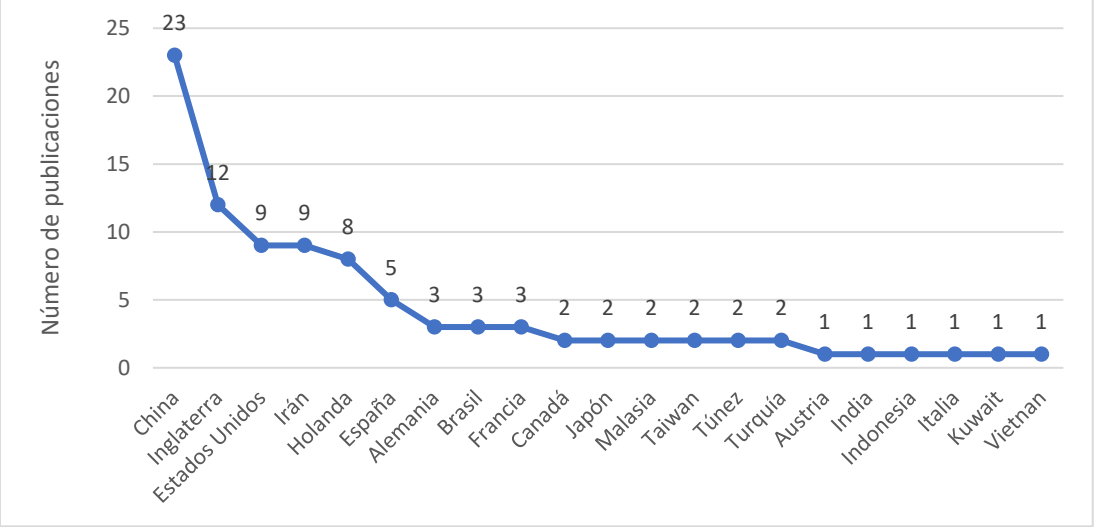


Figura 4. Publicaciones por país

Posterior a este análisis se hace lectura del resumen y/o introducción de cada artículo y se eliminan aquellos que no hablan sobre la programación de máquinas respecto a una secuencia producción, al finalizar este análisis el número obtenido de publicaciones es 49. Simultáneamente a este proceso se hace un análisis de nodos en el software NVIVO donde se lleva a cabo una clasificación de nodos respecto a las palabras claves y a los métodos de solución (Ver Figura 5). Apéndice B.

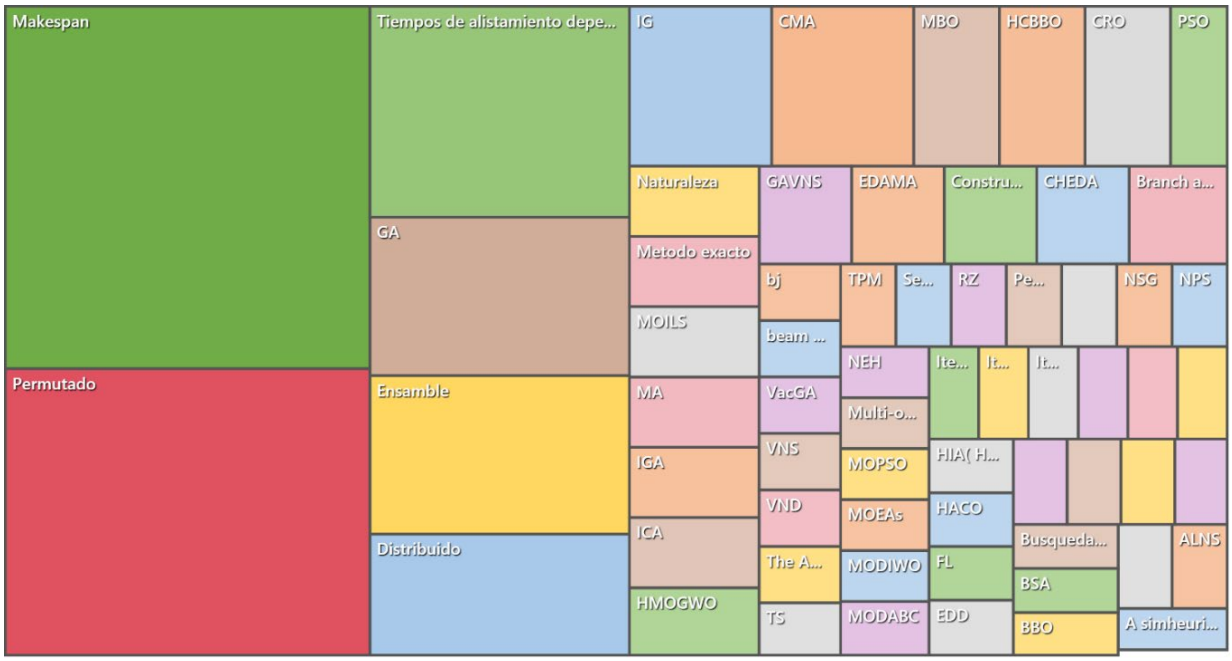


Figura 5. Mapa jerárquico de nodos. Adaptado del software NVIVO https://www.qsrinternational.com/nvivo/trial/trial-spanish/free-trial-download-windows

3.2. Análisis de la Literatura

El estudio del Flowshop empezó en 1954 con el artículo propuesto por Johnson (1954). Desde ese momento varios investigadores han continuado con el estudio de este tipo de problema, además, han agregado modificaciones para adaptarlo a situaciones reales de la industria. En la Figura 6 se muestra la evolución y variantes de algunas ramas que ha tenido este problema para llegar a lo que se conoce hoy, como Flowshop Permutado y Distribuido con Etapa de Ensamble.

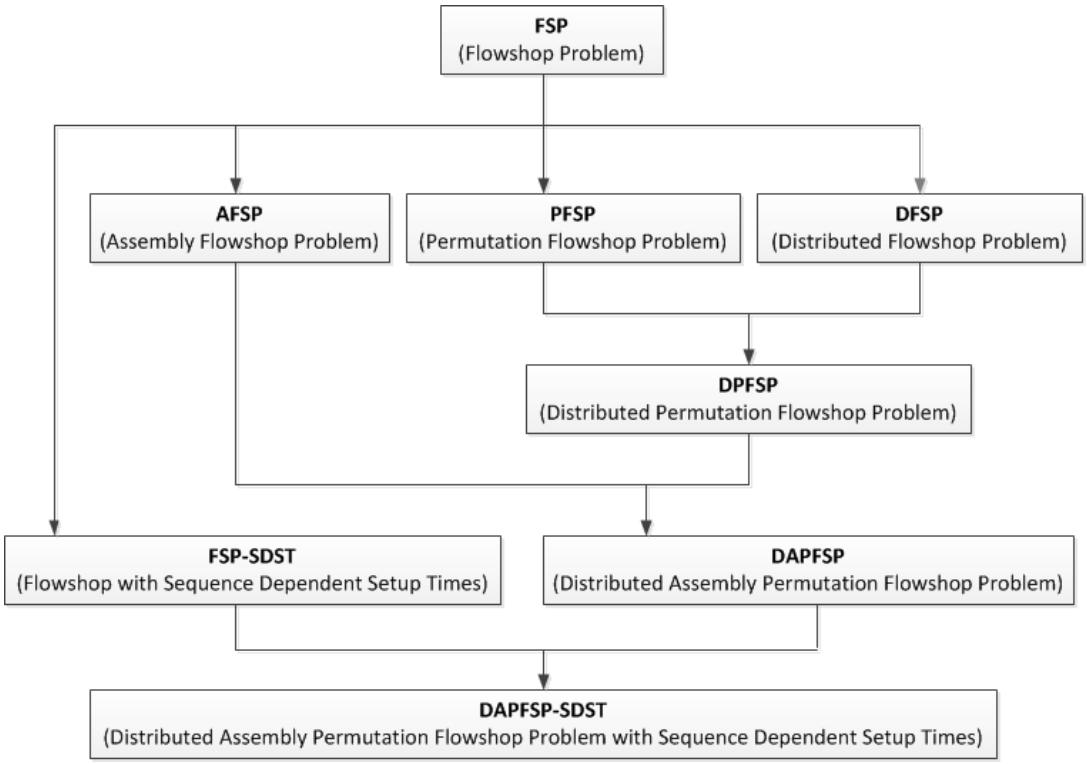


Figura 6. Variantes FSP. Adaptado de Correa González, Miguel Eduardo, Daniela Fernanda, Ortiz Delgado, and Edwin Alberto Garavito Hernández. 2017. "DAPFSP-SDST CON FÁBRICAS HETEROGÉNEAS 1 Solución Del Problema de Flowshop Distribuido y Permutado Con Etapa de Ensamble."

3.2.1. Flowshop Permutado (Permutation Flowshop Scheduling Problem PFSP).

Para encontrar la secuencia optima de producción de un Flowshop Permutado (PFSP) se asume que dicha secuencia es la misma para todas máquinas. Esta consideración se toma para reducir el campo de posibles soluciones. Desde que se demostró que el problema de Flowshop es un problema de NP-Hard cuando se trabaja con más de dos máquinas, y que los algoritmos exactos no están adaptados para programación a gran escala, se han venido proponiendo heurísticas y metaheurísticas que permiten la solución en tiempos computacionales aceptables para este problema (Jiang, Li, Hu, Hu, & Wei, 2018).

Hongyu He (2016), plantea como función objetivo del Flowshop Permutado, minimizar el máximo de tardanzas, considerando tiempos exponenciales que dependen de una tasa de aprendizaje. Para la solución se propone un algoritmo basado en la regla de fecha más próxima (EDD) y dos metaheurísticas basadas en revisiones de literatura (Algoritmo BJ y Algoritmo FL). Asimismo, se codifican en el software Visual Studio 2018 dos algoritmos, uno con enfoque Branch and Bound y el otro con Enfoque Basado en la Partición de Límites (BBNP).

El mismo método de solución de Branch and Bound se empleó nuevamente por Takano & Nagano (2017) para solucionar un PFSP, donde plantean como límite superior el tiempo de inactividad en la máquina y como límite inferior el tiempo de bloqueo. Este algoritmo es evaluado con 4 limites superiores distintos (LBTN1, LBTN2, LBTN3 y LBTN4), al comparar los 4 algoritmos con otros 540 problemas de una base de datos de la literatura, se escoge el que presenta mejores resultados, posteriormente este es contrastado con un modelo de programación lineal entera mixta (MILP).

Ese mismo año, 2017, en un artículo publicado por Benbouzid-Si Tayeb, Bessedik, Benbouzid, Cheurfi, & Blizak, se soluciona un *PFSP* mediante un algoritmo híbrido entre el algoritmo Genético Inmune y el Descendente Vacunado (*Genetic Immune Algorithm with Vaccinated Offspring, VacGA*). Con el fin de comparar la eficiencia de este algoritmo se codifica en el lenguaje C++ 4.0, posteriormente se compara con algoritmos genéticos convencionales recientes, lo que arroja como resultado una mayor eficiencia en el VacGA. Un año después, Long Wen soluciona un *PFSP* con tiempos de alistamiento, con la propuesta del algoritmo híbrido GAVNS, este algoritmo es una combinación entre un algoritmo Genético y un algoritmo de Búsqueda de Vecindario Variable. Para validarlo, GAVNS es comparado con otros existentes en la literatura

como el GA, ILS, GA_REEV, GA_AA, entre otros; esto da como resultado que el GAVNS presenta mejor efectividad en todas las instancias comparadas. (Peng, Wen, Li, Gao, & Li, 2018).

Por otra parte, un algoritmo híbrido es nuevamente usado por Chakravorty & Laha(2017), para la minimización del *makespan* de un PFSP. Se combinan los algoritmos recocido simulado (*Simulated Annealing, SA*) y algoritmo inmune (IA), generando el algoritmo HIA. Este algoritmo es comparado con instancias de distintos tamaños generadas previamente por el autor Taillard, las cuales varían entre 20 y 500 números de trabajos. Finalmente, después de la comparación, se arroja como resultado que HIA presenta un mejor rendimiento en al menos 13 algoritmos.

Con el objetivo de minimizar las tardanzas totales en un PFSP, Fernandez-Viagas, Valente, & Framinan (2018), propone dos tipos heurísticas: por un lado 8 variaciones del algoritmo voraz Iterado iterado(*Iterated Greedy Based Algorithm, IG*) y por otra parte un algoritmo basado en la búsqueda de haz (*Beam Search Initialization, BS*). Como conclusión se obtiene que la variación del algoritmo voraz iterado, IA_{RAS} (variante de intercambio aleatoria adyacente), supera estadísticamente a las otras 7 variaciones del IA y a la metaheurística BS.

Ahora bien, en el estudio del PFSP, han contemplado otras variantes para acercar el problema a determinadas condiciones de la industria. Rahman, Sarker, & Essam (2018), consideran que la programación del Flowshop no es un problema estático, si no por el contrario, es dinámico, es decir que la operación de un determinado trabajo puede interrumpirse. Para solucionar este tipo de problema, los autores presentan un Algoritmo Memético (*Memetic Algorithm, MA*), el cual maneja tanto pedidos únicos, como múltiples, estos a su vez pueden presentar interrupciones aleatorias. Este algoritmo es comparado con los algoritmos propuestos por el autor Taillard, tomando como criterio el Promedio de Porcentaje de Desviación (APD), se concluye que el MA presenta un mejor APD en la mayoría de los problemas.

$$APD = \sum_{i=1}^{R} \left(\frac{BS - BKS}{BKS} \times 100 \right) \Big/ IR \quad (1)$$

Donde:

BKS: mejor solución conocida

BS: solución generada por el método de solución

IR: Número de corridas independientes para cada instancia

3.2.2. Flowshop con ensamble (Assembly Flowshop Scheduling Problem AFSP). Este tipo de problema considera dos etapas: una de producción o elaboración de componentes y una segunda en donde una máquina ensambla las unidades fabricadas en la primera etapa (Sheikh, Komaki, & Kayvanfar, 2018). Un ejemplo de estudio de este tipo de Flowshop es el que muestra Kazemi, Mazdeh, & Rostami (2017), en el que, aparte de contemplar una etapa de ensamble, considera que después de esta fase se debe decidir el tamaño de lote a enviar al cliente H. En este sentido, se tiene la capacidad de enviar el trabajo inmediatamente que es ensamblado o esperar a que otros trabajos sean terminados, esto tomando en cuenta que existe un costo por trabajos tardíos. Para la solución de este, los autores plantean 2 metaheurísticas codificadas en el software MATLAB R2013, un algoritmo competitivo imperialista (*Imperialist Competitive Algorithm, ICA*) y un algoritmo híbrido del ICA (HICA). Asimismo, desarrollan un algoritmo de programación lineal entera (MILP) programado en el software GAMS. Como resultado se encontró que HICA es mejor que ICA, valorándolo desde el punto de vista del coeficiente RPD, sin embargo, el tiempo de ejecución de ICA es mejor y además el MILP presenta un alto tiempo de corrida respecto a los otros dos.

$$RPD = \frac{|Solución\ generada - Mejor\ solución| * 100}{|Mejor\ solución|} \quad (2)$$

Por otra parte, Komaki, Teymourian, Kayvanfar, & Booyavi (2017), presenta un Flowshop con tres etapas de ensamble, en donde los productos tienen *m* componentes y *m+2* operaciones. En la primera etapa son elaboradas las *m* componentes, en la segunda etapa es procesada la *m+1* operación y finalmente en la tercera etapa son ensambladas las componentes. Para la solución de este Flowshop, ellos proponen la versión discreta mejorada del algoritmo de Optimización de Cuckoo (*Improved Discrete Cuckoo Optimization Algorithm, IDCOA).* Los resultados son probados con la generación probabilística de instancias.

Asimismo, Basir, Mazdeh, & Namakshenas (2018), para solucionar un *AFSP*, propone un algoritmo Genético mejorado de dos niveles (*Bi-level Genetic Algorithm, IGA*), donde el primer nivel es encargado de resolver la secuencia de procesamiento, y el segundo asigna los trabajos a los lotes. Este algoritmo es implementado en el software Matlab 2014 y comparado con un algoritmo Genético (GA) también propuesto en el mismo artículo; como resultado el IGA arroja un porcentaje de error (RPE) menor que el GA, en todos los experimentos realizados.

Ahora bien, la etapa de ensamble no solo ha sido implementada para solucionar el problema de Flowshop Permutado, sino también en otro tipo de variantes, como es el caso de Lv & Lei (2018), quien soluciona un Flowshop híbrido (HFSP) con el objetivo de minimizar simultáneamente las tardanzas totales y el *makespan*; para esto, se implementa en el software Microsoft Visual C++ 6.0, un algoritmo de nueva búsqueda en el vecindario con intercambio global *(A novel Neighborhood Search with Global Exchange, NSG*).

Finalmente, Liu et al. (2018), proponen la solución de un Flowshop con tres máquinas: M1 y M2 para la primera etapa, encargadas de la elaboración de partes, y una tercera máquina, M3, destinada al ensamble de componentes. Su propuesta se basa en encontrar la secuencia de producción para estas tres máquinas, con el objetivo de minimizar el tiempo de flujo; como método de solución plantean seis algoritmos híbridos de Optimización de Enjambre de Partículas (*Hybrid of Particle Swarm Optimization Algorithm, PSO*), codificados en el software FORTRAN y además propone un algoritmo de Branch and Bound. Para la comparación de los algoritmos hacen uso del análisis de varianza ANOVA y posteriormente a este análisis se desarrollaron pruebas de Fisher para las instancias que no presentan diferencias significativas.

3.2.3. Flowshop Distribuido y Permutado (Ditributed Permutation Flowshop Scheduling Problem DPFSP). En la actualidad, muchas compañías no tienen solo una fábrica destinada a la producción; por el contrario, para la elaboración del producto final se necesita el funcionamiento de más de una fábrica; con la intención de adaptar este problema a estas compañías se origina el DFSP, el cual busca encontrar la secuencia de producción en cada fábrica.

No obstante, algunos autores han trabajado en conjunto el PFSP y el DFSP, originando de esta manera un problema de *n* trabajos, *m* máquinas y *f* fábricas, como es el caso de Bargaoui, Belkahla Driss, & Ghédira (2017), quienes, solucionan un DPFSP, a través de la metaheurística de la Optimización de la Reacción Química (*Chemical Reaction Optimization, CRO*). Este es programado en lenguaje C++ en el software Visual Studio 2015 y comparado con un algoritmo de Búsqueda en la Vecindad Variable Descendente (VND) y la heurística NEH2, concluyendo, que el CRO produce mejores resultados que los otros algoritmos.

Los mismos autores vuelven a utilizar la metaheurística CRO, para solucionar nuevamente un DPFSP, pero esta vez la combinan con técnicas de inteligencia artificial distribuidas, conocidas como sistemas multivalentes (MAS), para formar el algoritmo híbrido CROMAS. Este es implementado usando el lenguaje JADE y a su vez es comparado según el índice de porcentaje de desviación (RPD), con las heurísticas NEH2, HDCS y con un algoritmo genético. Se obtiene, que el CROMAS supera en el 60% de las instancias al algoritmo genético y en un 87% al HDCS. Finalmente, el algoritmo NEH2, es superado por las otras 3 metaheurísticas, sin embargo, es un algoritmo bastante rápido y simple (Bargaoui, Driss, & Ghédira, 2017).

No obstante, el DFSP no solo ha sido adaptado al PFSP, como es la propuesta de Weishi Shao, quien plantea iteraciones de algoritmos voraces basados en la Búsqueda de Vecindario Variable (VNS), Búsqueda en Vecindario Variable Descendente VND y la Estructura del Vecindario Probabilístico (*Random Neigborhood Structure, RNS*), estos algoritmos se evalúan y se comparan respecto a criterios de efectividad y de capacidad de búsqueda, al momento de resolver un problema de Flowshop Distribuido sin esperas (DNWFSP)(W. Shao, Pi, & Shao, 2017).

Por otra parte, Rifai, Nguyen, & Dawal (2016), planean encontrar el número óptimo de fábricas, los trabajos asignados para cada fábrica y para cada máquina, y posteriormente, la secuencia de producción para un Flowshop Permutado Distribuido con reentrantes. Esto con el objetivo de minimizar simultáneamente, el *makespan*, el costo total y el promedio de tardanzas; para ello, se propone un algoritmo Multiobjetivo Adaptado a la Búsqueda en el Vecindario (*Multi-objetive Adaptive Large Neighborhood Search, MOALNS*), basado en el principio de Pareto. Este algoritmo es comparado con el algoritmo de Gran Búsqueda en la Vecindad (LNS), y el algoritmo Genético de Clasificación no Dominado (NSGA-II); como resultado, se obtiene que el MOALNS presenta

mejores resultados en indicadores como la tasa de mejora (IR) y en el índice relativo de desviación (RDI (γ)).

$$RDI(\gamma) = \frac{fi(\gamma) - fib}{fiw - fib} \quad (3)$$

Donde:

Fi (γ) el resultado obtenido por el algortimo γ

Fib: mejor valor objetivo de la función

Fiw: peor valor objetivo de la función

En la Tabla 6 se puede observar los tipos de algoritmos que han sido utilizados para solucionar un problema de tipo DPFSP.

Tabla 6.

Algoritmos para solucionar el DPFSP

Año	***Autores***	***Función objetivo***	***Algoritmo propuesto***
2017	Hafewa Bargaoui, Olfa Belkahla Driss, Khaled Ghédiraa	Min: *makespan*	CROMAS: híbrido entre Optimización de una reacción química (CRO) y Sistemas multi-agentes (MAS)
2017	Hafewa Bargaoui, Olfa Belkahla Driss, Khaled Ghédiraa	Min: *makespan*	CRO-Optimización de una reacción química

Continuación Tabla 6

Algoritmos para solucionar el DPFSP

Año	***Autores***	***Función objetivo***	***Algoritmo propuesto***
2016	Jin Deng, Ling Wang	Min: *makespan* y tardanzas totales	CMA-Algoritmo competitivo memético
2015	Jin Deng, Ling Wang, Sheng-yao Wang and Xiao-long Zheng	Min: *makespan*	CMA-Algoritmo competitivo memético
2015	Sheng-Yao Wang and Ling Wang	Min: *makespan*	EDAMA- híbrido entre Algoritmo de estimación de distribución (EDA) y algoritmo memético (MA)

3.2.4. Tiempos de alistamiento dependientes de la secuencia (Sequence Dependent Setup Times SDST).

Los tiempos de alistamiento son aquellas operaciones no productivas que deben ser realizadas previamente a la fase de producción, como la limpieza o mantenimiento de máquinas, preparación de herramientas, entre otros (Z. Shao, Pi, & Shao, 2018). Si se busca la minimización del *makespan*, es importante contemplar estos tiempos de alistamiento en la solución de un Flowshop, ya que muchas veces estos tiempos dependen de la secuencia de producción.

Xu, Wu, Yin, & Lin (2017), plantean la solución de Flowshop Permutado Multi objetivo, contemplando tiempos de alistamiento dependientes de la secuencia (MOPFSP-SDST). Para la solución, se propone un algoritmo de Búsqueda Iterada Multi Objetivo (*Iterated Local Search for*

the Multi-Objetive, MOILS), codificándolo en el lenguaje C++. Más adelante, Sioud & Gagné (2018) proponen un algoritmo mejorado de optimización de aves migratorias (*Enhanced Migrating Birds Optimization Algorithm, EMBO*) para la solución de un Flowshop permutado con el objetivo de minimizar el *makespan*, EMBO es codificado en el lenguaje C++ y es comparado con otras metaheurísticas existentes en la literatura como AHA, HGA, VND, IMBO, IG; concluyendo que EMBO presenta mejores resultados al comparar instancias.

Posteriormente, Li, Yang, Ruiz, Chen, & Sui (2018) propone un algoritmo voraz para la solución de un Flowshop sin esperas (NWFSP) y con tiempos de alistamiento dependientes de la secuencia. Las pruebas computacionales y comparaciones estadísticas, usando ANOVA, arrojan que este algoritmo presenta mejores resultados que otras metaheurísticas existentes para solucionar este tipo de problema. Un año más tarde Ying & Lin (2018) presentan un algoritmo con dos fases metaheurísticas (TPM) para solucionar un NWFSP. Como resultado, este algoritmo presenta una mejora sobre otros algoritmos existentes en la literatura debido a que concede instancias grandes en tiempos computacionales aceptables.

Finalmente es importante resaltar el trabajo realizado por Deng, Wang, Wang, & Zheng (2016), el cual plantea mediante un algoritmo Memético Competitivo (CMA) la solución de un Flowshop Distribuido con dos etapas de ensamble (DTSAFSP), para la minimización del *makespan*. Este algoritmo es comparado con uno de Búsqueda en la Vecindad Variable (*Variable Neighborhood Search, VNS*) y un híbrido entre un algoritmo Genético y un reducido de VNS generando el GA-RVNS, arrojando mejores resultados computacionales para el CMA.

4. Marco Teórico

4.1.Flowshop (FSP)

Un Flowshop es un problema de optimización combinatoria que consiste en *m* máquinas que se encuentran ubicadas en serie, donde hay *n* trabajos que deben ser procesados en cada una de estas (ver Figura 7). Dichos trabajos deben pasar consecutivamente de máquina en máquina hasta llegar a la última, cumpliendo la secuencia de producción (González-Neira, Montoya-Torres, & Barrera, 2017). Asimismo, se conoce este, como un modelo simplificado del problema de programación de la producción en línea, del cual existen varias adaptaciones que consideran diferentes factores, parámetros y variables. Esto, con el fin de dar solución a situaciones que representen entornos más complejos de la manufactura actual en industrias como la de procesamiento de alimentos, producción de plásticos, entre otros. Las variantes del problema más conocidas son el PFSP (Flowshop permutado), DPFSP (Flowshop Distribuido y Permutado), DAPPFSP (Flowshop Distribuido y Permutado con etapa de Ensamble).

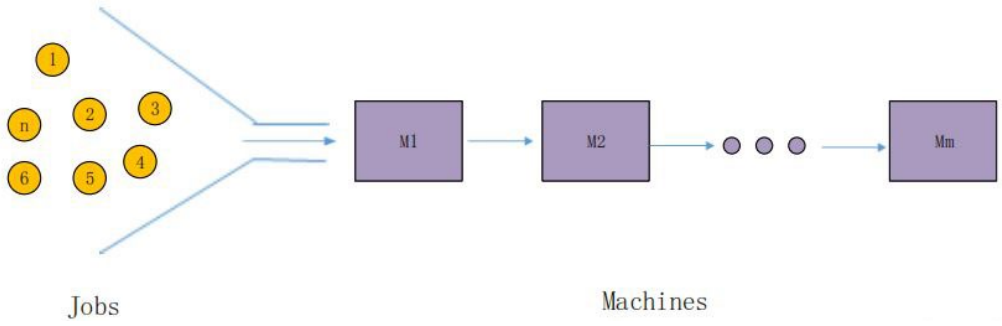


Figura 7. Representación del proceso en un Flowshop. Adaptado de Jiang, H., Li, S., Hu, J., Hu, J., & Wei, H. (2018). Hybrid Genetic Simulated Annealing Algorithm for Improved Flowshop Scheduling with *makespan* Criterion. Applied Sciences, 8(12), 2621. https://doi.org/10.3390/app8122621

El FSP generalmente presenta algunas asunciones (Naderi & Ruiz, 2010):

- Todos los trabajos son independientes y tienen disponibilidad para ser procesados en el tiempo 0.
- Las máquinas s están continuamente disponibles.
- Cada máquina puede procesar solo un trabajo al tiempo.
- Cada trabajo puede ser procesado en una sola máquina al tiempo.
- Una vez que el procesamiento de un trabajo específico ha comenzado en una máquina específica, no se puede interrumpir, y el procesamiento continúa hasta que se complete.
- Los tiempos de alistamientos son independientes de la secuencia y son incluidos en el tiempo de procesamiento, o ignorados.
- El almacenamiento de producto en proceso infinito está permitido.
- Las máquinas están continuamente disponibles.

4.2. Flowshop Permutado (PSFP)

Un Flowshop permutado es un sistema de producción en donde los trabajos siguen un flujo por todas las máquinas en el mismo orden, problema que es muy común en los sistemas de producción intermitentes, es decir, en sistemas que tienen la capacidad de fabricar varios productos y realizar manufactura de lotes pequeños (Fuchigami, Sarker, & Rangel, 2018). Asimismo, el PSFP clásico consiste en n trabajos que deben ser llevados a cabo en m máquinas, en donde generalmente el objetivo de optimización es minimizar el makespan (Zhang, Tong, & Ma, 2014). Para encontrar solución a este tipo de problema, se han establecido diferentes propuestas algorítmicas, tales como optimización por colonia de hormigas, algoritmo genético, algoritmo voraz, algoritmo VND, algoritmo de búsqueda tabú, entre otras.

Generalmente el PFSP, toma en cuenta ciertas asunciones de capacidad (restricciones) del FSP, una de ellas es que cada máquina procesa solo un trabajo a la vez y la otra que cada trabajo puede ser procesado por una sola máquina al tiempo. Zobolas, Tarantilis, & Ioannou (2009) mencionan que de acuerdo con el número de trabajos (*n*) y el número de máquinas(*m*), la cantidad de instancias del Flowshop son designadas por la expresión (1), dónde *N* representa el número de instancias:

$$N = n * m \tag{4}$$

4.3.Flowshop Permutado y Distribuido (DPFSP)

Este problema es una adaptación del problema PFSP, en donde se tienen *n* trabajos que deben ser procesados en un número *f* de fábricas que son idénticas y que contienen el mismo grupo de *m* máquinas(Naderi & Ruiz, 2010). Generalmente el criterio de optimización en este tipo de Flowshop se centra en minimizar el tiempo de terminación o *makespan* entre fábricas.

Naderi & Ruiz (2010) proponen una forma de calcular el número de posibles soluciones para este problema. En este caso se denota f_h como el número de trabajos asignados a la fábrica h, $h = \{1,2, \dots, F\}$, donde lo *n* trabajos tienen que ser repartidos en las *F* fábricas, por lo tanto, se establece que para la primera fábrica f_1 se tiene $\binom{n}{f_1}$ combinaciones posibles, por lo tanto al hacer un proceso de análisis con las *F* fábricas restantes se obtiene que el número total de soluciones es:

$$\binom{n}{f_1} f_1! * \binom{n - f_2}{f_2} f_2! * \dots * \left(n - \sum_{h-1}^{F-1} fh \right) f_F! \tag{5}$$

4.4. Flowshop Permutado y Distribuido con etapa de Ensamble (DAPFSP)

El DAPFSP es una combinación entre el Flowshop Distribuido y Permutado (DPFSP) y el problema de Flowshop con etapa de Ensamble. Este problema consiste básicamente en dos etapas, una etapa de producción y otra de ensamble (ver Figura 8) además, este puede ser generalizado en tres subproblemas, programación de trabajos, programación de producto y asignación de fábricas (Lin, Wang, & Li, 2017).

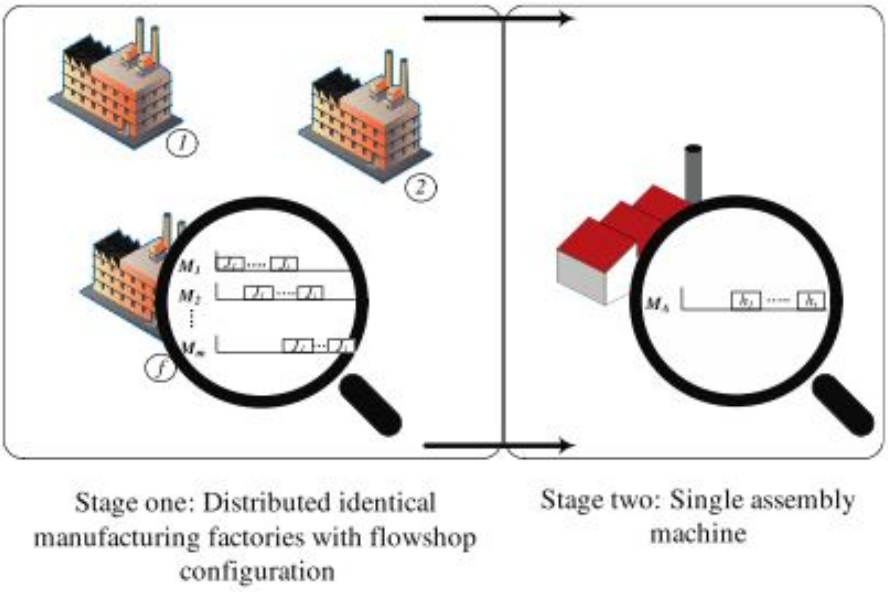


Figura 8. Representación gráfica de DAPFSP. Adaptado de Gonzalez-Neira, E. M., Ferone, D., Hatami, S., & Juan, A. A. (2017). A biased-randomized simheuristic for the distributed assembly permutation Flowshop problem with stochastic processing times. Simulation Modelling Practice and Theory, 79(November), 23–36. https://doi.org/10.1016/j.simpat.2017.09.001

4.5.Tiempos de alistamiento (Setup Times)

Los tiempos de alistamiento (Setup Times) en general, pueden ser definidos como el tiempo que requiere preparar un recurso que es necesario para llevar a cabo una tarea (Allahverdi & Soroush,

2008). Estos tiempos, juegan un papel importante dentro la manufactura actual en las fábricas, debido a que tienen una amplia influencia en las entregas a tiempo del producto. Además, dichos tiempos se ven representados en el costo que implica la preparación de los recursos antes de comenzar una tarea, donde se incluyen factores como el tiempo que se requiere para limpiar la máquina, para cargarla, realizarle ajustes o cambiar la herramienta (Zhang et al., 2014).

4.6. Makespan

El *makespan* está definido en la literatura como el máximo tiempo de completamiento de las tareas a procesar en el sistema, es decir el tiempo comprendido entre el primer trabajo en la primera máquina y el último trabajo en la última máquina. La minimización del *makespan* es considerado como criterio un de optimización, y es una de las medidas de desempeño utilizadas con mayor frecuencia por muchos autores. Además, es también utilizado como indicador de control de piso.(Low, Hsu, & Su, 2008)(Mansouri, Aktas, & Besikci, 2016)(Alberto & Hernandez, 2017)(Peña-tibaduiza, Garavito-hernández, Moratto-chimenty, & Pérez-figueredo, 2014).

4.7.Programación lineal entera mixta (MILP)

Este Método de programación matemática, se caracteriza porque dentro de sus condiciones está que tanto las restricciones, como la función objetivo deben tener una relación lineal y algunas de las variables deben tomar valores enteros o binarios. En la MILP es posible manejar variables continuas o fraccionadas y que sean no negativas. Este método puede ser usado para encontrar la solución óptima de pequeños y medianos problemas, ya que no es tan efectivo al usar una gran carga computacional. Algunos de los métodos exactos de solución más usados son Branch and Bound, Branch and Cut, relajación lagrangiana y planos de corte(Jian, Pan, & Yang, 2019).

4.8.Heurísticas

A continuación, se presentan algunas heurísticas usadas para dar solución a diferentes problemas de optimización.

4.8.1. Algoritmo de búsqueda local. Esta heurística es usada para resolver problemas de optimización, donde el objetivo es minimizar o maximizar la función objetivo-asociada al problema. Este algoritmo asume una estructura de búsqueda en el vecindario. Tiene inicio con una población inicial generada a partir de otro algoritmo o aleatoriamente, a partir de este momento se generan movimientos a otros vecindarios. La mejor solución de cada vecindario se denomina óptimo local, esta es guardada antes de generar un nuevo movimiento, el algoritmo termina cuando no se encuentra una mejor solución, lo cual se le denomina, óptimo global, lo que significa encontrar la solución óptima para toda la población (Selman, Kautz, & Cohen, 1994).

La calidad de la solución óptima para un algoritmo de búsqueda local depende de la complejidad del problema. Ya que esta esta heurística no es recomendada para problemas de gran cantidad de datos, surgen metaheurísticas que son más eficientes en el manejo de big data, evitando caer en óptimos locales.

4.8.2. Métodos de descomposición. Son aquellos algoritmos que para su solución descompone en subproblemas el problema original, los cuales son más sencillos de resolver, teniendo en cuenta que todos pertenecen al mismo problema.

4.8.3. Métodos de reducción. Este algoritmo identifica aquellas propiedades que cumplen las buenas soluciones y las introduce como restricciones del problema, el objetivo es reducir la cantidad de posibles soluciones, logrando simplificar el problema. Es este método se tiene el riesgo de eliminar soluciones optimas del problema original.

4.9.Metaheurísticas

A continuación, se presentan algunas heurísticas usadas para dar solución a diferentes problemas de optimización.

4.9.1. Algoritmo genético. Los algoritmos genéticos o evolutivos son métodos robustos de búsqueda que se asemejan a la teoría o modelo de evolución propuesto por Charles Darwin; estos algoritmos permiten tratar problemas de optimización emulando el proceso evolutivo de las poblaciones, donde participan parámetros que tienen como objetivo encontrar la mejor solución entre un grupo de soluciones que han sido sometidas a cruces y mutaciones, para maximizar o minimizar una función de adaptación o función *fitness*.

Chu- Beasley propuso por primera vez en el año 1997 el uso de algoritmos genéticos para la solución de problemas de optimización, esto debido a la calidad en las respuestas y el alto desempeño computacional que proporciona en diversos problemas. Una de las características más importantes de este algoritmo, es que garantiza que se mantenga el tamaño de la población constante en cada iteración y modificación a la que es sometida.

Algunas de las características de los algoritmos genéticos que permiten identificar una diferencia con los métodos tradicionales de optimización son según Valencia (2014):

- Codificación del conjunto de parámetros, debido a que en general no se evolucionan los parámetros, sino que estos son codificados.
- Búsqueda en paralelo con una población de puntos.
- Uso de una función de una función de adaptación directamente.
- Existencia de reglas de transición probabilísticas entre una interacción y otra.

Las etapas de un algoritmo genético se presentan en la Figura 9, que se muestra a continuación.

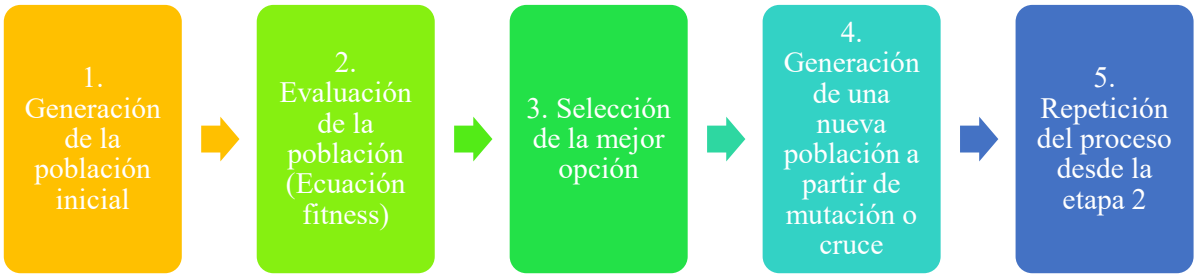


Figura 9. Etapas del algoritmo genético

4.9.1.1. Operador de selección. La selección es una de las fases que conforma al algoritmo Genético, en donde básicamente se busca asignar el mayor número de copias idénticas al padre más apto. Esto con el fin de generar un contraste con los individuos que presentan una adaptación menor al problema. Este método es conocido como *selección proporcional* (Valencia, 2014).

Existe una manera de identificar la probabilidad de selección $p_{i,t}$ y el número esperado de copias N_e de los individuos a partir de la adaptación en la población.

$$p_{i,t} = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{6}$$

$$N_e[i] = M p_{i,t=} \frac{f_i}{\overline{f_t}} \tag{7}$$

Donde f_j es la adaptación del j. ésimo individuo, $\overline{f_t}$ es la adaptación promedio población y M es el tamaño de esta. Por otra parte, dentro de la selección se encuentran múltiples métodos, que son utilizados dependiendo de la forma en que se adecuan al problema tratado. (ver Tabla 7)

Tabla 7.

Métodos de selección.

Método	Descripción
Ruleta	A cada individuo de la población (cromosoma) se le asigna una parte de la ruleta en forma de porción. De esta manera, los mejores cromosomas reciben una porción mayor dentro de la ruleta, garantizando así la asignación de una probabilidad de selección más alta.
Muestreo Estocástico	Se implementa en una sola fase. Es una modificación a la selección por ruleta. De esta manera el operador es aplicado como en la ruleta, con la diferencia de que la asignación de los individuos a esta se hace con una proporción que es proporcional a valor de la función objetivo para cada cromosoma.
Torneo	Se caracteriza por realizar comparaciones directas entre individuos. Existe la selección determinística, que consiste en seleccionar un número *p* de individuos con el fin de escoger al cromosoma más apto para pasar a la siguiente generación. Por otra parte, en la selección probabilística, se fija un parámetro como punto de referencia, p

Nota: información adaptada de Valencia, P. E. (2014). Optimización Mediante Algoritmos Genéticos. Instituto de Ingenieros de Chile, (May), 83–92.

4.9.1.2. Operador de cruce. El cruce en un algoritmo genético hace referencia a la reproducción sexual existente en la teoría de evolución. Normalmente, este operador trabaja con dos cromosomas al tiempo, los cuales generan dos descendientes. Estos descendientes cuentas con características combinadas de los cromosomas de los dos padres.

Existen múltiples tipos de operadores de cruce, que se aplican según el problema de optimización a tratar. A continuación, se muestran algunos de estos:

Tabla 8

Tipos de operadores de cruce

Método	Descripción
Cruce de punto	A partir de dos padres, se define una posición al azar de cruce. Donde, el hijo 1 hereda los genes que están antes del punto de corte en el padre 1 y el hijo 2 hereda la parte restante. Posteriormente se rellenan las otras partes de los hijos con los genes del padre 2.
Cruce de dos puntos por orden	Este tipo de cruce se caracteriza por funcionar de igual manera que el cruce de punto. Sin embargo, la diferencia es que existen dos puntos de cruce aleatorios, por lo tanto, las secciones que se encuentran por fuera de estos dos puntos son heredadas de forma idéntica a los hijos que corresponden respectivamente a cada padre. De esta manera, el resto de las casillas dentro de los cromosomas de los hijos, se rellenan con los genes de los padres opuestos.
PMX "Partially Matched Crossover"	En este caso, se cuenta con dos puntos de corte. Dónde, los genes que son heredados a los hijos correspondiente son aquellos que se encuentran dentro de los dos puntos de cruce. Por otra parte, las casillas restantes son rellenadas con los genes del padre opuesto.
UOB "Uniform Order Based Crossover"	En este caso, se asigna un vector binario a cada padre, y los genes que correspondan a los valores uno de cada padre es heredado a sus hijos directos, en la misma posición, los demás que tienen el valor 0, se asignan al hijo opuesto en su orden relativo.

Continuación Tabla 8

Tipos de operadores de cruce

Método	Descripción
SBOX "Similar Block Order Crossover"	Este tipo de cruce se caracteriza por copiar de forma idéntica, los bloques de más de dos genes que se repiten en los padres a cruzar. Posteriormente, se obtienen dos puntos de cruce y se hereda al hijo directo los genes contenidos entre los dos puntos, los demás genes son transferidos de cada padre al hijo opuesto en su orden relativo.

Nota: información adaptada de Valencia, P. E. (2014). Optimización Mediante Algoritmos Genéticos. Instituto de Ingenieros de Chile, (May), 83–92.

4.9.1.3. Operador de mutación. La mutación es uno de los operadores básicos del GA, el cual permite que la población tenga mayor aleatoriedad. Este operador, se encarga principalmente de aumentar o reducir el espacio de búsqueda de posibles soluciones.

Ruiz en el 2003 menciona varios operadores de mutación para tener en cuenta a la hora de tratar problemas de Flowshop con un algoritmo genético. Estos son, mutación *por intercambio (SWAP mutation)*, donde se intercalan las posiciones de dos alelos. Por otra parte, menciona la *mutación por intercambio adyacente (POSITION mutation),* que se caracteriza por generar una limitación para la mutación *Swap*, debido a que cuenta con la condición de realizar el intercambio de las posiciones en caso de que los alelos del vector se encuentren adyacentes. Finalmente, enuncia la mutación con más influencia en los algoritmos genéticos, *mutación por desplazamiento (SHIFT mutation),* la cual consiste en realizar el desplazamiento de un bloque de alelos consecutivos a una posición diferente dentro del vector.

4.9.2. Algoritmo Voraz. Un algoritmo voraz tiene como función principal elegir la solución óptima para un subproblema local y así poder llegar a una solución óptima general sin la

necesidad de reconsiderar soluciones anteriores. Estos algoritmos se caracterizan por contar con tres elementos; un grupo de candidatos que pueden formar parte de la solución, un subconjunto conocido como solución factible, conformado por los candidatos que satisfacen las restricciones del problema y una solución óptima, que es la solución factible que cumple con el criterio de la función objetivo.

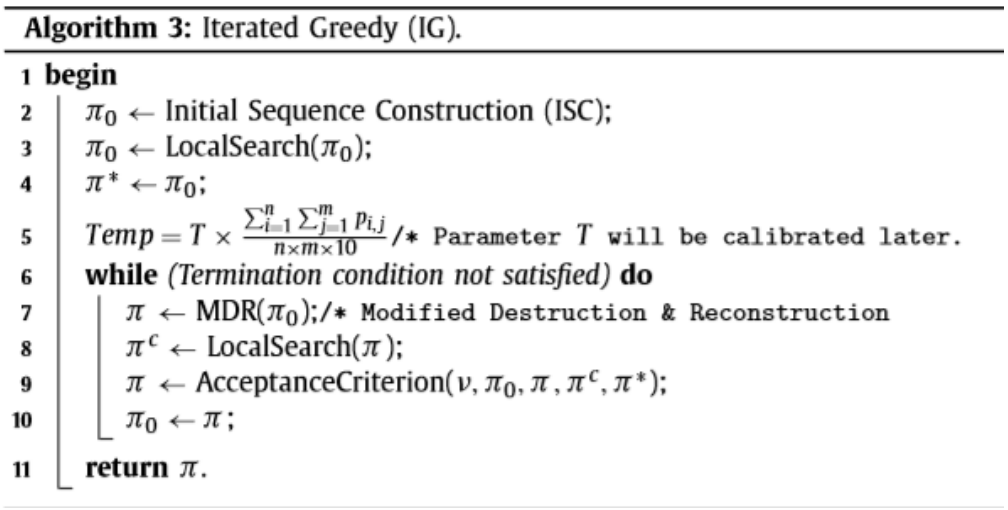
Las etapas de un algoritmo voraz se presentan en la Figura 10, que se muestra a continuación.

Figura 10. Etapas del algoritmo voraz. Adaptado de Shao, W., Pi, D., & Shao, Z. (2017). Optimization of *makespan* for the distributed no-wait *flowshop* scheduling problem with iterated greedy algorithms. Knowledge-Based Systems, 137, 163–181. https://doi.org/10.1016/j.knosys.2017.09.026

El algoritmo voraz más común es el algoritmo iterado Greedy, el cual se desarrolla en 4 pasos principales (W. Shao et al., 2017)

- Solución inicial, la cual puede ser generada con alguna heurística
- Destrucción y construcción, en este paso se selecciona aleatoriamente un elemento de la secuencia, o solución y se inserta en otra posición
- Búsqueda local, que es aplicada a la solución incumbente
- Criterio de aceptación, se usa para determinar si alguna nueva solución reemplaza a la solución incumbente

A continuación, en la Figura 11 se presenta el pseudocódigo del algoritmo voraz iterativo.



Algorithm 3: Iterated Greedy (IG).

1 **begin**
2 $\pi_0 \leftarrow$ Initial Sequence Construction (ISC);
3 $\pi_0 \leftarrow \text{LocalSearch}(\pi_0)$;
4 $\pi^* \leftarrow \pi_0$;
5 $Temp = T \times \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} p_{i,j}}{n \times m \times 10}$ /* Parameter T will be calibrated later.
6 **while** *(Termination condition not satisfied)* **do**
7 $\pi \leftarrow \text{MDR}(\pi_0)$;/* Modified Destruction & Reconstruction
8 $\pi^c \leftarrow \text{LocalSearch}(\pi)$;
9 $\pi \leftarrow \text{AcceptanceCriterion}(v, \pi_0, \pi, \pi^c, \pi^*)$;
10 $\pi_0 \leftarrow \pi$;
11 **return** π.

Figura 11. Pseudocódigo del algoritmo voraz iterativo. Información tomada de Li, X., Yang, Z., Ruiz, R., Chen, T., & Sui, S. (2018). An iterated greedy heuristic for no-wait *flowshop*s with sequence dependent setup times, learning and forgetting effects. *Information Sciences*, *453*, 408–425. https://doi.org/10.1016/j.ins.2018.04.038

4.9.3. Algoritmo VNS. La búsqueda de entorno variable (Variable Neighbourhood Search, VNS) es una metaheurística, que está basada en cambiar sistemáticamente de estructura de entornos dentro del proceso de búsqueda, con el fin de cumplir con el objetivo del problema. La VNS está basada en tres hechos simples propuestos por Hansen, Mladenovic, & Moreno Perez (2003):

- Un mínimo local con una estructura de entornos no lo es necesariamente con otra.
- Un mínimo global es mínimo local con todas las posibles estructuras de entornos.
- Para muchos problemas, los mínimos locales con la misma o distinta estructura de entornos están relativamente cerca.

4.9.3.1. VNS Descendente (VND). Esta variación del VNS, que se conoce como VND por contar con la característica de hacer una búsqueda descendente, consiste en remplazar iterativamente la solución local existente, por una solución nueva, que es el resultado de una

búsqueda local, mientras se encuentra una solución mejor en un entorno diferente (Garrucho, 2014).

Esta búsqueda local se desarrolla mediante un conjunto de estructuras, las tres más comunes y usadas en la literatura son: inserción y swap y 2-opt. La inserción selecciona un elemento y lo reinserta en una nueva posición, mientras el swap escoge 2 elementos e intercambia sus posiciones, estas operaciones se realizan n veces, hasta que se cumple con un criterio de parada. Por último 2-opt se encarga de seleccionar 2 rutas o secuencias y cruzarlas y reordenarlas. Las primeras dos estructuras se encargan de la búsqueda descendente y la tercera se encarga de que el algoritmo no quede atrapado en un óptimo (Chen, Huang, & Dong, 2010).

A continuación, en la Figura 12 se muestra las fases o pasos que sigue un VND.

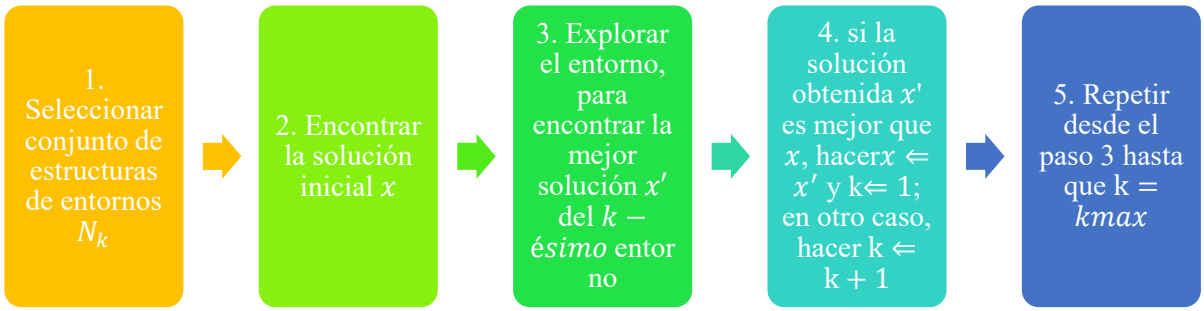


Figura 12. Estructura general del algoritmo VND. Adaptado de Chen, P., Huang, H. kuan, & Dong, X. Y. (2010). Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem. Expert Systems with Applications, 37(2), 1620–1627. https://doi.org/10.1016/j.eswa.2009.06.047

El pseudocódigo para el algoritmo VND se presenta en la Figura 13 que se encuentra a continuación.

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

***Repetir** hasta que no se obtenga mejora:*

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

Figura 13. Pseudocódigo algoritmo VND. Tomado de Glover, F. (2016). Handbook of meta-heuristics, (January 2003).

4.9.4. Recocido Simulado. El Recocido Simulado (RS) es un algoritmo que ayuda a evitar que el algoritmo en su búsqueda resulte en un óptimo local. Esto se lleva a cabo cuando se permite que algunos de los movimientos dentro del algoritmo se dirijan hacia soluciones menos buenas o incluso peores. Asimismo, los movimientos son controlados con una función de probabilidad, que se encarga de disminuir la probabilidad de realizar esos movimientos, a medida que la búsqueda avanza (Dowsland & Diaz, 2003)

4.10. Complejidad computacional

Los problemas de optimización generalmente son clasificados por su complejidad computacional (ver Tabla 9). De esta manera, la dificultad que existe en estos se mide con la cantidad de recursos computacionales necesarios para llegar a una solución. El tiempo es uno de los recursos que es propuesto en la teoría de complejidad computacional para realizar la respectiva clasificación Garey, M., & Jhonson, D. (1979).

Tabla 9.
Clases de complejidad computacional

Clase de complejidad	Decripción
P	Problemas que pueden ser resueltos en un tiempo polinómico haciendo uso de una máquina de Turing[5] determinista.
NP	Este grupo esta conformado por problemas en los que el algoritmo no puede ser resuelto en tiempo polinomial, pero puede ser solucionado con una Máquina de Turing no determinista.
NP – Complete	Pertenencen al grupo de problemas NP, conocido como el subgrupo de problemas más dificiles de resolver, por lo tanto existen menos probabilidades de encontrar una solución en un tiempo polinomial.
NP – Hard	Este subgrupo de problemas que pertenece a los NP, se destaca por ser al menos igual de díficil al grupo NP, con la caracteristica diferenciadora de no tener definida su complejidad, generando así que sea más dificil de solucionar que un problema de tipo NP- Complete.

Nota: información adaptada de Chavez Cruz, Marco Antonio, Pedro Moreno Bernal, and Jesus del Carmen Peralta Abarca. 2004. "Aplicación de La Teoría de La Complejidad En Optimización Combinatoria." Narraciones De La Ciencia Y La Tecnología 0(20): 32–42. http://inventio.uaem.mx/index.php/inventio/article/view/273.

4.11. Optimización

[5] La máquina de Turing, inventada por el matemático inglés Alan M. Turing en 1937, es una máquina autómata artesanal que se utiliza para clasificar los problemas de acuerdo con el tipo de máquina que puede existir para resolverlos (Aplicación de la teoría de la complejidad en optimización combinatoria)

La optimización forma parte importante de la investigación de operaciones. Dentro de esta área existen gran cantidad de problemas que tienen la necesidad de ser solucionados haciendo uso de métodos de programación y algoritmia (ver Figura 14). Asimismo, un problema de optimización se caracteriza por enfocarse en encontrar, dentro de un conjunto *X* de soluciones factibles, una solución que optimice la función objetivo *f(x)*.

$$min\{f(x)|x \in X\} \; o \; max\{f(x)|x \in X\} \tag{8}$$

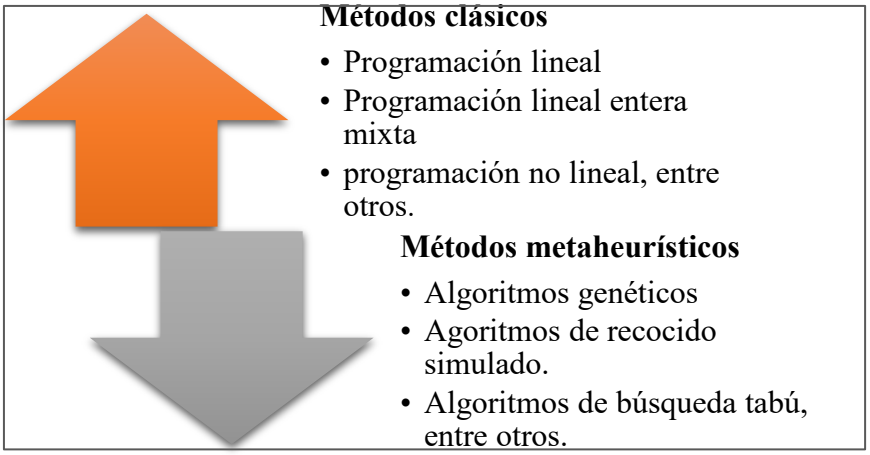


Figura 14.Métodos de la optimización. Adaptado de Ramos, A., Sánchez, P., Ferrer, J. M., Barquín, J., & Linares, P. (2010). Modelos Matemáticos de Optimización. Universidad Pontifica de Comillas.

Para resolver un problema de optimización es necesario encontrar el valor que deben tomar las variables con el fin de lograr que la función objetivo sea optima, teniendo en cuenta el conjunto de restricciones (Ramos, Sánchez, Ferrer, Barquín, & Linares, 2010). Asimismo, estos problemas presentan tres componentes importantes; la función objetivo, las variables y las restricciones (ver Tabla 10).

Tabla 10

Componentes de un problema de optimización

Componente	Descripción
Función Objetivo	Representa la medida cuantitativa del sistema que se desea optimizar (maximizar o minimizar), como por ejemplo minimizar los costos de producción
Variables	Hacen referencia a las decisiones que pueden ser tomadas para tener influencia en la función objetivo. Existen variables de control y variables de estado.
Restricciones	Son las ecuaciones e inecuaciones que representan las relaciones que deben ser satisfechas por ciertas variables del problema

Nota: Información adaptado de Ramos, A., Sánchez, P., Ferrer, J. M., Barquín, J., & Linares, P. (2010). Modelos Matemáticos de Optimización. Universidad Pontifica de Comillas.

4.11.1. Optimización combinatoria. La optimización combinatoria se define como un problema P = (S, f.), donde existe:

- Variables $X = \{X_1, \dots, x_n\}$;
- Variable dominante $D_1, \dots, Dn$;
- Restricciones entre variables;
- Una función objetivo f que debe ser minimizada o maximizada, donde $f: D_1 x \dots x D_n \rightarrow I_R^+$;

El conjunto de posibles soluciones factibles este definido por:

$$S = \{s = \{(x_1, v_1), \dots, (x_n, v_n)\} | V_i \in D_{i,s} \; que \; satisface \; todas \; las \; restricciones\} \qquad (9)$$

Usualmente S es conocido como el espacio de solución, donde cada elemento del conjunto debe ser visto como un candidato a ser solución. Para resolver el problema de optimización combinatoria, se debe encontrar una solución $s^* \in S$ que cumpla con la función objetivo, logrando el valor mínimo o máximo que se pueda obtener de esta.

4.12. Análisis de varianza ANOVA

Es una herramienta estadística que aplica conceptos de regresión lineal para determinar si existen diferencias significativas entre las medias poblacionales o si por el contrario estas no difieren. Asimismo, esta herramienta tiene amplias aplicaciones en las industrias en lo que concierne al control estadístico de los procesos.

Se debe utilizar tres tipos de hipótesis para hacer un uso satisfactorio de la ANOVA (Gibergans Bàguena, 2009):

- Cada conjunto de datos debe ser independiente del resto.
- Los resultados obtenidos para cada conjunto deben seguir una distribución normal.
- Las varianzas de cada conjunto de datos no deben diferir de forma.

4.13. Reglas de asignación de trabajos

Estas son utilizadas para seleccionar el orden de los trabajos a producir, estas heurísticas proporcionan soluciones razonables en tiempos aceptables. Estas heurísticas toman en cuenta uno o varios indicadores a minimizar entre ellos están: fecha entrega, fecha caducidad, llegada al inventario, tiempos de procesamiento, tiempos de alistamiento, entre otros. El uso de estas reglas puede ayudar en la gestión de inventarios iniciales, en proceso y de productos terminados. Las más conocidas se presentan en la Tabla 11

Tabla 11.

Reglas de asignación de trabajos

Regla de asignación	Descripción
SPT (Shortest Processing Time)	Seleccionar en primer lugar el trabajo con menor tiempo de proceso.
LPT (Large Processing Time)	Programar primero los trabajos con mayor tiempo de proceso.
AT-RPT (Arrival time-remaining processing time)	Selecciona el siguiente trabajo de la cola en función de su hora de llegada al sistema con respecto al tiempo total de procesamiento restante.
EDD (Earliest Due Date)	Los trabajos en espera con fecha de entrega más temprana tienen prioridad para la asignación.
MDD (Modified Due-Date)	Selecciona el trabajo con menor fecha de entrega modificada (MDD), donde MDD equivale al máximo entre la fecha de entrega y la fecha de finalización.
FIFO (First in First Out)	El orden en que llegan los trabajos al centro de procesamiento es el mismo usado para programar su ejecución.
LIFO (Last in First Out)	Se selecciona primero aquel trabajo que ha llegado de último a la cola de procesamiento.
Ramdom	Se programan aleatoriamente los trabajos que se encuentran en la cola.

Continuación Tabla 11

Reglas de asignación de trabajos

Regla de asignación	Descripción
Slack	Se da prioridad a los trabajos con menor tiempo de holgura. Donde la holgura es la diferencia entre la fecha de entrega y el tiempo necesario para terminar el proceso.
Slack Ratio	Los trabajos son programados de acuerdo con la ratio entre el tiempo de holgura y el tiempo de proceso pendiente. Los de mayor ratio tienen prioridad.
CR (Critical Ratio)	Se seleccionan primero los trabajos con menor CR $$CR = \frac{Fecha\ de\ entrega - fecha\ actul}{Tiempo\ de\ producción\ restante}$$
S/RMWK (Slackper Remaining Work)	Esta regla selecciona el trabajo en la cola que presente el mayor tiempo inactividad dividido por el tiempo restante total del trabajo.
SST (Shortest Set-up)	Asigna la prioridad más alta al trabajo con menor tiempo de alistamiento.

Nota: información adaptada de Gómez, P., Andrés, C., & Stania, L. (2006). Estudio Experimental de un Taller Cerámico de Máquinas Paralelas con Secuenciación Dinámica.

5. Descripción del DAPFSP-SDST con fábricas heterogéneas.

El DAPFSP-SDST, es un problema de optimización combinatoria, que busca encontrar la secuencia óptima de producción en *m* máquinas, para *n* trabajos, los cuales son asignados a *p* productos en *f* fábricas. Este es un problema con 2 etapas, donde en la primera etapa se realiza la producción de los *n* trabajos, y en la segunda etapa son ensamblados los *p* productos. Además, este problema cuenta con tiempos de alistamiento dependientes de la secuencia, es decir que el tiempo de preparación depende del trabajo o producto predecesor y sucesor. Por otra parte, contempla que los tiempos de alistamiento y producción son distintos entre fábricas, por esta razón, el problema contempla fábricas heterogéneas.

Para la solución del problema se asume lo siguiente:

- Un trabajo *n* se procesa en solo una fábrica *f* y una máquina *m.*
- Todas las fábricas pueden procesar cualquier trabajo.
- Todas las fábricas tienen al menos dos máquinas y todas tienen el mismo número de máquinas.
- Una máquina *m* solo procesa un solo trabajo a la vez.
- Todos los trabajos se procesan en las máquinas en el mismo orden.
- La secuencia de procesamiento es la misma para todas las máquinas de una misma fábrica.
- Un trabajo *n* solo avanza a la siguiente máquina si ha terminado su procesamiento en la máquina actual.
- El tiempo de alistamiento de las máquinas depende del trabajo que se acaba de procesar y del siguiente.
- Está permitido el alistamiento temprano de las máquinas en las dos etapas.
- Los tiempos de alistamiento y procesamiento son distintos entre máquinas y fábricas, su valor se conoce y es determinístico.

- Se consideran trabajos ficticios cero, que representan el estado inicial de la máquina.
- La máquina de ensamble solo procesa un producto a la vez.
- Para poder ensamblar un producto *p* todos los trabajos *n* que lo componen deben haber terminado el proceso de producción.

5.1. Modelo matemático

El modelo matemático usado en este trabajo se obtiene de la adaptación realizada por Correa González, Ortíz Delgado, & Garavito Hernández en 2017 al modelo que propone Ruiz et al (2013) para el DAPFSP y al primer modelo desarrollado por Naderi y Ruiz (2010) para el DPFSP. A continuación, se presentan los conjuntos, parámetros, variables y restricciones tenidas en cuenta a la hora de diseñar los algoritmos propuestos.

Conjuntos

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

Parámetros

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

- $\boldsymbol{AST_{s,p}}$**:** Tiempo de alistamiento de la máquina de ensamble para el producto p cuando lo precede inmediatamente el producto s.
- $\boldsymbol{ASIG_{j,p}}$**:** Parámetro binario con valor 1 si el trabajo j pertenece al producto p, y valor cero de lo contrario.

Variables

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

Función objetivo:

$$\textbf{Min } Cmax \tag{10}$$

Restricciones

$$\sum_{f=1}^{F} Y_{j,f} = 1 \ \ \forall \boldsymbol{j} \tag{11}$$

$$\sum_{j=1}^{n} X_{0,j,f} = 1 \ \forall \boldsymbol{f} \tag{12}$$

$$\sum_{f=1}^{F} X_{j,j,f} = 0 \ \forall \boldsymbol{j} \tag{13}$$

$$\sum_{f=1}^{F} \sum_{k=0,k\neq j}^{n} X_{k,j,f} = 1 \ \forall \boldsymbol{j} \tag{14}$$

$$\sum_{f=1}^{F} \sum_{j=1,j\neq k}^{n} X_{k,j,f} \leq 1 \ \forall \boldsymbol{k} > \boldsymbol{0} \tag{15}$$

$$\sum_{k=0,k\neq j}^{n} \left(X_{k,j,f} + X_{j,k,f}\right) \leq 2 * \ Y_{j,f} \ \forall \boldsymbol{j}, \boldsymbol{f} \tag{16}$$

$$\sum_{f=1}^{F} \left(X_{k,j,f} + X_{j,k,f}\right) \leq 1 \ \ \forall \boldsymbol{k} = \{\boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{n-1}\}, \forall \boldsymbol{j} > \boldsymbol{k} \tag{17}$$

$$C_{j,i} \geq C_{j,i-1} + \sum_{k=0}^{n} \sum_{f=1}^{F} (X_{k,j,f} * \ PT_{j,i,f}) \ \ \forall \boldsymbol{j}, \boldsymbol{i} \tag{18}$$

$$C_{j,i} \geq C_{k,i} + \sum_{f=1}^{F} [X_{k,j,f} * (PT_{j,i,f} + ST_{k,j,i,f})] + [(\sum_{f=1}^{F} X_{k,j,f}) - 1]M \ \forall \boldsymbol{k}, \boldsymbol{j}, \boldsymbol{i} \tag{19}$$

$$\sum_{s=0,s\neq p}^{P} Z_{s,p} = 1 \ \ \forall \boldsymbol{p} \tag{20}$$

$$\sum_{p=1}^{P} Z_{s,p} \leq 1 \ \ \forall \boldsymbol{s} \tag{21}$$

$$Z_{s,p} + Z_{p,s} \leq 1 \ \forall \boldsymbol{s}, \boldsymbol{p}, \boldsymbol{s} > \boldsymbol{p} \tag{22}$$

$$AC_p \geq (C_{j,m} * ASIG_{j,p}) + \sum_{s=0,s\neq p}^{P} Z_{s,p} * AT_p \ \ \forall \boldsymbol{j}, \boldsymbol{p} \tag{23}$$

$$AC_p \geq AC_s + \left[Z_{s,p} * \left(AT_p + AST_{s,p}\right)\right] + \left(Z_{s,p} - 1\right)M \ \forall \boldsymbol{s}, \boldsymbol{p} \quad (24)$$

$$Cmax \geq AC_p \ \ \forall \boldsymbol{p} \quad (25)$$

$$C_{j,i} \geq 0 \ \forall \boldsymbol{j}, \boldsymbol{i} \quad (26)$$

$$AC_p \geq 0 \ \ \forall \quad (27)$$

$$Cmax \geq 0 \quad (28)$$

$$Y, X, Z \ \in \ \{0,1\} \quad (29)$$

La función objetivo para este problema se encarga únicamente de la minimización del makespan o tiempo de completamiento. La descripción del cálculo del makespan se realiza con detalle en el capítulo 7 en el método de evaluación.

La ecuación 11 contiene una variable binaria que se encarga de que cada trabajo sea procesado en una sola fábrica; la ecuación 12 asegura que solo existe un trabajo por fábrica que tenga al trabajo dummy como predecesor, existiendo de esta manera un trabajo "0" en cada fábrica; la ecuación 13 asegura que un trabajo no pueda ser predecesor o sucesor de sí mismo; la número 14 se encarga de que cada trabajo sea predecesor una sola vez; la ecuación 15 ratifica que cada trabajo solo sea sucesor de otro trabajo máximo una sola vez, a excepción del último trabajo en producirse en cada fábrica; en la ecuación 16 se asegura que cada trabajo tenga su predecesor y sucesor en la misma fábrica. En la ecuación 17 se tiene que un trabajo no puede ser sucesor y predecesor de otro al mismo tiempo; en la ecuación 1*8* se asegura que el tiempo de completamiento de un trabajo sea mayor que su tiempo de completamiento en la máquina anterior, asegurando terminar el procesamiento en una máquina antes de empezar en la otra; la ecuación 19 se tiene que el tiempo de procesamiento de un trabajo en una fábrica debe ser mayor al tiempo de procesamiento de su trabajo predecesor, asegurando de esta manera que una máquina no procese dos trabajos al mismo

tiempo. Las primeras nueve restricciones están relacionadas con la etapa de procesamiento de los trabajos en las fábricas, de la diez a la catorce son restricciones afines a la etapa de etapa de ensamble.

La ecuación 20, se encarga de que cada producto tenga un único predecesor; la ecuación número *21* asegura que cada producto tenga un único sucesor a excepción del último producto en ensamblarse; la ecuación 22 restricción asegura que un producto no sea sucesor y predecesor de otro al mismo tiempo. La ecuación *23* garantiza que todos los trabajos de un producto estén terminados para comenzar el ensamble, de modo que el tiempo de completamiento de un producto sea mayor al tiempo de completamiento del último trabajo de ese producto; el número *24* asegura que el tiempo de completamiento de un producto sea mayor al tiempo de completamiento del producto anterior, de manera que para empezar el ensamble de un producto se debe terminar el del producto anterior. Finalmente, la ecuación *25* asegura que el *makespan* sea el mayor al tiempo de completamiento de todos los productos.

6. Algoritmo Genético (GA)

Para abordar el problema DAPFSP-SDST, se decide utilizar un algoritmo genético debido a las recomendaciones establecidas por los autores más influyentes en investigaciones sobre este tipo de Flowshop. A partir de estas recomendaciones, se diseñan dos algoritmos para dar solución al problema, denominados Algoritmo Genético 1 (*GA1*) y Algoritmo Genético 2 (GA2), en los que se presentan algunas variaciones dentro de las diferentes etapas que conforman a un GA básico.

Por otra parte, los algoritmos *GA1 y el GA2* se codifican en MATLAB R2018B, haciendo uso de funciones modulares. En el apéndice C se muestran los scripts para este algoritmo.

6.1. Representación de la solución para GA1 y GA2

La solución es representada en un vector que contiene la asignación y secuencia de los trabajos en cada fábrica, seguida de la secuencia de ensamble de los productos. En este vector, cada secuencia de fabricación o ensamble está separada por un trabajo dummy que se representa con el número cero "0". Asimismo, al final de la secuencia de ensamble, se muestra el valor del *makespan* correspondiente al vector en cuestión.

La longitud del vector está dada por $1X(f + n + p + 3)$, donde *f* es el número de fábricas de la instancia, que están representadas en el vector con el número cero "0", este número corresponde al trabajo *dummy* presente al inicio de estas fábricas dentro de la etapa de procesamiento. Por otra parte, *n* representa el número de trabajos a ser asignados en la fábrica *f* y *p* el número de productos que deben ser ensamblados en la etapa de ensamble. Las dos posiciones adicionales representan el producto *dummy* de la etapa de ensamble y el valor del *makespan* que se obtiene para la secuencia establecida. A continuación, se encuentra la representación gráfica del vector solución para una instancia de 12 trabajos (n=12), tres fábricas (f=3) y dos productos (p=2). De esta manera, la longitud del vector es igual a $1X(f + n + p + 3) = 20$, Ver Figura 15.

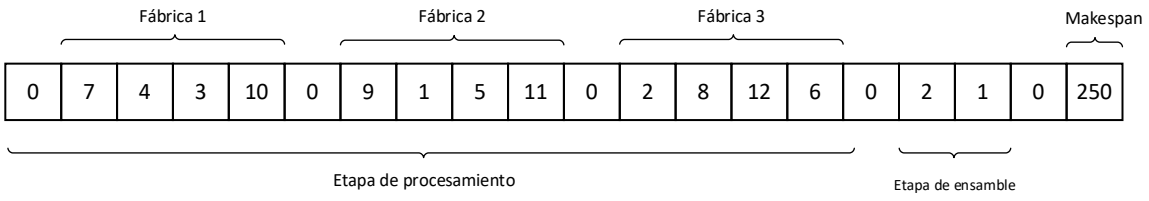


Figura 15. Vector solución.

En este vector, las máquinas no son tenidas en cuenta ya que la secuencia de fabricación de una máquina *m* es idéntica a la secuencia de las *m-1* máquinas restantes que pertenecen a la misma fábrica. Por lo tanto, se puede afirmar que existe una única secuencia de procesamiento por fábrica.

6.2.Descripción del algoritmo *GA1*

A continuación, se presenta la descripción de la forma en que se trabajan cada una de las estructuras propuestas para el *GA1*. Este algoritmo es diseñado con el fin de optimizar la secuencia de procesamiento y de ensamble, dando prioridad en su desarrollo al tratamiento de los datos que corresponden a los trabajos dentro de la etapa de procesamiento. Por otra parte, como se ilustra en el diagrama de flujo mostrado en la Figura 16. Diagrama de flujo GA1, el algoritmo comienza generando aleatoriamente la población inicial de la secuencia de fabricación; posteriormente, se continúa con la ejecución de los operadores seleccionados para este algoritmo. Se empieza realizando selección por torneo con dos competidores, continúa con el cruce y finaliza con el operador de mutación. Finalmente, el algoritmo evalúa el makespan y guarda el mejor individuo de la iteración. GA1 se detiene cuando cumple el criterio de parada, el cual es el número de iteraciones.

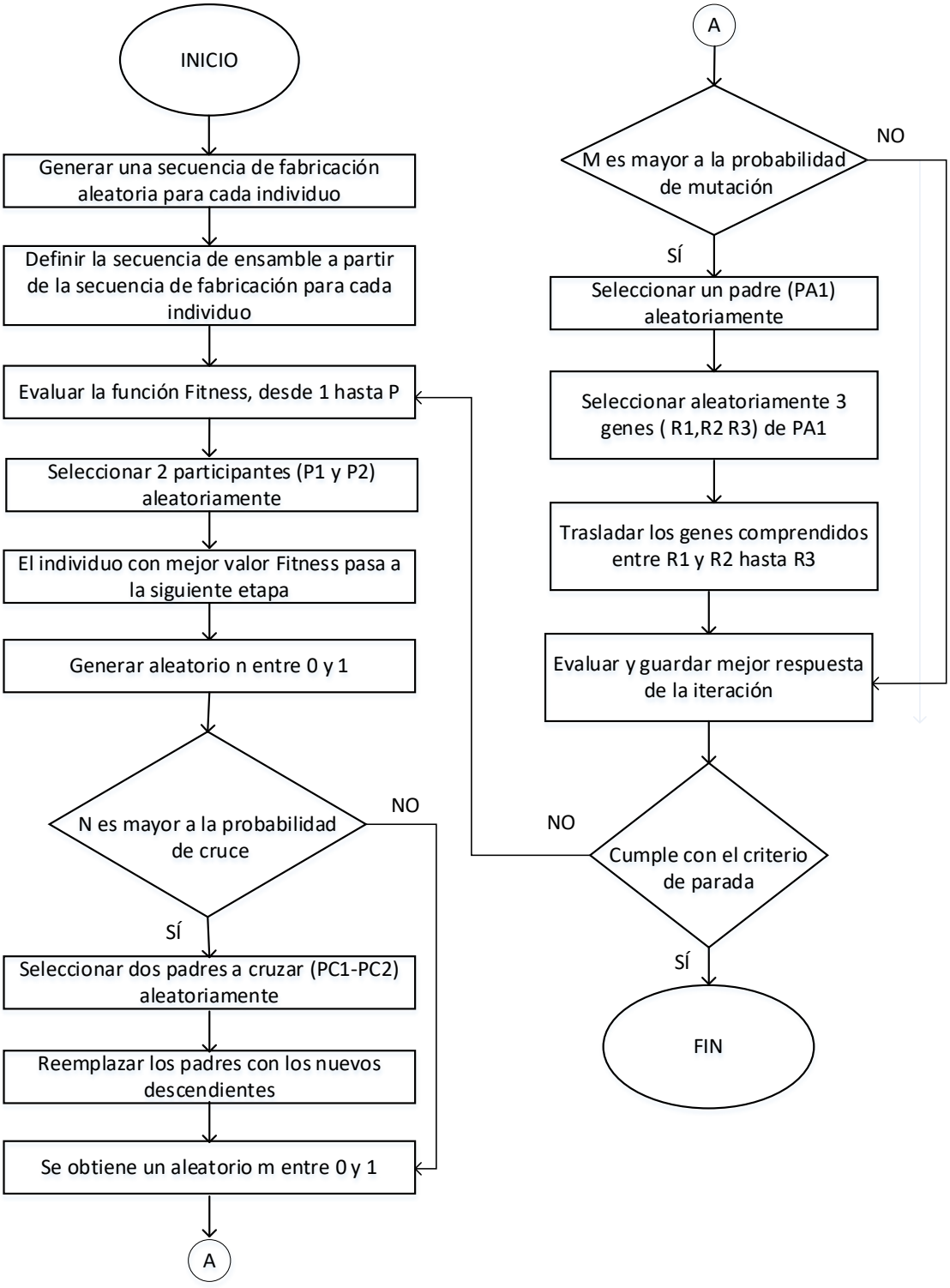


Figura 16. Diagrama de flujo GA1

6.2.1. Generación de la población inicial. La población inicial en este algoritmo es generada de manera aleatoria, por lo tanto, se genera una secuencia de fabricación para cada individuo dentro de la población que es al azar, para posteriormente, definir a partir del método de evaluación la secuencia de ensamble.

6.2.2. Método de evaluación. El método de evaluación del algoritmo presenta la forma en que se estructura la función de adaptación (*fitness*) del problema y el cálculo completo del *makespan*, para cada cromosoma.

6.2.2.1. Función de adaptación (fitness). La función *fitness* está conformada por dos partes La *primera parte* contiene la función objetivo del problema para cada individuo, que en este caso es el *Cmax(i)* (tiempo de completamiento o *makespan*). Por otra parte, la *segunda sección* de la función contempla un castigo. Este castigo se incluye con el fin de disminuir la probabilidad de que una solución con un valor de *makespan* mayor al del promedio general, sea seleccionada como óptima. De esta manera se define el castigo como la diferencia entre el *Cmax(i)* del individuo analizado y el promedio general del *makespan* para la población total, $\overline{\boldsymbol{Cmax}}$.

A continuación, se presenta la expresión matemática (30) para la función de adaptación en caso de que $Cmax(i) > \overline{Cmax}$.

$$F(i) = Cmax(i) + |Cmax(i) - \overline{Cmax}| \tag{30}$$

Por otra parte, si el $Cmax(i) \leq \overline{Cmax}$ la función no presenta castigo y queda de la siguiente manera.

$$F(i) = Cmax(i) \tag{31}$$

6.2.2.2. Cálculo del makespan. Inicialmente el enfoque del método de evaluación se centra en relacionar la secuencia de fabricación previamente generada con la secuencia de ensamble. Por lo tanto, se identifica que el trabajo *j* esté contenido en la fábrica *f* y se guarda la relación de este trabajo con el producto *p* al cual pertenece. Posteriormente se procede a sumar los tiempos de completamiento de los trabajos en las fábricas, teniendo en cuenta que, si el trabajo j es el primero en procesarse, se debe considerar su tiempo de alistamiento respecto al trabajo dummy "0".

Por otra parte, cuando se conoce que los trabajos *j* que pertenecen a un mismo producto *p* ya han terminado la etapa de procesamiento se procede a calcular el tiempo de completamiento del producto teniendo en cuenta las siguientes consideraciones:

- Primera consideración: si el producto a ensamblar es el producto inicial, su tiempo de completamiento es igual al tiempo de completamiento del último trabajo de este producto en la etapa de procesamiento, más su tiempo de ensamble y alistamiento, respecto al producto dummy "0".
- Segunda consideración: si el tiempo de completamiento del último trabajo en la etapa de procesamiento que corresponde al producto en cuestión es mayor o igual a la suma del tiempo de completamiento de los productos antecesores, el tiempo de completamiento del producto actual, es igual al tiempo de completamiento de sus trabajos en las fábricas, más el tiempo de ensamble y alistamiento de este producto.
- Tercera consideración: en caso de que el tiempo total acumulado, sea menor a la suma de los tiempos de completamiento de los productos antecesores, se identifica la existencia de un tiempo muerto, por lo tanto, se procede a mitigar estos tiempos, buscando ubicar los tiempos de alistamiento del producto en cola, dentro de ese periodo muerto.

6.2.3. Selección. Para el problema en cuestión, se propone el uso de un operador de selección denominado *selección por torneo*. Esto se debe a que este operador es uno de los más usados por los autores más relevantes e influyentes en estudios relacionados con problemas de Flowshop.

Como se observa en la Figura 17, se establece que dentro del operador solo se debe trabajar con dos participantes, que son seleccionados aleatoriamente y son comparados a partir de los resultados que se obtienen en la función de evaluación. El ganador del torneo pasa directamente a la siguiente etapa del GA. De igual forma, si un participante gana *n* veces el torneo, este pasa *n* veces a la siguiente etapa del algoritmo.

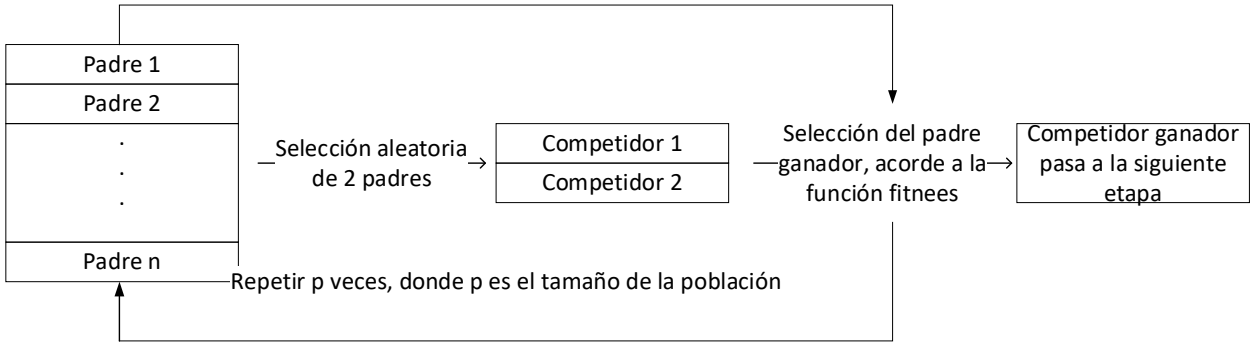


Figura 17. Operador Selección

6.2.4. Cruce. El operador de cruce seleccionado es el *cruce de dos puntos con bloques similares* (Similar Block 2- point Order Crossover, SB2OX), propuesto por Ruiz (2003), quien compara 8 diferentes alternativas de cruce, para una de las variantes del problema de Flowshop y comprueba a partir de un diseño de experimentos que el SB2OX presenta mejor comportamiento que los demás operadores involucrados en el estudio.

Esta etapa inicia con la población de individuos ganadores del torneo en la fase de selección. A continuación, se obtienen dos números aleatorios enteros que van desde 1 hasta el tamaño de la

probación (*t*), y que representan la posición de los padres que deben ser cruzados, con el fin de generar dos nuevos hijos. Posteriormente, se obtiene un tercer número aleatorio, que representa la probabilidad de cruce (*Pcruce*) correspondiente a esos dos individuos. De esta manera, si la probabilidad de cruce de la iteración es mayor o igual a la probabilidad de cruce establecida en el algoritmo, los cromosomas (individuos) son sometidos al proceso de cruce.

Se estructuran 3 etapas para trabajar el SB2OX. En la ***primera etapa***, se establece que los bloques dentro de los cromosomas de los padres, que contienen al menos dos alelos idénticos consecutivos, se copian directamente a los hijos ver Figura 18.

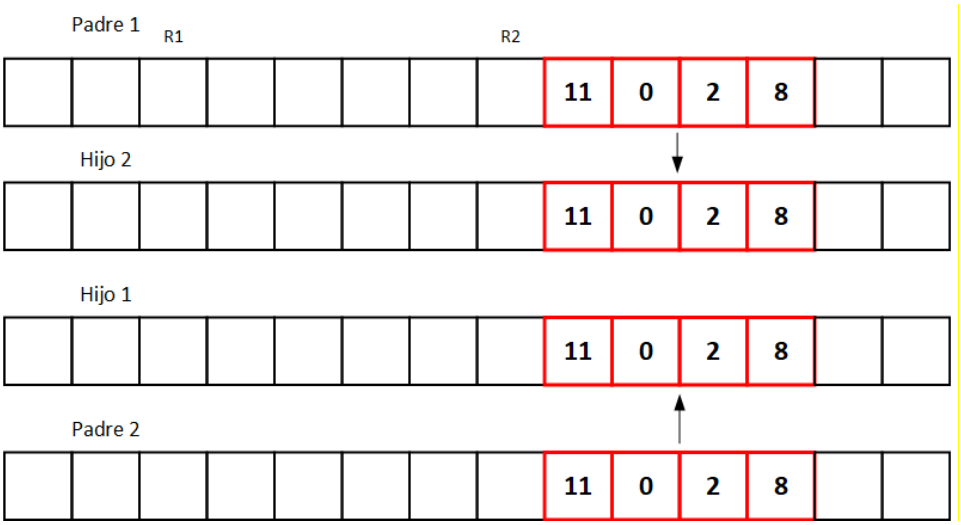


Figura 18. Operador cruce. Primera etapa. Adaptado de Ruiz, R. (2003). Metaheurísticas para la programación flexible de la producción.

En la ***segunda etapa***, se generan dos números aleatorios enteros adicionales que van desde 1 hasta $1x(f+n+p+3)$, para identificar los puntos de cruce en el cromosoma y así obtener las secciones que deben ser heredadas a cada uno de los hijos. De esta manera, los alelos que se encuentran dentro de los puntos de cruce establecidos pasan a uno de los hijos respectivamente, es decir del Padre 1 al Hijo 1 y del Padre 2 al Hijo 2 Figura 19.

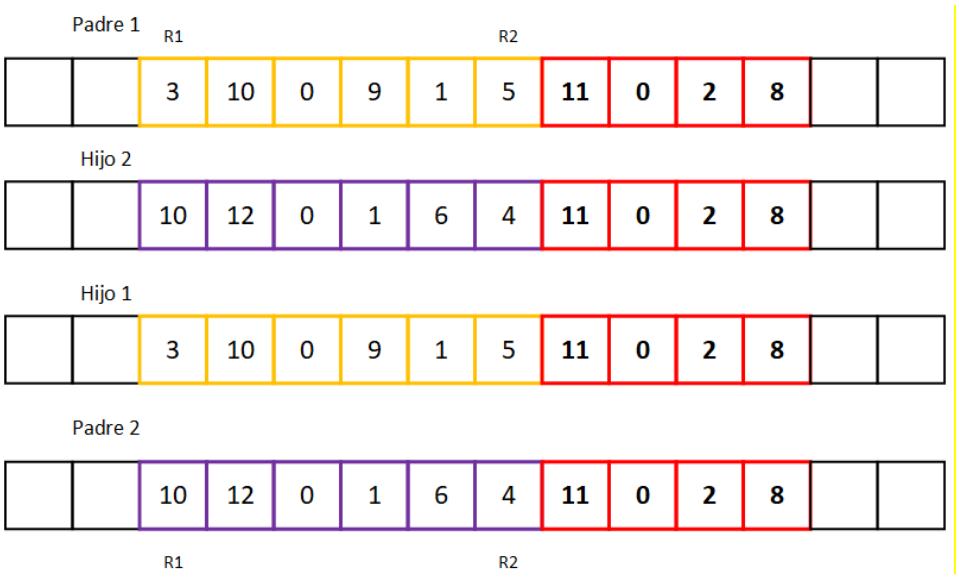


Figura 19. Operador cruce. Segunda etapa. Adaptado de Ruiz, R. (2003). Metaheurísticas para la programación flexible de la producción.

En la ***última etapa***, se heredan las posiciones faltantes del otro padre, es decir, que como se observa en la Figura 20, los genes restantes del padre 1 son heredados al hijo 2 y los genes del padre 2 son heredados al hijo1.

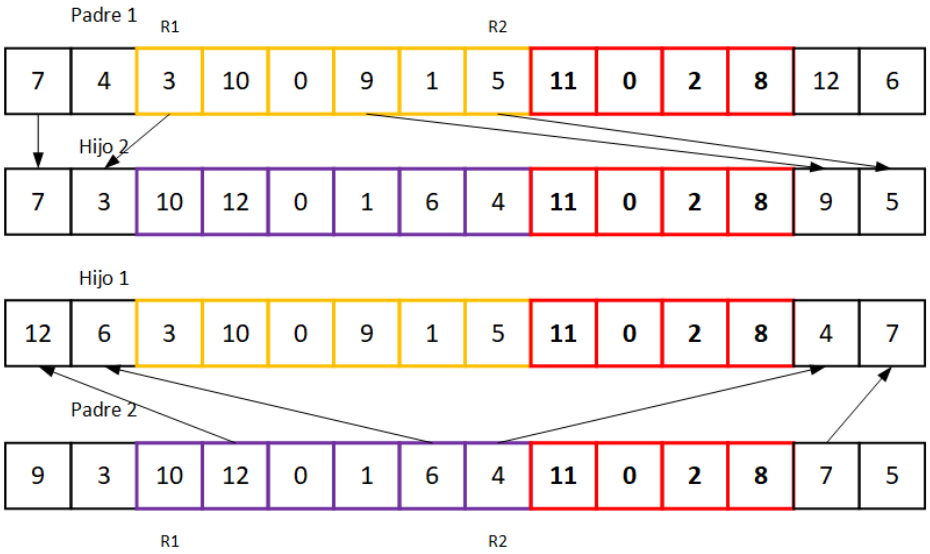


Figura 20. Operador cruce. Etapa final. Adaptado de Ruiz, R. (2003). Metaheurísticas para la programación flexible de la producción.(Ruiz, 2003)(Ruiz, 2003)

Al final de este operador, pasan a la siguiente etapa de mutación únicamente, los hijos que se obtienen después de someter a cruce a los padres.

6.2.5. Mutación. En esta etapa, el operador de mutación a trabajar es el operador por desplazamiento (SHIFT mutación), el cual es considerado dentro de las tres opciones de mutación que se tienen en cuenta en las investigaciones realizadas por Ruiz (2003). Estos estudios se basan en la identificación de las metaheurísticas adecuadas para dar solución a problemas de Flowshop. Asimismo, los resultados demuestran la superioridad y ventaja que tiene SHIFT, sobre los otros tipos de operadores establecidos.

Para empezar la etapa de mutación, se genera un número aleatorio inicial que va de 1 hasta *t*, donde *t* representa el tamaño de la población. Este número permite identificar el individuo a mutar. A continuación, se obtiene otro número aleatorio (de 0 a 1), este número hace referencia a la probabilidad de mutación, es decir a la probabilidad de que el individuo seleccionado sea sometido a esta. De esta manera, si la probabilidad asociada al individuo es mayor o igual a la probabilidad de mutación establecida en el algoritmo, el individuo es mutado.

En la estructura del operador SHIFT, se establecen dos fases. La ***primera fase*** se caracteriza por definir tres números aleatorios entre 0 y 1 (R1, R2 y R3). Donde R1 y R2 representan el inicio y fin del bloque del cromosoma a desplazar. Por otra parte, R3, establece la posición en la que se realiza el desplazamiento, esta posición no debe estar contenida dentro del bloque. En la **segunda fase**, se desplaza el bloque hasta la posición R3, y se acomodan los demás alelos del cromosoma, en su posición relativa. Ver Figura 21

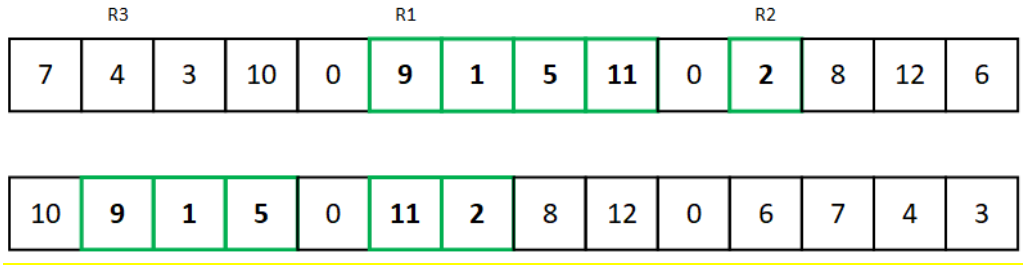


Figura 21. Operador mutación. Adaptado de Ruiz, R. (2003). Metaheurísticas para la programación flexible de la producción.

Es importante tener presente que este proceso de mutación se realiza *p* veces en cada iteración, donde *p* es el tamaño de la población inicial. De manera que para cada individuo se genera un aleatorio el cual es confrontado con la probabilidad de mutación. Finalmente, el individuo que resulta de la mutación pasa a ser parte de la población.

6.3. Descripción del algoritmo GA2

Este algoritmo es una variación del algoritmo *GA1*. Donde la estructura que se diseña y desarrolla, se encarga de optimizar las secuencias de las dos etapas del problema, pero centra su atención en el tratamiento de los productos durante la etapa de ensamble. Como se observa en el diagrama de flujo mostrado en la Figura 22 el algoritmo comienza generando aleatoriamente la población inicial de la secuencia de ensamble, posteriormente se continúa con la ejecución de los operadores seleccionados para este algoritmo. De la misma manera que en el GA1, se empieza realizando selección por torneo, continúa con el cruce y finaliza con el operador mutación, finalmente el algoritmo evalúa el makespan y guarda el mejor individuo de la iteración. GA2 se detiene cuando cumple el criterio de parada, el cual es el número de iteraciones.

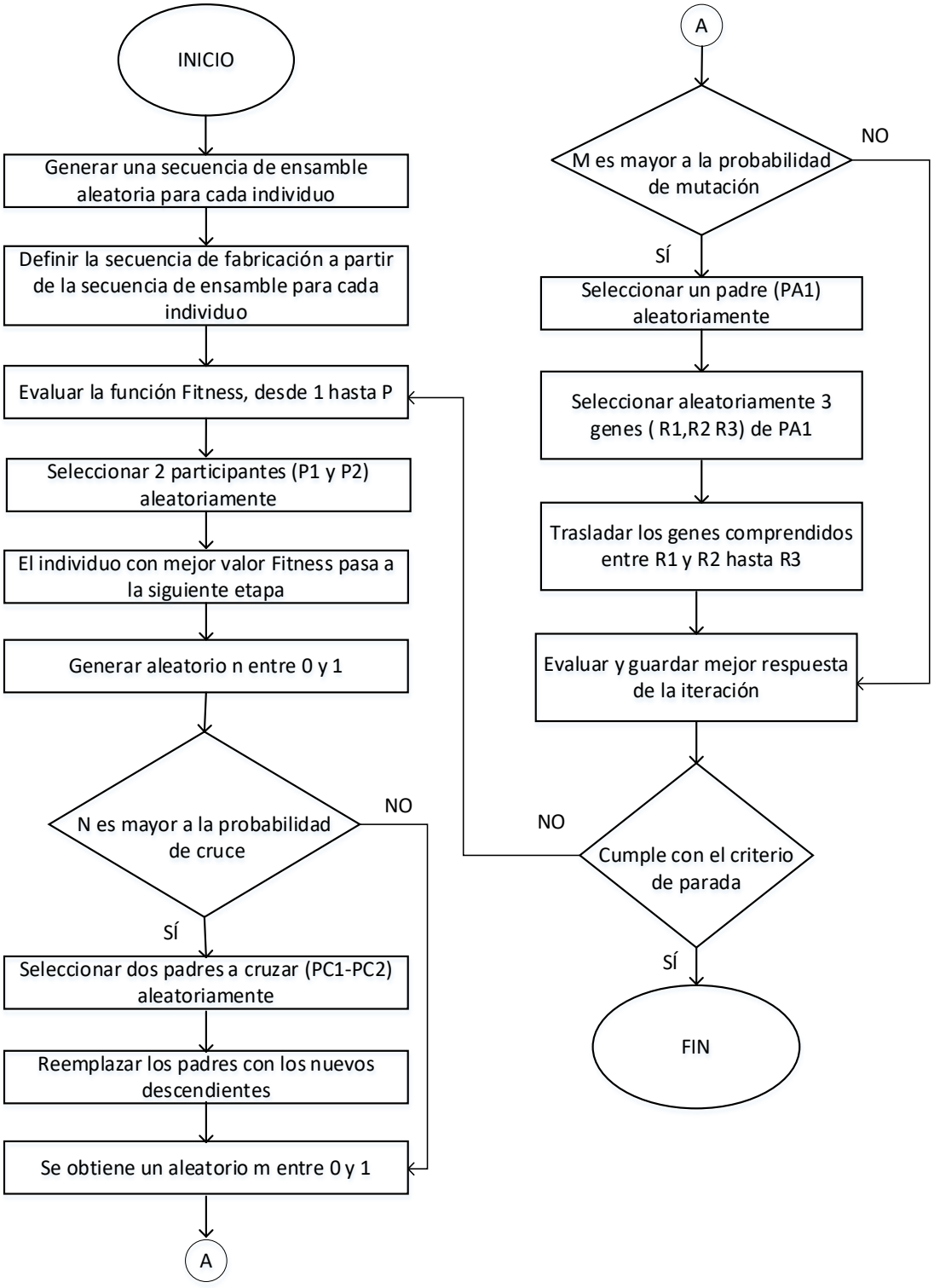


Figura 22 Diagrama de flujo GA2

A lo largo del algoritmo se realizan modificaciones en algunas de las estructuras establecidas para el algoritmo *GA1*, esto se debe a que el nuevo enfoque de asignación se centra en la etapa de ensamble. De esta manera, la generación de la población inicial y el método de evaluación sufren modificaciones significativas. Por otra parte, la codificación de las estructuras restantes como selección, mutación y cruce conservan la misma distribución definida en el *GA1*.

En este caso en la **población inicial,** se genera aleatoriamente una secuencia de ensamble para cada individuo. Posteriormente, se establece con ayuda del método de evaluación, una secuencia de fabricación aceptable asociada a la secuencia de ensamble previamente establecida.

En el **método de evaluación**, la función de adaptación (*fitness*) conserva la configuración mostrada en la ecuación (30). Por otra parte, se generan modificaciones a la estructura, debido a que se parte de la secuencia de ensamble aleatoria generada al inicio. Por lo tanto, se establece una metodología alterna para relacionar la secuencia de ensamble con la secuencia de fabricación.

Esta metodología empieza por identificar los trabajos j que pertenecen a cada uno de los productos p. Posteriormente, se encarga de asignar a las fábricas los trabajos j que forman parte del producto p que debe ser ensamblado primero, con el fin de garantizar que estos trabajos estén listos en el menor tiempo posible y se disminuyan los tiempos muertos en la etapa de ensamble. De esta manera se realiza la asignación de los demás trabajos a las fábricas considerando el orden de ensamble en la secuencia generada inicialmente.

Finalmente se realiza el cálculo de los tiempos de fabricación y de ensamble de la misma forma que se hace en el algoritmo *GA1*, para obtener de esta manera el tiempo de completamiento o *makespan*.

.

7. Algoritmo híbrido entre Algoritmo Genético y VND (*HGA1-VND*)

En este capítulo se propone un segundo método de solución para el DAPFSP-SDST denominado HGA1-VND, en donde uno de los operadores del algoritmo genético es reemplazado por una de las estructuras de la metaheurística VND, se decide hacer este híbrido teniendo en cuenta las recomendaciones encontradas en la revisión de literatura, en donde la búsqueda variable en el vecindario descendente ha arrojado buenos resultados para el problema planteado.

La representación de la solución para este algoritmo se genera de igual manera que en el algoritmo genético (*GA1 y GA2*), descrito con detalle en el capítulo 7 en el numeral 7.1. Asimismo, este algoritmo es codificado en MATLAB R2018B, tomando como base algunas funciones usadas en el algoritmo *GA1*. En el apéndice D se presenta la compilación de las diferentes funciones elaboradas para este algoritmo.

7.1.Descripción del algoritmo

El *HGA1-VND* tiene como estructura base gran parte de las funciones codificadas para el algoritmo genético propuesto al en el capítulo anterior. De manera que, para este algoritmo, la generación de la población inicial, los operadores de selección y cruce y el método de evaluación presentan la misma codificación del algoritmo *GA1*. Sin embargo, el operador de mutación es reemplazado por la estructura de inserción que hace parte de las estructuras de algoritmo VND.

7.1.1. Inserción. Esta estructura se implementa después de ejecutar el operador de cruce. Seguido a esto se genera un número aleatorio, el cual es confrontado con el factor seleccionado de probabilidad de inserción, si cumple con el criterio se realiza inserción. Esta estructura empieza con la generación de un segundo número aleatorio, el cual escoge el individuo de la población al

que se le aplica inserción, este aleatorio va comprendido desde 1 hasta t, donde t es el tamaño de la población. De este individuo se selecciona la fábrica con mayor tiempo de procesamiento; finalmente se elige aleatoriamente un trabajo de esta fábrica y se inserta en una posición aleatoria de la fábrica con menor tiempo de completamiento. De la misma manera que se realiza en el operador mutación para el algoritmo genético, la inserción se efectúa p veces en cada iteración, donde p es el tamaño de la inicial**.**

A continuación, en la Figura 23, se presenta el diagrama de flujo con las estructuras correspondientes al algoritmo híbrido entre genético y VND. En este caso, la sección del diagrama que se encuentra sombreada corresponde a la estructura del VND que reemplaza al operador de mutación dentro del diseño del HGA1-VND.

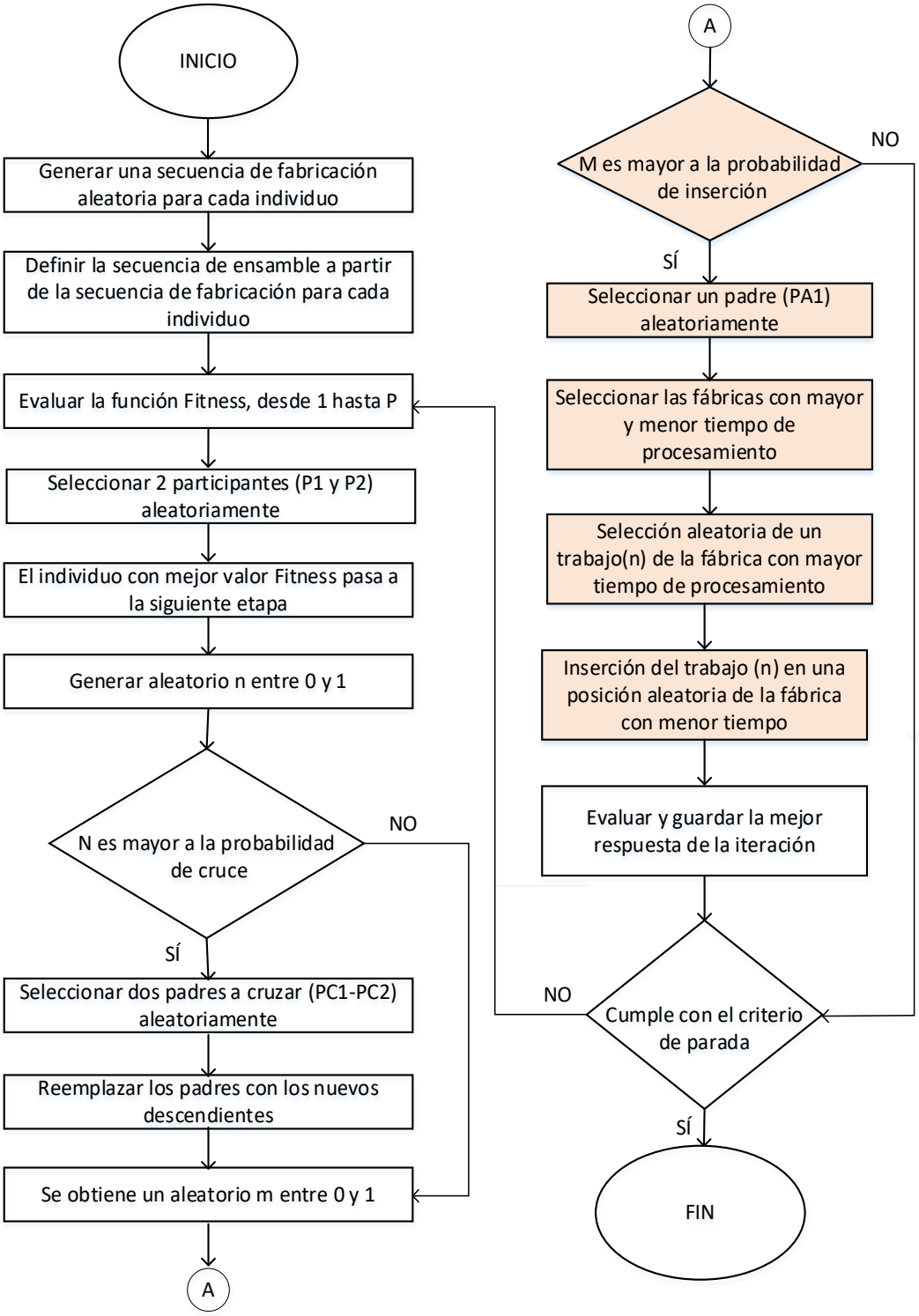


Figura 23. Diagrama de flujo *HGA1-VND*

8. Algoritmo híbrido entre Algoritmo Genético y Algoritmo Voraz (*HGA1-GR*).

Este algoritmo es el tercer método de solución propuesto en la presente investigación para dar solución del problema DAPFSP-SDST denominada HGA1-GR. Esta metaheurística, se caracteriza por generar la población inicial a través de un algoritmo voraz; se opta trabajar con este algoritmo debido a que está metaheurística hace parte de las más usadas para abordar este problema en los últimos años.

La representación de la solución es la misma de los algoritmos presentados anteriormente, la cual es detallada en el capítulo 7, numeral 7.1. De la misma manera que el algoritmo *GA1* y *HGA1-VND*, este algoritmo también es codificado en MATLAB R2018b. Los scripts de este híbrido se muestran en el apéndice E.

8.1.Descripción del algoritmo

Este algoritmo implementa los operadores (selección, cruce y mutación) y el método de evaluación del *GA1*. Presentando como única diferencia la manera en que se genera la población inicial, debido a que en este caso esta es generada atreves de un Algoritmo Voraz y no de manera aleatoria.

8.1.1. Población inicial. La generación de la población inicial para este algoritmo comienza con la asignación aleatoria de un trabajo a cada fábrica, a continuación, se selecciona los trabajos sucesores a los primeros trabajos que fueron asignados, escogiendo aquel trabajo que presente menor tiempo de fabricación (tiempo total que gasta el trabajo en pasar por todas las máquinas de la fábrica) y tiempo de alistamiento. Este proceso se realiza p veces, donde p es el tamaño de la población inicial.

A continuación, en la Figura 24 se presenta el diagrama de flujo con las estructuras correspondientes al algoritmo híbrido entre Genético y Voraz. En este diagrama, la parte sombreada de color verde representa el uso de la estructura del algoritmo voraz, para generar la población inicial.

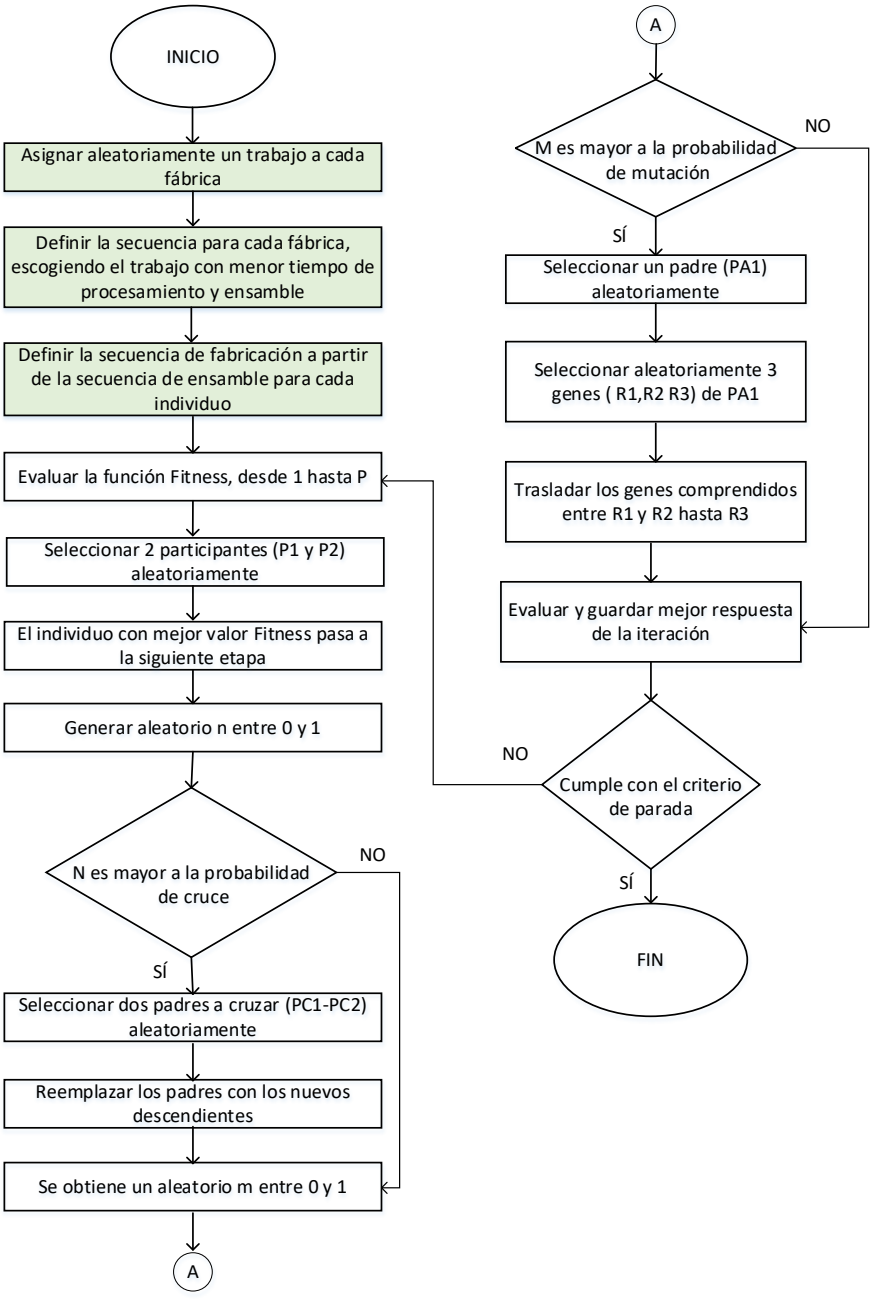


Figura 24. Diagrama de flujo *HGA1-GR*

9. Calibración de parámetros de los algoritmos

Con el propósito de generar los mejores resultados, se realiza un proceso de calibración de los factores que por recomendaciones en la literatura son considerados en los diseños de experimentos, debido a que presentan variabilidad en los valores a medida que se cambia de problema de optimización.

Para definir los parámetros que mejor se adaptan a los algoritmos diseñados y al problema en cuestión, se realiza un diseño de factores 3^4, en el cual se utilizan dos replicas, generando así 162 corridas por subgrupo de instancia. De esta manera, se evalúa el efecto del tamaño de la población, la probabilidad de cruce, la probabilidad de mutación y el número de iteraciones sobre el rendimiento de cada uno de los algoritmos desarrollados. A continuación, en la Tabla 12 se presentan los factores junto con los 3 niveles utilizados en el diseño.

Tabla 12.

Factores y niveles del diseño de experimentos

	Nivel		
Factor	**Alto**	**Medio**	**Bajo**
Tamaño Población	100	200	300
Probabilidad de Cruce	0,4	0,6	0,8
Probabilidad de mutación	0,05	0,1	0,4
Número de iteraciones	100	150	200

Este diseño es realizado por grupos de instancias, entendiendo por instancia un estado del problema que cuenta con atributos específicos. Es decir, que se lleva a cabo una clasificación de las instancias por número de trabajos, con el fin de identificar con mayor exactitud la combinación de parámetros más adecuada para cada algoritmo. Existen 900 instancias en la literatura, que se encuentran divididas en tres grupos, instancias grandes, medianas y pequeñas. Las instancias pequeñas, cuentan con subgrupos de 8 y 12 trabajos y las medianas están conformadas por instancias de 16, 20 y 24 trabajos. Por otra parte, las instancias grandes poseen 100 y 200 trabajos. En la Figura 25 se presentan los grupos y subgrupos de instancias, con su respectiva clasificación.

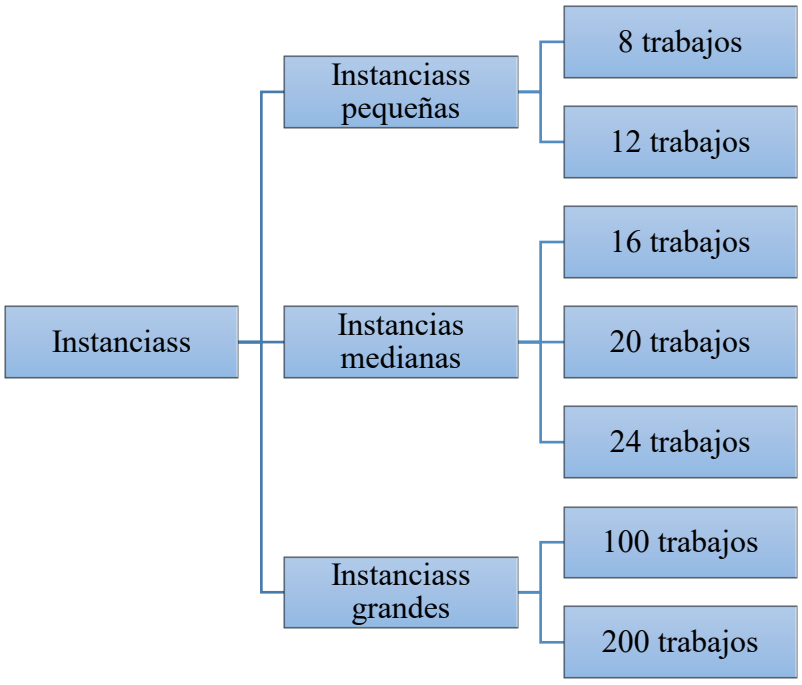


Figura 25. Grupos y subgrupos de instancias

En cada subgrupo de instancias pequeñas y medianas, se seleccionan 3 instancias diferentes para realizar el análisis del diseño. Por otra parte, en las instancias grandes, se seleccionan 4 instancias, dos de estas son de 100 trabajos y las restantes son de 200 trabajos.

A partir del diseño se lleva a cabo un análisis de varianza ANOVA, con el fin de identificar el efecto que existe de los factores sobre la variable *makespan* y la combinación de parámetros con la que se obtienen mejores resultados.

A continuación, se presenta la calibración de los parámetros para tres de los cuatro algoritmos propuestos inicialmente (GA1, HGA1-VND Y HGA1-GR). Esto se debe a que, aunque se definen dos estructuras de Algoritmo Genético (GA1 y GA2), el GA1 presenta mejores resultados para la variable makespan, en un tiempo computacional menor. En el apéndice F, G y H se presenta la información de las instancias utilizadas para realizar el diseño de experimentos de la calibración de los algoritmos GA1, HAG1-VND y HAG1-GR, respectivamente.

9.1.Instancias pequeñas

Dentro de esta sección del diseño, se cuenta con el análisis de los dos subgrupos de las instancias pequeña para cada uno de los tres algoritmos. Se estudian las instancias que contienen 8 y 12, trabajos respecto a la generación de la respuesta de la variable *makespan*. Estas instancias cuentan con 2,3 y 4 productos, 2,3 y 4 fábricas en la etapa de procesamientos y 2,3,4 y 5 máquinas en cada fábrica.

9.1.1. Instancias de 8 trabajos. Para las instancias de 8 trabajos inicialmente se propone hacer el análisis de la varianza respecto a las respuestas generadas para la variable *makespan.* Se obtiene el promedio de tres instancias seleccionadas aleatoriamente, pero al ser instancias tan pequeñas, las respuestas para la variable no presentan cambios significativos respecto a la variación de parámetros. Por lo tanto, se decide realizar el análisis a partir del tiempo computacional empleado en cada uno de los tratamientos.

En el *Algoritmo genético GA1* se corren tres instancias, la instancias *N8P4F4M3, N8P4F2M2 y N8P3F4M4.* En este caso con ayuda de la Figura 26 se identifica que, para el tamaño de la población, el valor que genera un tiempo computacional menor es el de 100. Por otra parte, para la probabilidad de cruce y de mutación encontramos que los tiempos computacionales menores son obtenidos cuando sus valores son 0,8 y 0,4. Finalmente se identifica que cuando se trabaja con un número de iteraciones de 100, los resultados se obtienen en menor tiempo.

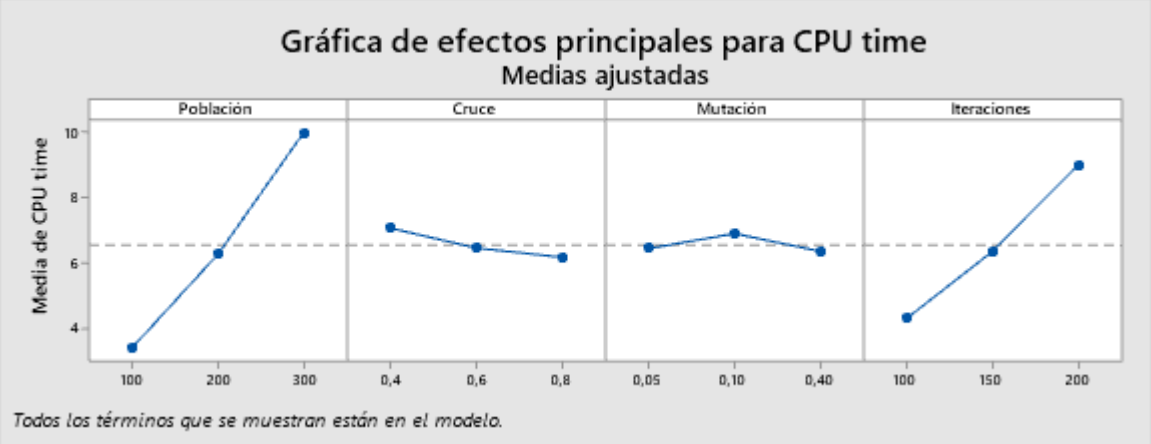


Figura 26. Efectos principales para CPU time en instancias de 8 trabajos del *GA1*

Para el *Algoritmo híbrido entre Genético y VND* se seleccionan las instancias *N8P4F2M5, N8P3F2M5* y *N8P4F2M3*. A partir de la Figura 27 se identifica que el tamaño de población más adecuado es 100. Adicional, se identifica que una probabilidad de cruce de 0,8, una probabilidad de mutación de 0,4 y un número de iteraciones de 100, generan un menor tiempo computacional.

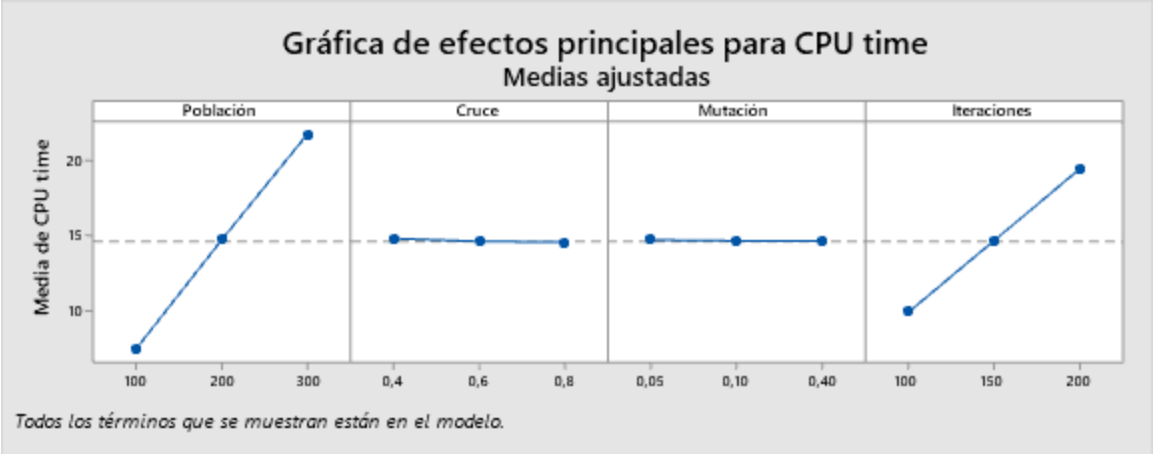


Figura 27. Efectos principales para CPU time en instancias de 8 trabajos del *HGA1-VND*

Finalmente, en el caso del *Algoritmo híbrido entre Genético y Voraz* se analizan las instancias *N8P3F2M2, N8P4F2M3 y N8P4F4M2* y se selecciona finalmente un tamaño de población de 100, una probabilidad de cruce de 0,8, una probabilidad de mutación 0,1 y un número de iteraciones de 100. Teniendo en cuenta que estos valores en cada uno de los parámetros generan menor tiempo computacional. En la Figura 28 se observan los efectos principales de estas instancias en la variable *makespan* para este algoritmo.

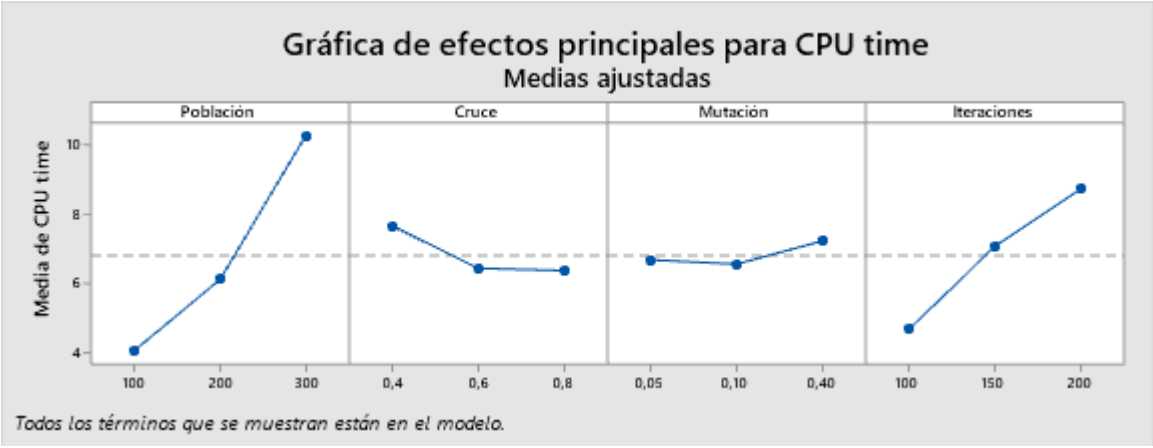


Figura 28. Efectos principales para CPU time en instancias de 8 trabajos del *HGA1-GR*

A continuación, se presenta la Tabla 13, con el resumen con los valores de los parámetros seleccionados para cada algoritmo.

Tabla 13.

Niveles de factores seleccionados para instancias de 8 trabajos

	Factores			
Algoritmo	**Población**	**P. Cruce**	**P. Mutación**	**Iteraciones**
GA1	100	0,8	0,05	100
HGA1-VND	100	0,8	0,4	100
HGA1-GR	100	0,8	0,1	100

9.1.2. Instancias de 12 trabajos. Para las instancias de 12 trabajos, se seleccionan tres instancias aleatoriamente en cada uno de los tres algoritmos, y se analiza el promedio de estas a partir del diseño de factores $\mathbf{3^4}$, finalmente se seleccionan los niveles de cada factor que proporcionen el menor *makespan*. Esto se realiza con ayuda de la gráfica de *Efectos principales para makespan.*

En el caso del *GA1*, se analizan las instancias *N12P4F4M4, N12P3F4M5 y N12P4F3M5*. Como resultado se obtiene que la combinación de factores que proporciona un menor valor en la variable *makespan*, es un tamaño de población de 200, una probabilidad de cruce de 0,8, una probabilidad de mutación de 0,05 con un número de iteraciones de 200.

Por otra parte, en la Figura 29 se puede observar que el factor tamaño de población presenta un empate entre los niveles 200 y 300, por lo tanto, en este caso se realiza un análisis del efecto de los niveles de este factor en la variable tiempo computacional (CPU time) y se decide seleccionar

el nivel que menor tiempo computacional proporcione. De esta manera, se escoge el tamaño de población equivalente a 200 individuos.

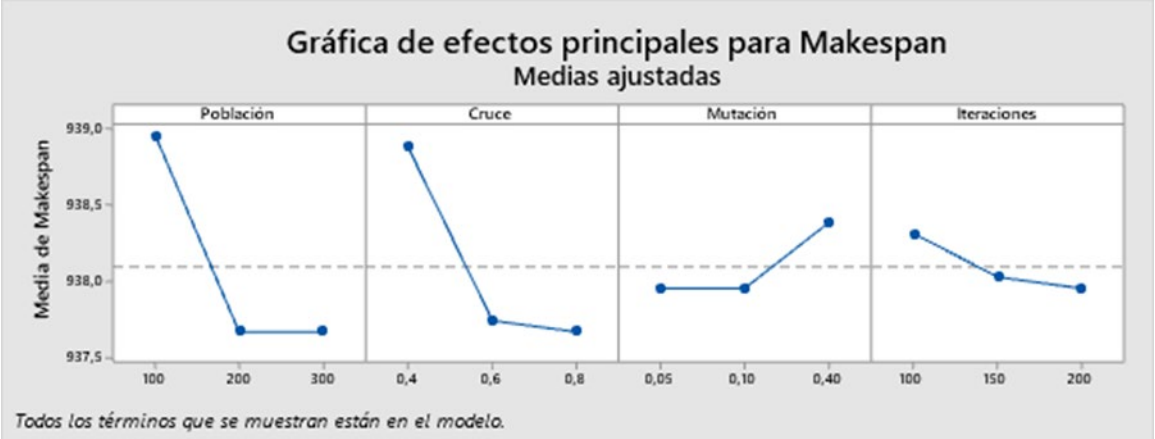


Figura 29. Efectos principales para *makespan* en instancias de 12 trabajos del *GA1*

En el caso del *HGA1-VND*, se seleccionan las instancias *N12P3F2M2, N12P3F3M4 y N12P4F3M5* para realizar el análisis y se obtiene que, con un tamaño de población de 200, una probabilidad de cruce de 0,8, una probabilidad de mutación de 0,4 y un número de iteraciones igual a 150, se genera el menor tiempo de completamiento en instancias de 12 trabajos para este algoritmo.

Como se observa en la Figura 30, existe un empate entre el número de iteraciones 150 y 200, por lo tanto, se analiza de nuevo la influencia de cada nivel en el CPU time y se selecciona el nivel que genere un menor tiempo computacional, que para este caso es un número de iteraciones de 150.

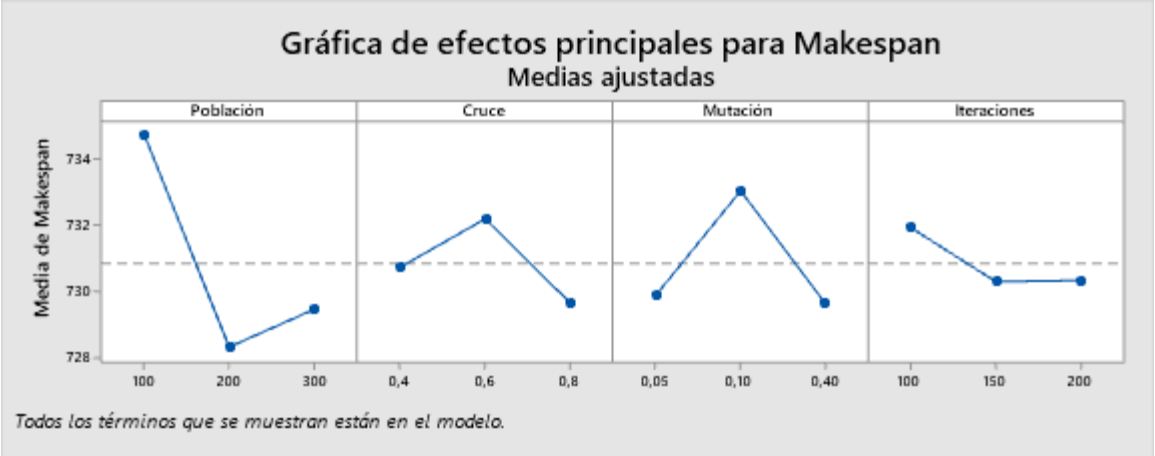


Figura 30. Efectos principales para *makespan* en instancias de 12 trabajos del *HGA1-VND*

A partir de la Figura 31 y teniendo en cuenta que para el *HGA1-GR*, se analizan las instancias *N12P4F4M3, N12P4F3M5 y N12P3F3M5*, se identifica que la combinación adecuada de niveles para cada factor debe ser un tamaño de población de 200, una probabilidad de cruce de 0,6, una probabilidad de mutación de 0,1 y un número de iteraciones de 150. En este caso, nuevamente para los factores que presentan empates entre sus niveles respecto a la variable *makespan*, se realiza el análisis de la influencia de estos sobre el CPU time.

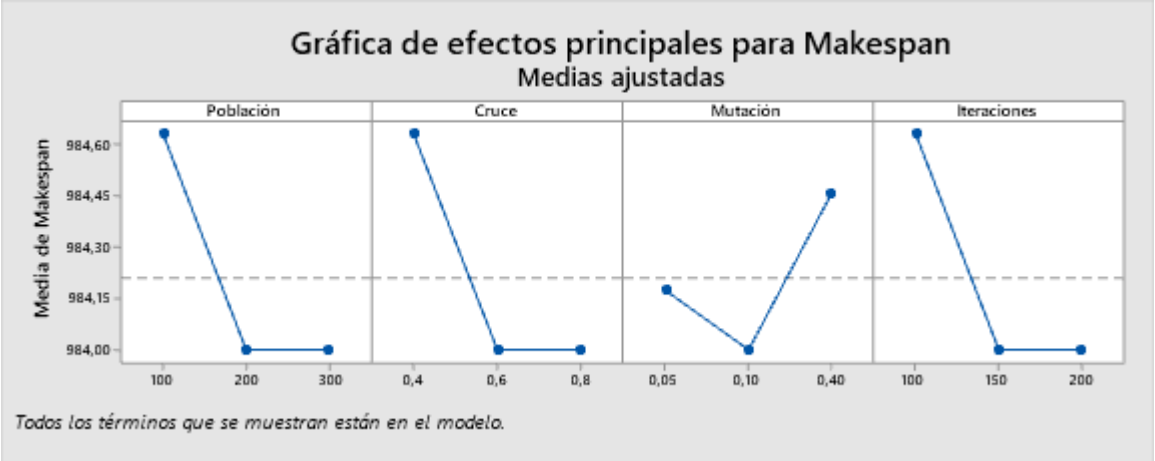


Figura 31. Efectos principales para *makespan* en instancias de 12 trabajos del *HGA1-GR*

A continuación, en la Tabla 14 se presenta los niveles de cada factor seleccionado para cada algoritmo después del análisis de las gráficas.

Tabla 14.

Niveles de factores seleccionados para instancias de 12 trabajos

	Factores			
Algoritmo	**Población**	**P. Cruce**	**P. Mutación**	**Iteraciones**
GA1	200	0,8	0,05	200
HGA1-VND	200	0,8	0,4	150
HGA1-GR	200	0,6	0,1	150

9.2.Instancias medianas

Dentro de las instancias medianas se encuentran aquellas que esta conformadas por un número de trabajos equivalente a 16, 20 y 24. Por otra parte estas instancias pueden contar con un valor de *p* productos entre 2,3 y 4, un valor de *f* fábricas entre 2,3 y 4 y un número de *m* máquinas entre 2,3,4 y 5.

9.2.1. Instancias de 16 trabajos. Las instancias compuestas por 16 trabajos, representa el primer grupo de instancias medianas. Para esta parte de la investigación se seleccionan 3 instancias aleatoriamente en cada uno de los tres algoritmos y se identifica la combinación de parámetros que optimiza de mejor manera el objetivo de investigación del problema.

En la Figura 32 se identifica que al analizar las instancias *N16P3F2M5, N16P4F4M2 y N16P4F2M2* para el *GA1*, los niveles de cada factor que menor tiempo de completamiento generan

son un tamaño de población de 300, una probabilidad de cruce de 0,4, una probabilidad de mutación de 0,1 y un número de iteraciones de 200.

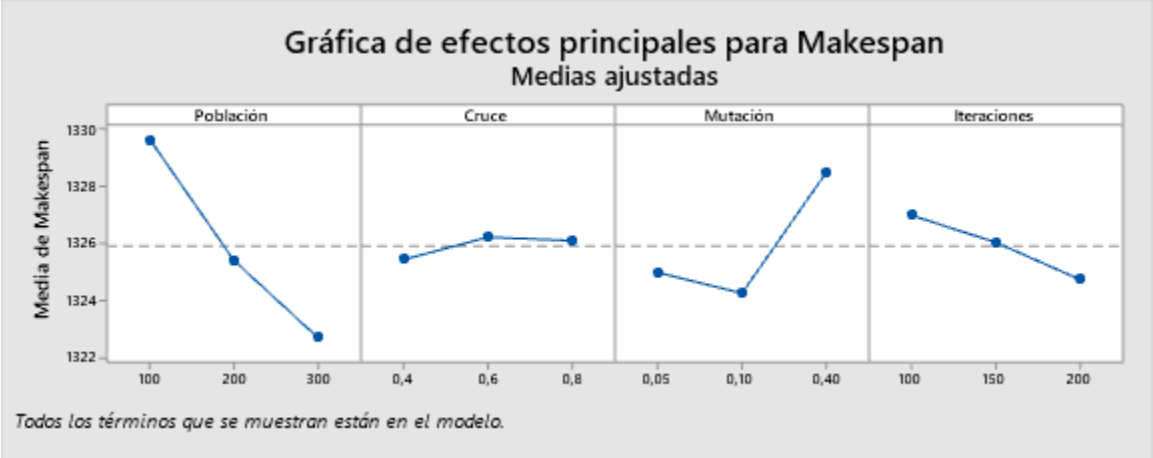


Figura 32. Efectos principales para *makespan* en instancias de 16 trabajos del *GA1*

En el caso del *HGA1-VND*, se seleccionan las instancias *N16P3F2M2, N16P4F4M2 y N16P3F4M4* para realizar el análisis del diseño factorial y a partir de la interpretación de la información presente en la Figura 33, se define que el tamaño de la población más adecuado es de 300 individuos. Por otra parte, la probabilidad de cruce y mutación y el número de iteraciones deben ser de 0,8, 0,4 y 150 respectivamente.

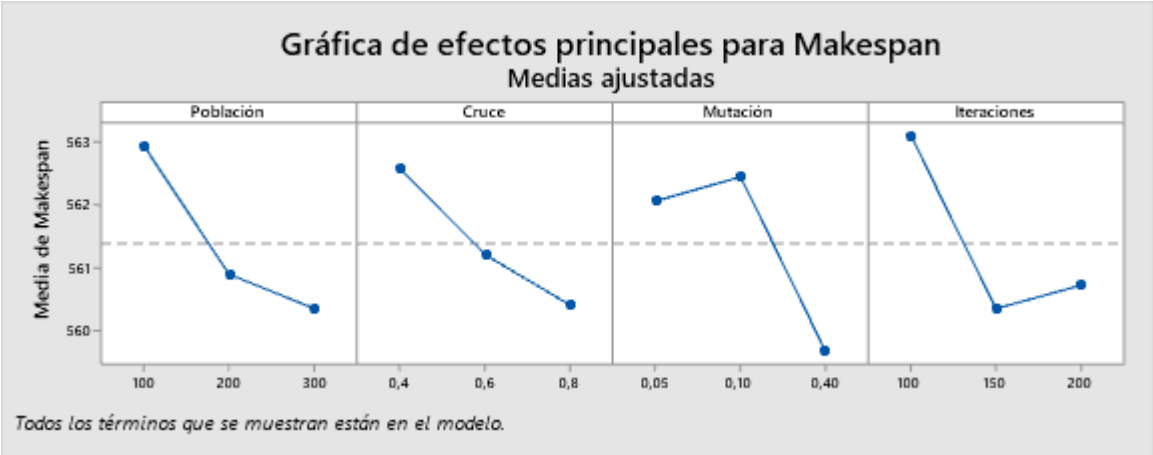


Figura 33. Efectos principales para *makespan* en instancias de 16 trabajos del *HGA1-VND*

Se define a partir del análisis de la Figura 34, que la combinación de niveles de factor que mejor se adapta al *HGA1-GR* cuenta con un tamaño de población de 300, una probabilidad de cruce de 0,4, una probabilidad de mutación de 0,1 y 150 iteraciones. Para efectos del análisis del diseño, se tuvieron en cuenta las instancias *N16P4F4M3, N16P3F2M5 y N16P3F2M2.*

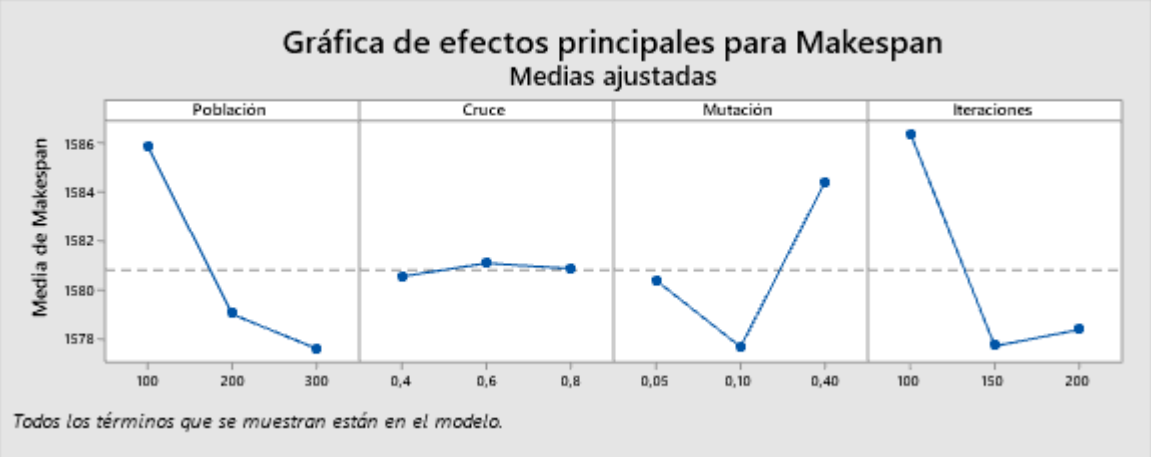


Figura 34. Efectos principales para *makespan* en instancias de 16 trabajos del *HGA1-GR*

A continuación, en la Tabla 15 se presenta los niveles seleccionados finalmente para cada factor en cada uno de los tres algoritmos desarrollados.

Tabla 15.

Niveles de factores seleccionados para instancias de 16 trabajos

	Factores			
Algoritmo	**Población**	**P. Cruce**	**P. Mutación**	**Iteraciones**
GA1	300	0,4	0,1	200
HGA1-VND	300	0,8	0,4	150
HGA1-GR	300	0,4	0,1	150

9.2.2. Instancias de 20 trabajos. Las instancias de 20 trabajos son analizadas teniendo en cuenta tres instancias seleccionadas aleatoriamente para cada uno de los algoritmos estructurados. De esta manera, se obtiene la combinación de niveles de factor más adecuada a partir de la interpretación de los efectos principales en la variable respuesta *makespan*.

Para la calibración de los factores del *GA1*, se seleccionan las instancias *N20P4F4M3, N20P3F4M4 y N20P4F2M5*. Asimismo, a partir de la Figura 35, se identifican que el mejor nivel para el tamaño de población tiene un valor de 100 individuos y que la probabilidad de cruce debe ser de 0,6. Por otra parte, la probabilidad de mutación y numero de iteraciones es de 0,05 y 100, respectivamente.

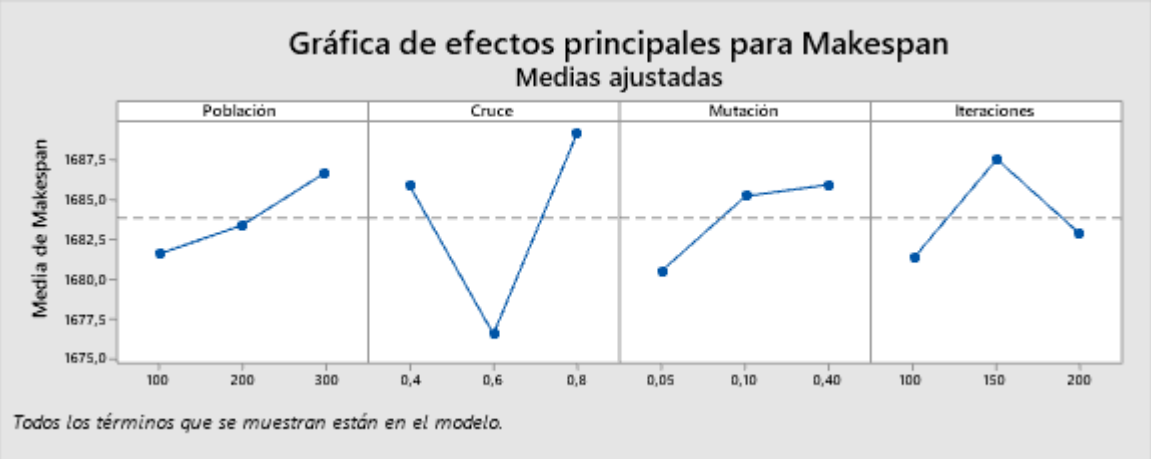


Figura 35. Efectos principales para *makespan* en instancias de 20 trabajos del *GA1*

En el *HGA1-VND*, se escogen las instancias *N20P4FM2, N20P4F4M3 y N20P3F2N4* para realizar el respectivo análisis factorial. De este análisis, se obtiene la Gráfica de efectos principales para el *makespan* (Ver Figura 36), donde se identifica que una población de 200 individuos, una probabilidad de cruce de 0,8 una probabilidad de mutación de 0,4 y número de iteraciones de 200, generan un menor tiempo de completamiento para instancias de 20 trabajos.

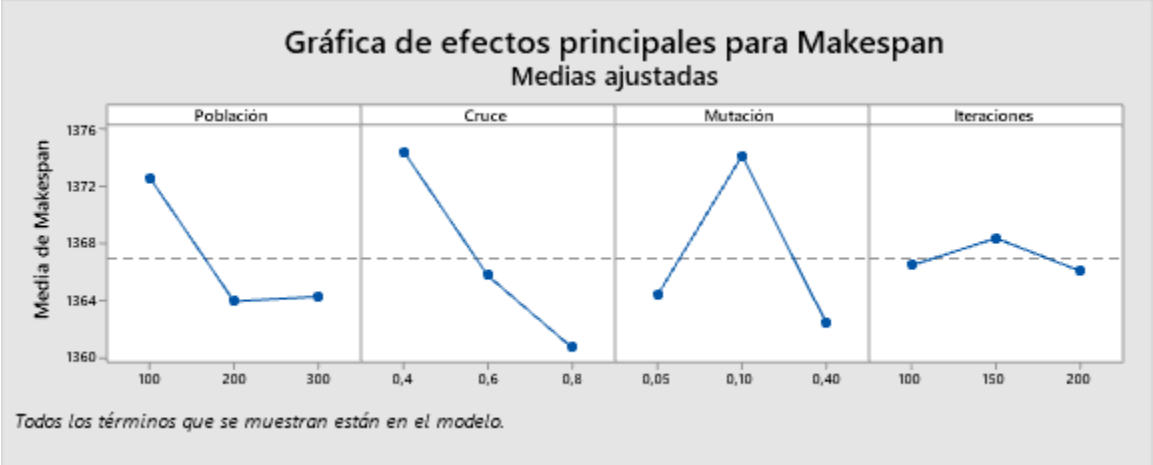


Figura 36. Efectos principales para *makespan* en instancias de 20 trabajos del *HGA1-VND*

Por otra parte, se analizan las instancias *N20P4F4M4, N20P3F4M5 y N20P4F2M4* con el objetivo de identificar la combinación de niveles de factor más adecuada para el algoritmo *HGA1-GR*. En la Figura 37, se observa que los niveles que generan el menor valor de *makespan* en cada factor son 300 para el tamaño de la población, 0,6 para la probabilidad de cruce, 0,4 para la probabilidad de mutación y 150 para el número de iteraciones.

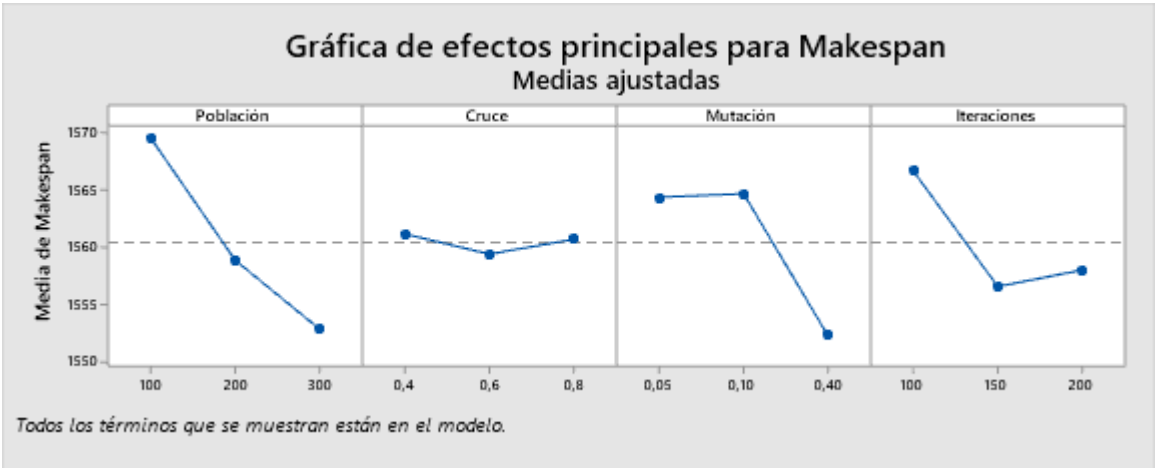


Figura 37. Efectos principales para makespan en instancias de 20 trabajos del HGA1-GR

A continuación, se presenta la Tabla 16, en donde se especifica la combinación de niveles de factor seleccionada para cada uno de los tres algoritmos.

Tabla 16.

Niveles de factores seleccionados para instancias de 20 trabajos

	Factores			
Algoritmo	**Población**	**P. Cruce**	**P. Mutación**	**Iteraciones**
GA1	100	0,6	0,05	100
HGA1-VND	200	0,8	0,4	200
HGA1-GR	300	0,6	0,4	150

9.2.3. Instancias de 24 trabajos. Estas instancias cuentan con la mayor cantidad de trabajos posibles dentro del subgrupo de instancias medianas. Para llevar a cabo el análisis e identificación de la mejor combinación de parámetros se analiza en cada caso el promedio de tres instancias seleccionadas aleatoriamente.

Para el algoritmo *GA1*, se seleccionan las instancias *N24P4F3M5, N24P3F4M4 y N24P4F4M3*. A continuación, se realiza el análisis del diseño de experimentos y se obtiene como resultado la gráfica que se presenta en la Figura 38. De esta manera se identifica que se obtiene un menor valor de la variable *makespan* cuando se cuenta con un tamaño de población de 300, una probabilidad de cruce de 0,4, una probabilidad de mutación de 0,1 y un número de iteraciones de 200.

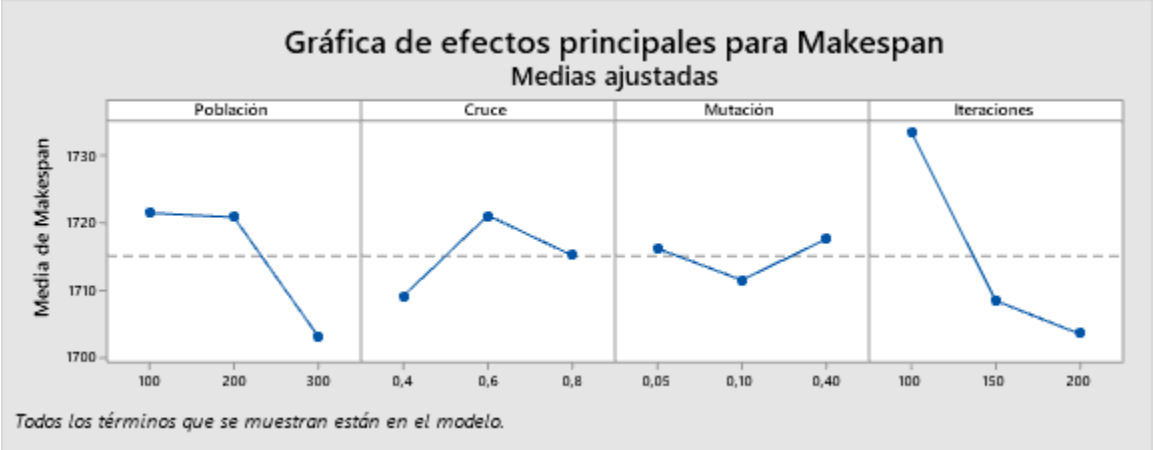


Figura 38. Efectos principales para *makespan* en instancias de 24 trabajos del *GA1*

En el caso del *HGA1-VND*, se realiza un análisis del diseño con ayuda de las instancias *N24P3F2M3, N24P3F3M5 y N24P4F4M4*. Dentro de los resultados, se obtiene la gráfica que se presenta en la Figura 39. A partir de esta gráfica, se identifica y define que un tamaño de población de 100, una probabilidad de cruce de 0,6, una probabilidad de mutación de 0,4 y un número de iteraciones de 100, logran minimizar

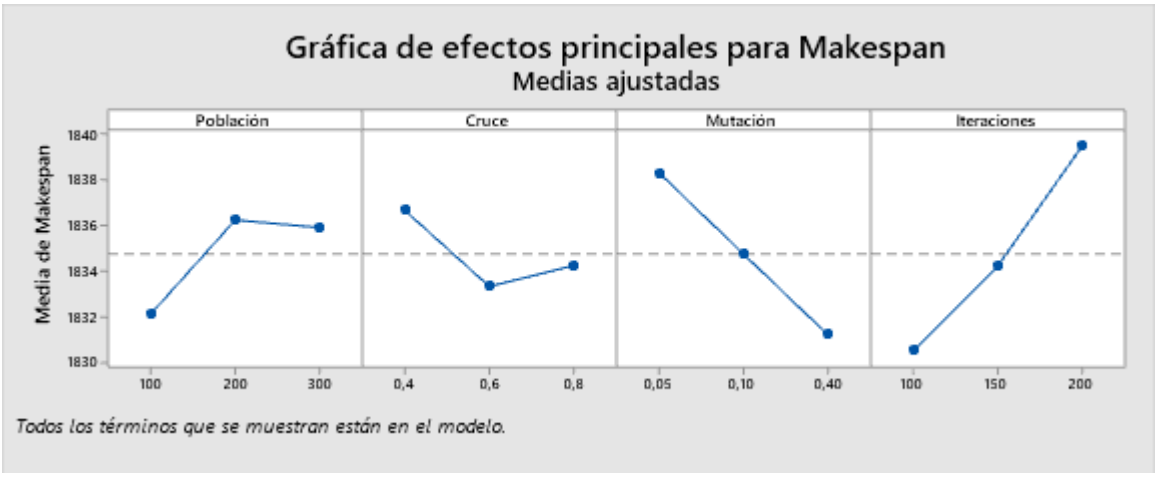


Figura 39. Efectos principales para *makespan* en instancias de 24 trabajos del *HGA1-VND*

Se realiza un análisis de las instancias *N24P4F4M2, N24P4F3M5 y N24P3F4M5* en donde se obtiene como resultado la gráfica que se presenta en la Figura 40. De esta gráfica se identifica que

para el *HGA1-GR* en instancias de 24 trabajos, se genera un menor tiempo de completamiento cuando se cuenta con una población de 300 individuos, una probabilidad de cruce de 0,8, una probabilidad de mutación de 0,4 y 200 iteraciones.

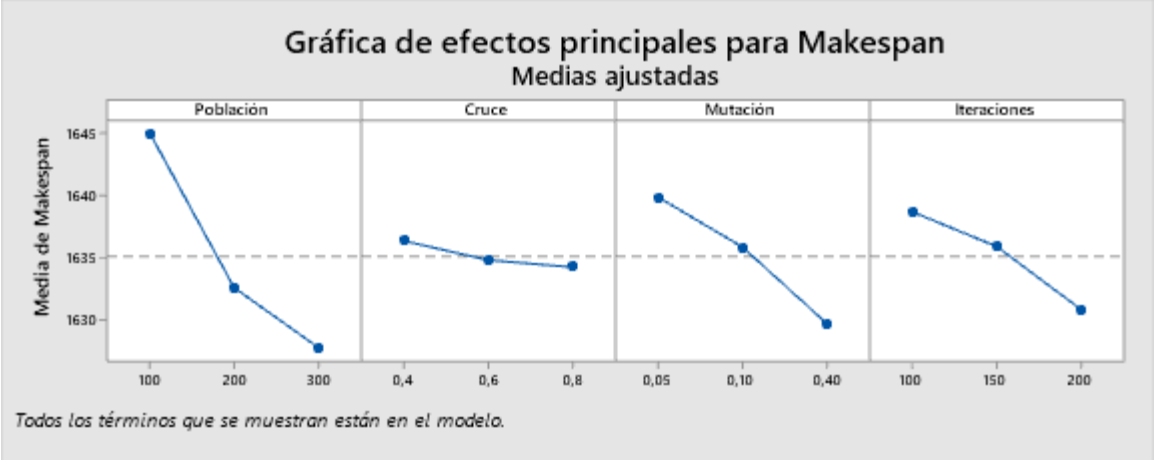


Figura 40. Efectos principales para *makespan* en instancias de 24 trabajos del *HGA1-GR*

A continuación, se presenta en la Tabla 17, un resumen de los niveles de factor seleccionados para cada uno de los tres algoritmos.

Tabla 17.

Niveles de factores seleccionados para instancias de 24 trabajos

	Factores			
Algoritmo	**Población**	**P. Cruce**	**P. Mutación**	**Iteraciones**
GA1	300	0,4	0,1	200
HGA1-VND	100	0,6	0,4	100
HGA1-GR	300	0,8	0,4	200

9.3.Instancias grandes

Este grupo de instancias contiene las instancias que presentan 100 y 200 trabajos. Debido a la cantidad de trabajos a procesar, el número de fábricas en las que se producen dichos trabajos, el número de máquinas en cada fábrica y finalmente los productos a ensamblar, son también superiores a las demás instancias, por esta razón el número de fábricas, *f*, varía entre 4,6 y 8; el número de máquinas, *m*, toma valores de 5,10 o 20, y el número de productos *p* puede ser de 30, 40 y 50.

Para la calibración, se corren 2 instancias de cada subgrupo (2 para las instancias de 100 trabajos y 2 para las instancias de 200 trabajos). Esto se ejecuta conforme a los niveles de factores descritos en la Tabla 13. El resultado del *makespan* de estas dos instancias se promedia y se analiza en Minitab de acuerdo con el diseño de experimentos 3^4 descrito al comienzo del capítulo 11.

Los niveles de factores se seleccionan con el análisis de la gráfica de efectos principales para *makespan*, donde se escoge el nivel del factor que presente el menor *makespan*.

9.3.1. Instancias de 100 trabajos. En la Figura 41, se presenta la gráfica de efectos principales para *makespan* en el algoritmo *GA1*. Se analizan las instancias de *N100-P40-F4-M5 y N100-P40-F4-M20* y se obtiene que los niveles de factor que generan un tiempo de completamiento menor son una población de 300 individuos, una probabilidad de cruce de 0,8, una probabilidad de mutación de 0,40 y un número de iteraciones de 100.

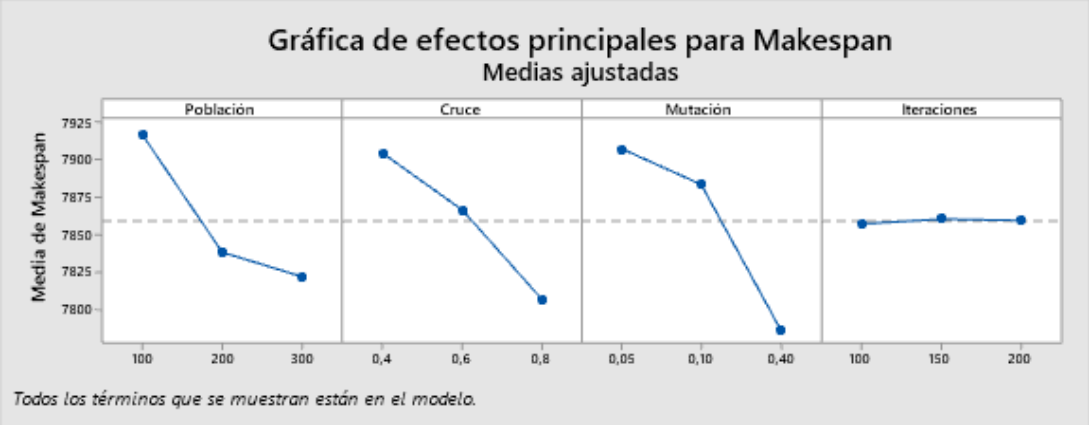


Figura 41. Efectos principales para *makespan* en instancias de 100 trabajos del *GA1*

Para la calibración *HGA1-VND*, se corren y analizan las instancias *N100-P40-F4-M5 y N100-P40-F4-M20* y se obtiene como resultado la gráfica de efectos principales para *makespan* se presenta en la Figura 42. De esta gráfica se obtienen los niveles de cada factor a tener en cuenta para este algoritmo con instancias de tamaño 100. Estos niveles son, tamaño de población igual a 300, probabilidad de cruce de 0,8, probabilidad de mutación de 0,4 y número de iteraciones de 150.

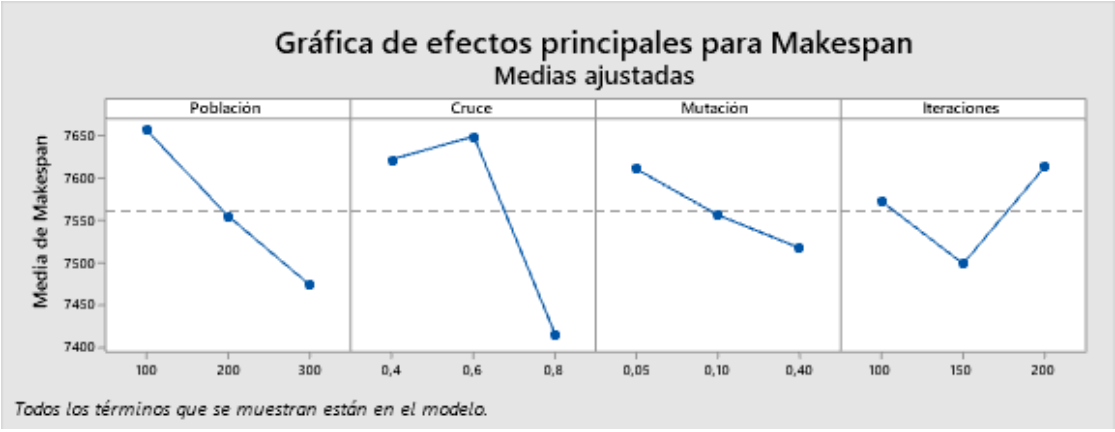


Figura 42. Efectos principales para *makespan* en instancias de 100 trabajos del *HGA1-VND*

Para el *HGA1-GR*, de nuevo se analizan las instancias *N100-P40-F4-M5 y N100-P40-F4-M20.* De este análisis y teniendo en cuenta la Figura 43 se identifica que, con un tamaño de población

de 300, una probabilidad de cruce 0,8, una probabilidad de mutación de 0,4 y un número de iteraciones de 200, se minimiza con mayor efectividad el valor generado para la variable respuesta *makespan*.

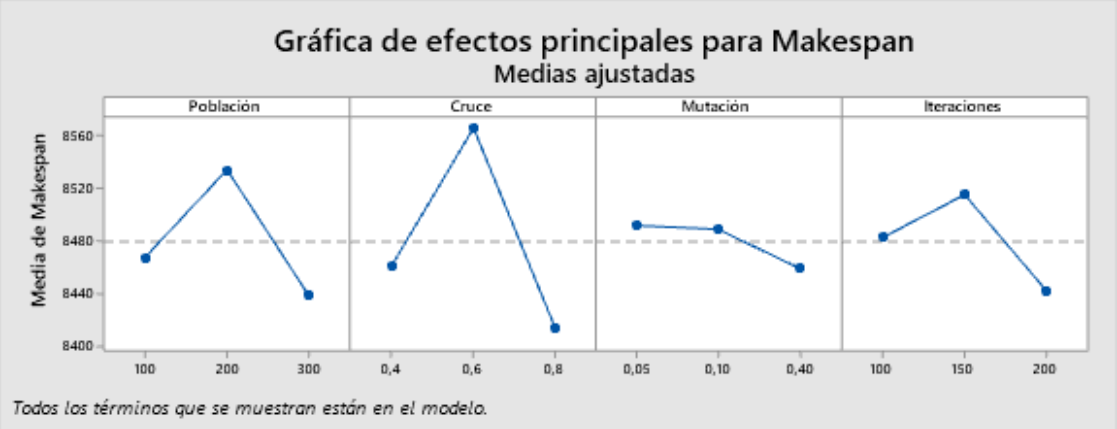


Figura 43. Efectos principales para *makespan* en instancias de 100 trabajos del HGA1- GR

A continuación, en la Tabla 18, se presentan los niveles de factores seleccionados para instancias de 100 trabajos en cada uno de los tres algoritmos.

Tabla 18.

Niveles de factores seleccionados para instancias de 100 trabajos

	Factores			
Algoritmo	**Población**	**P. Cruce**	**P. Mutación**	**Iteraciones**
GA1	200	0,8	0,4	100
HGA1-VND	200	0,8	0,4	150
HGA1-GR	300	0,8	0,4	200

9.3.2. Instancias de 200 trabajos. Las instancias analizadas para la calibración de cada uno de los tres algoritmos son *N200-P30-F4-M5 y N200-P50-F8-M10.* De este análisis se obtiene

como resultado la *Gráfica de efectos principales para makespan,* permitiendo así realizar la identificación de los mejores niveles de factor en cada caso.

En la Figura 44 se identifica que los niveles de factor que minimizan el objetivo de optimización del problema para un *GA1* son un tamaño de población de 300, una probabilidad de cruce de 0,8, una probabilidad de mutación de 0,1 y un número de iteraciones de 150.

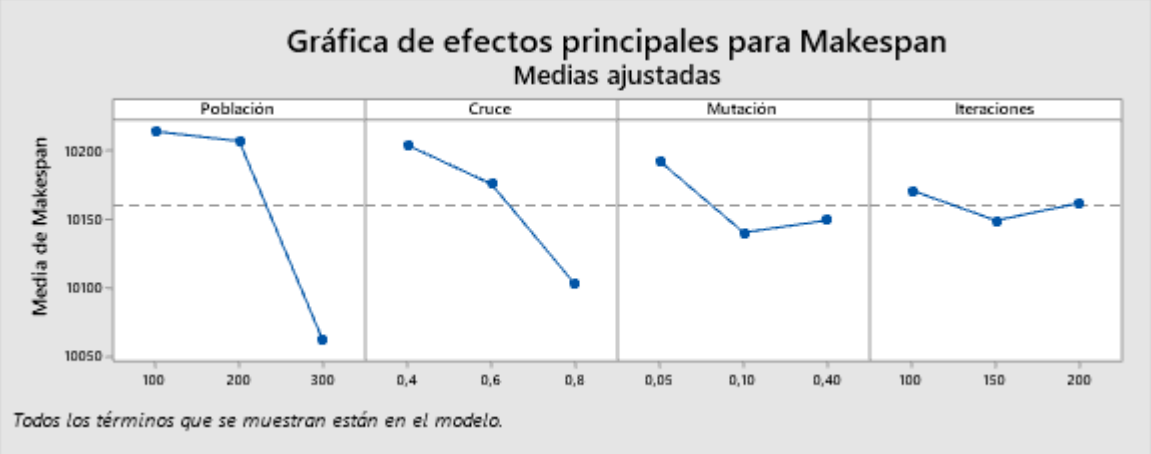


Figura 44. Efectos principales para *makespan* en instancias de 200 trabajos del GA1

Por otra parte, para el *HGA1-VND* se identifica a partir de la Figura 45 que una población de 200 individuos, una probabilidad de cruce de 0,8, una probabilidad de mutación de 0,4 y un número de iteraciones de 200, generan un valor menor en los resultados de la variable *makespan*.

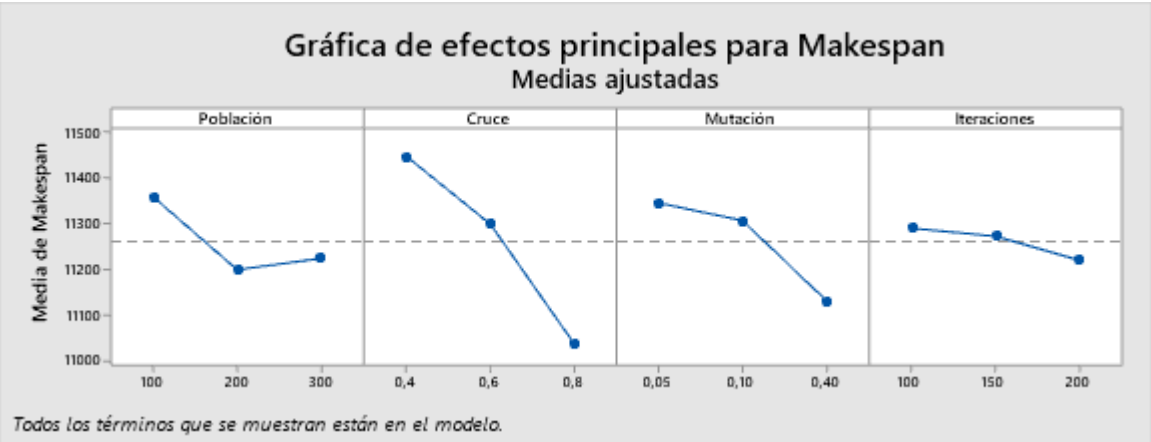


Figura 45. Efectos principales para *makespan* en instancias de 200 trabajos del HGA1-VND

Finalmente, en el caso del *HGA1-GR* se realiza un análisis e identificación de los niveles de factor que mejores resultados proporcionan sobre la variable *makespan* haciendo uso de la gráfica que se presenta en la Figura 46. De esta manera, es seleccionada una población de 300 individuos, una probabilidad de cruce de 0,8, una probabilidad de mutación de 0,4 y un número de iteraciones de 200.

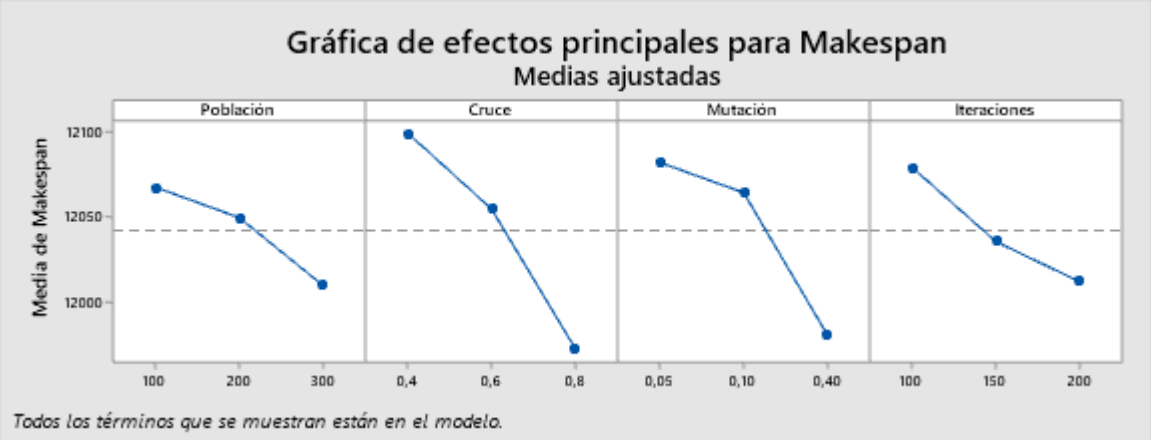


Figura 46. Efectos principales para *makespan* en instancias de 100 trabajos del HGA1-GR

A continuación, se presenta la Tabla 19, que contiene los niveles de factor seleccionados para cada uno de los tres algoritmos.

Tabla 19.

Niveles de factores seleccionados para instancias de 200 trabajos

	Factores			
Algoritmo	**Población**	**P. Cruce**	**P. Mutación**	**Iteraciones**
GA1	300	0,8	0,1	150
HGA1-VND	200	0,8	0,4	200
HGA1-GR	300	0,8	0,4	200

10. Evaluación de desempeño de los Algoritmos

Se realizar la evaluación del desempeño de cada uno de los tres algoritmos diseñados, a partir de la ejecución de 150 instancias. Estas se encuentran conformadas por un número de 24 instancias para los 4 subgrupos de pequeñas (N=8 Y N=12) y medianas (N=16 Y N=24) y de 27 para los 2 subgrupos de grandes (N=100 Y N=200). Por otra parte, no se tienen en cuenta las instancias de 20 trabajos para el análisis, esto debido a que en la literatura no existen resultados de estas instancias para llevar a cabo la comparación del desempeño.

Cada uno de los algoritmos es ejecutado en un ordenador con procesador Core i7-8700 de 3,2 GHz, con 8Gb de memoria RAM y sistema operativo Windows 10.

Teniendo en cuenta los resultados obtenidos para cada una de las instancias en cada uno de los tres algoritmos, se realiza una comparación de los valores conseguidos en la variable respuesta *makespan*, con los valores existentes en la literatura de instancias para fábricas heterogéneas. Estas instancias, son generadas con distribuciones normales aleatorias por Correa y Ortíz en el 2017.

Como estadístico de comparación se utiliza el *Porcentaje de desviación relativa (RPD)*. Este se calcula respecto a las soluciones generadas por Correa y Ortíz en el 2017, quienes utilizan en su investigación un algoritmo VND y programación lineal entera mixta (MILP) para dar solución al DAPFSP-SDST. De esta manera, los resultados conseguidos por cada uno de los tres algoritmos para las instancias pequeñas y medianas se comparan con los generados por el VND y el MILP, mientras que, para las instancias grandes, únicamente se comparan con los resultados generados por el VND. En el apéndice I se muestra el cálculo del indicador RPD para los tres algoritmos. Por

otra parte, el RPD calcula la diferencia entre la solución del algoritmo y la mejor solución con respecto a la mejor solución. A continuación, se presenta la expresión matemática del RPD.

$$RPD = \frac{ALGsol - Best}{Best} * 100 \quad (32)$$

$$Best = Min(SOLteórica; SOLalg) \quad (33)$$

10.1. Análisis del desempeño para el Algoritmo Genético GA1

En la Tabla 20, se observa que para las instancias pequeñas con 8 trabajos el 42% y 83% de las instancias presenta mejores resultados al ser ejecutadas con el GA1, en comparación con los resultados obtenidos con el MILP y VND respectivamente. Por otra parte, para el subgrupo de 12 trabajos se identifica que este algoritmo comparado con el VND genera peores resultados para el 54% de las instancias.

Se infiere que para las instancias medianas de 16 trabajos son mejores los resultados del 50% de las instancias respecto al MILP y el 54% respecto al VND. De igual manera en el subgrupo de 24 trabajos se presenta mejoras en aproximadamente un 56% de las instancias en comparación con el VND y el MILP.

Para el subgrupo de 100 trabajos en instancias grandes, se mejoran los resultados del 42% de las instancias y al no presentar resultados idénticos entre los dos algoritmos, se obtienen peores resultados en el 58% de las instancias. Por otra parte, para instancias de 200 trabajos se consiguen resultados superiores de *makespan* en el 79% de las instancias al ser ejecutadas con el GA1.

Tabla 20.

Comparación de instancias GA1

		Número de instancias	Porcentaje de instancias	RPD
8	SMILP>SGA1	14	58%	
	SMILP=SGA1	6	25%	
	SMILP<SGA1	4	17%	3.72%
	SVND>SGA1	14	58%	
	SVND=SGA1	6	25%	
	SVND<SGA1	4	17%	2.59%
12	SMILP>SGA1	14	58%	
	SMILP=SGA1	4	17%	
	SMILP<SGA1	6	25%	5.27%
	SVND>SGA1	7	29%	
	SVND=SGA1	4	17%	
	SVND<SGA1	13	54%	5.02%
16	SMILP>SGA1	12	50%	
	SMILP=SGA1	4	17%	
	SMILP<SGA1	8	33%	4.45%
	SVND>SGA1	13	54%	
	SVND=SGA1	3	13%	
	SVND<SGA1	8	33%	3.63%
24	SMILP>SGA1	14	58%	

Continuación Tabla 20

Comparación de instancias GA1

		Número de instancias	Porcentaje de instancias	RPD
	SMILP=SGA1	0	0%	
	SMILP<SGA1	10	42%	4.39%
	SVND>SGA1	13	54%	
	SVND=SGA1	0	0%	
	SVND<SGA1	11	46%	8.45%
	SVND>SGA1	10	42%	
100	SVND=SGA1	0	0%	
	SVND<SGA1	14	58%	8.39%
	SVND>SGA1	5	21%	
200	SVND=SGA1	0	0%	
	SVND<SGA1	19	79%	11.94%

Finalmente, en la Figura 47, se observa el comportamiento del RPD, para cada uno de los subgrupos de instancias analizadas. De esta figura, se identifica que el algoritmo GA1 tiene un porcentaje de desviación relativa que oscila entre el 3,8% y 5%, esto sucede cuando se realizan comparaciones respecto a las soluciones generadas por el MILP en instancias pequeñas y medianas. Por otra parte, el porcentaje de desviación relativa tiene un comportamiento creciente cuando se compara GA1 vs VND, presentando porcentajes de RPD superiores al 8% en instancias con 24, 100 y 200 trabajos.

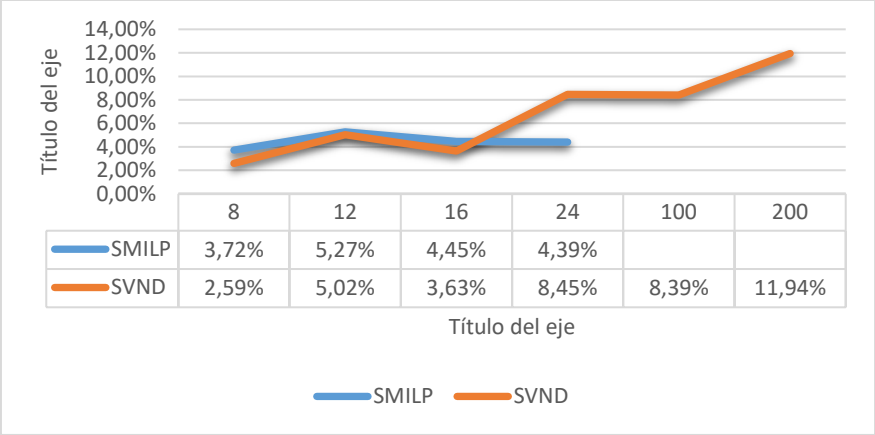


Figura 47. RPD para el GA1

10.2. Análisis del desempeño para el Algoritmo híbrido entre Algoritmo Genético y VND (HGA1-VND)

En el caso del Algoritmo híbrido entre Algoritmo Genético y VND (HGA1-VND) se identifica a partir de la Tabla 22., que para instancias de 8 trabajos se obtienen resultados de *makespan* superiores en el 63% y 71% de instancias comparando con el MILP y VND, respectivamente. Por otra parte, para instancias de 12 trabajos, se obtienen mejores resultados en el 54% de las instancias respecto al MILP y el 50% comparando con el VND.

Para las instancias medianas, en el subgrupo de 12 trabajos se obtienen resultados de *makespan* superiores en el 46% de las instancias en comparación con el MILP y VND. Mientras que en el subgrupo de 24 trabajos se consigue mejorar los resultados en un 63% de las instancias con relación al MILP y un 58% respecto al VND.

Finalmente, para las instancias grandes se identifica un mayor porcentaje de mejora de los resultados en instancias de 100 y 200 con este algoritmo. Sin embargo, el porcentaje de instancias con resultados más elevados en el *makespan* respecto al algoritmo VND es de 58% para el subgrupo de 100 trabajos y de 71% para el de 200 trabajos.

Tabla 21.

Comparación de instancias HAG1-VND

		Número de instancias	Porcentaje de instancias	RPD
8	SMILP>SHGA1-VND	15	63%	2.33%
	SMILP=SHGA1-VND	5	21%	
	SMILP<SHGA1-VND	4	17%	
	SVND>SHGA1-VND	17	71%	1.15%
	SVND=SHGA1-VND	4	17%	
	SVND<SHGA1-VND	3	13%	
12	SMILP>SHGA1-VND	13	54%	4.61%
	SMILP=SHGA1-VND	6	25%	
	SMILP<SHGA1-VND	5	21%	
	SVND>SHGA1-VND	12	50%	2.83%
	SVND=SHGA1-VND	6	25%	
	SVND<SHGA1-VND	6	25%	
16	SMILP>SHGA1-VND	11	46%	3.14%
	SMILP=SHGA1-VND	6	25%	
	SMILP<SHGA1-VND	7	29%	
	SVND>SHGA1-VND	11	46%	3.01%
	SVND=SHGA1-VND	6	25%	
	SVND<SHGA1-VND	7	29%	

Continuación Tabla 21

Comparación de instancias HAG1-VND

		Número de instancias	Porcentaje de instancias	RPD
24	SMILP>SHGA1-VND	15	63%	
	SMILP=SHGA1-VND	0	0%	
	SMILP<SHGA1-VND	9	38%	1.38%
	SVND>SHGA1-VND	14	58%	
	SVND=SHGA1-VND	1	4%	
	SVND<SHGA1-VND	9	38%	6.57%
100	SVND>SHGA1-VND	10	42%	
	SVND=SHGA1-VND	0	0%	
	SVND<SHGA1-VND	14	58%	8.09%
200	SVND>SHGA1-VND	7	29%	
	SVND=SHGA1-VND	0	0%	
	SVND<SHGA1-VND	17	71%	10.95%

En la Figura 48 se observa el comportamiento del RPD para el HGA1-VND. Ahora bien, teniendo en cuenta esta figura, se identifica que el porcentaje de desviación relativa del HGA1-VND es menos disperso a medida que se aumenta el número de trabajos en las instancias pequeñas y medianas. Por otra parte, el RPD para las instancias grandes es directamente proporcional al aumento en el número de trabajos presente en cada instancia, obteniendo así un aumento significativo en el porcentaje de desviación.

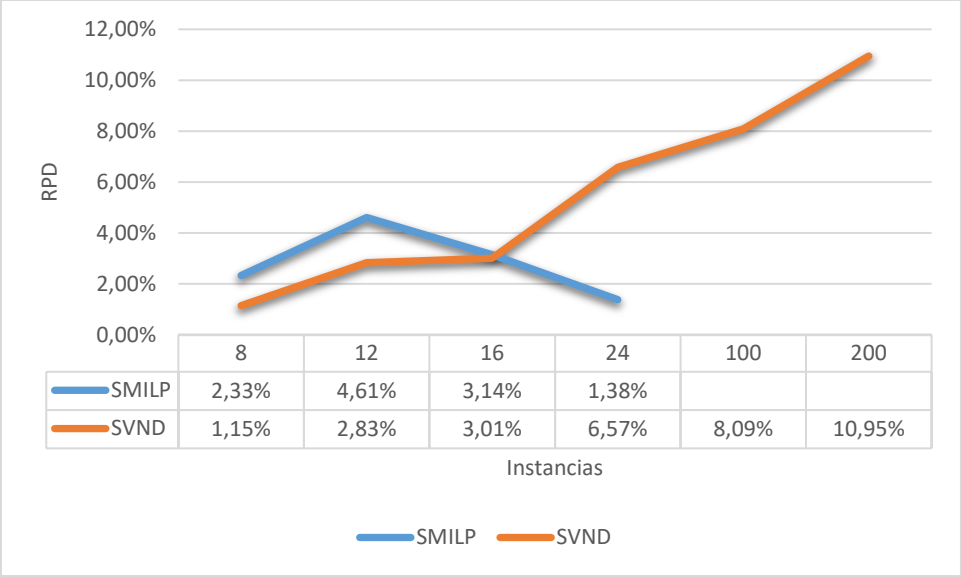


Figura 48. RPD para el HGA1-VND

10.3. Análisis del desempeño para el Algoritmo híbrido entre Algoritmo Genético y Algoritmo Voraz (HGA1-GR)

A partir del análisis que se realiza con ayuda de la Tabla 22, se identifica que las instancias de 8 trabajos presentan un 58% de instancias mejoradas comparando el HGA1-GR con el MILP y que este porcentaje aumenta a 63% cuando se contrastan los resultados generados por este algoritmo con respecto al VND. Para el subgrupo de 12 trabajos, se observa que comparando SHGA1-VND con SMILP y SVND se genera una disminución en los valores del *makespan* para el 63% de las instancias.

Dentro de las instancias medianas, para el subgrupo de 16 trabajos se mejoran los resultados del 50% de las instancias con el HGA1-GR comparando tanto con MILP como con VND. Por otra parte, en el subgrupo con 24 trabajos se consigue tener un 58% y 54% de instancias mejoradas realizando la comparación con el MILP y el VND respectivamente.

Las instancias de los subgrupos de 100 y 200 trabajos presentan un porcentaje menor al 40% de instancias mejoradas con respecto a los resultados obtenidos con el VND, esto indica que al ejecutar estas instancias con el HGA1-GR se obtienen valores de *makespan* muy elevados para más del 60% de las instancias.

Tabla 22.

Comparación instancias HAG1-GR

		Número de instancias	**Porcentaje de instancias**	**RPD**
8	SMILP>SHGA1 -GR	14	58%	2.75%
	SMILP=SHGA1 -GR	5	21%	
	SMILP<SHGA1 -GR	5	21%	
	SVND>SHGA1 -GR	15	63%	3.03%
	SVND=SHGA1 -GR	5	21%	
	SVND<SHGA1 -GR	4	17%	
12	SMILP>SHGA1 -GR	13	54%	5.11%
	SMILP=SHGA1 -GR	4	17%	
	SMILP<SHGA1 -GR	7	29%	
	SVND>SHGA1 -GR	13	54%	5.63%
	SVND=SHGA1 -GR	5	21%	
	SVND<SHGA1 -GR	6	25%	
16	SMILP>SHGA1 -GR	12	50%	4.78%
	SMILP=SHGA1 -GR	3	13%	
	SMILP<SHGA1 -GR	9	38%	

Continuación Tabla 22

Comparación instancias HAG1-GR

		Número de instancias	Porcentaje de instancias	RPD
	SVND>SHGA1 -GR	12	50%	
	SVND=SHGA1 -GR	5	21%	
	SVND<SHGA1 -GR	7	29%	5.43%
	SMILP>SHGA1 -GR	14	58%	
	SMILP=SHGA1 -GR	0	0%	
	SMILP<SHGA1 -GR	10	42%	5.46%
24	SVND>SHGA1 -GR	13	54%	
	SVND=SHGA1 -GR	0	0%	
	SVND<SHGA1 -GR	11	46%	6.95%
	SVND>SHGA1 -GR	9	38%	
100	SVND=SHGA1 -GR	0	0%	
	SVND<SHGA1 -GR	15	63%	9.62%
	SVND>SHGA1 -GR	4	17%	
200	SVND=SHGA1 -GR	0	0%	
	SVND<SHGA1 -GR	20	83%	11.85%

Por otra parte, el comportamiento del RPD para el HGA1 – GR se presenta en la Figura 49. A partir de esta gráfica, se identifica que el comportamiento del porcentaje de desviación relativa aumenta, a medida que aumenta el número de trabajos existentes en las instancias. Esto significa,

que a medida que el tamaño de las instancias aumenta, la desviación de la solución generada por el algoritmo propuesto está más distante de la mejor solución arrojada por el VND.

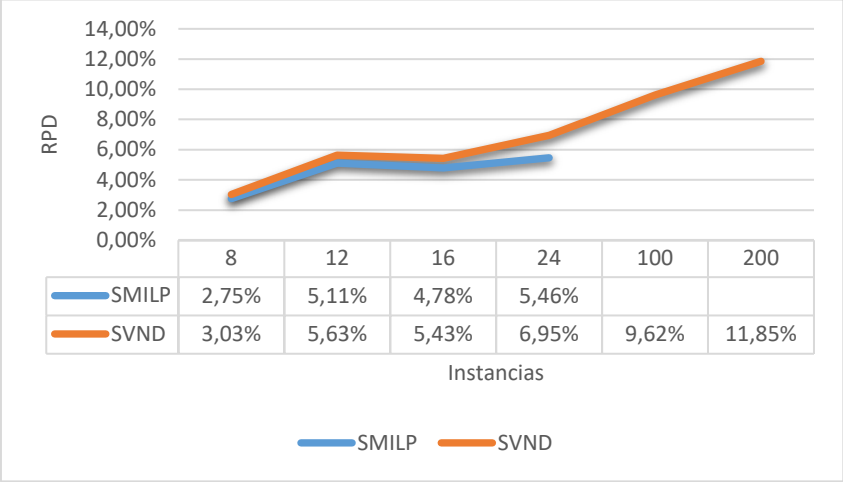


Figura 49. RPD para el HGA1-GR

10.4. Comparación del rendimiento de los tres algoritmos

A continuación, en la Tabla 23, se presenta el resumen del porcentaje de instancias en las que cada uno de los tres algoritmos genera mejores soluciones, respecto al MILP y el VND. De esta tabla se identifica que el algoritmo híbrido entre Algoritmo Genético y VND (HGA1 – VND) genera un rendimiento superior al 4% con respecto a los otros dos algoritmos diseñados. Por lo tanto, se establece que en la presente investigación el HGA1 – VND es el algoritmo que mejor se adapta para dar solución al DAPFSP-SDST con fábricas heterogéneas.

Tabla 23.

Porcentaje de instancias mejoradas

	GA1	**HGA1-VND**	**HGA1-GR**
SMILP	70,83%	73,96%	67,71%
SVND	54,00%	58,67%	54,00%

11. Resultados de los algoritmos

Para obtener resultados de cada uno de los tres algoritmos se ejecutaron 150 instancias, 24 instancias para cada uno de los 4 subgrupos de instancias pequeñas y medianas y 27 instancias para los subgrupos de 100 y 200 trabajos. De esta manera, se presentan 450 resultados para la totalidad de los algoritmos diseñados.

A continuación, se presenta en la Tabla 24, el valor mínimo y máximo conseguido para cada subgrupo de instancias en ejecutadas con el algoritmo GA1, acompañado por su respectiva media y desviación estándar.

Tabla 24.

Resúmenes resultados GA1

N	Min	Max	Media	Desviación
8	362	792	572.75	117.432242
12	480	1128	787.5	164.752251
16	585	2164	1503.375	380.951191
24	585	2164	1503.375	380.951191
100	209	9908	5667.2963	1481.51188
200	9019	15462	12040.5926	1564.74902

En la Figura 50, se observa la relación de la desviación estándar muestral obtenida para las instancias en el algoritmo GA1, y se identifica la existencia de una relación directamente proporcional entre el aumento en la desviación de los valores obtenidos para la variable *makespan* y el número de trabajos que contiene cada subgrupo de instancias.

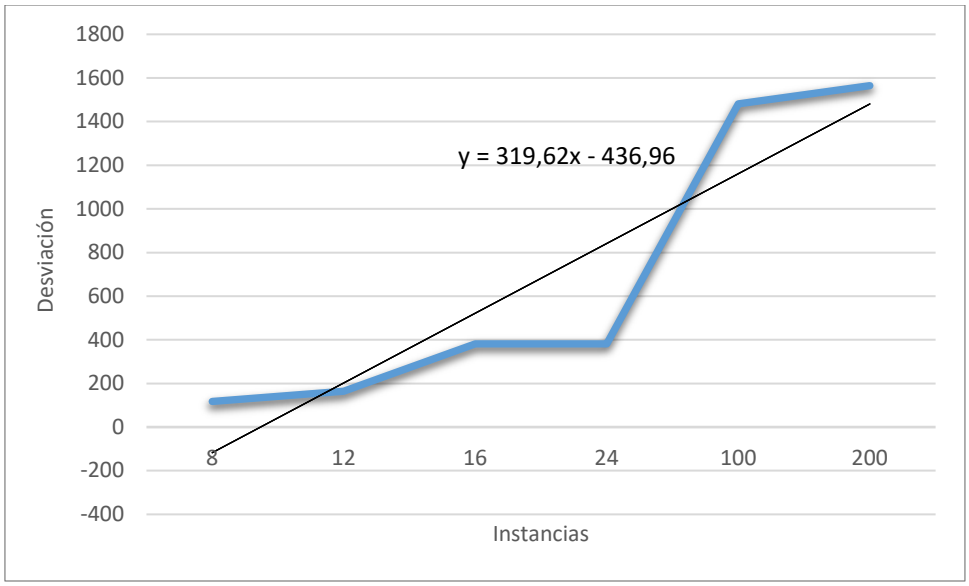


Figura 50. Desviación vs Instancias para GA1

Por otra parte, en la Tabla 25, se presenta el resumen de los resultados obtenidos para las instancias al ser ejecutadas con el algoritmo HGA1-VND. En esta tabla se observan los valores mínimos y máximos, la media y la desviación estándar para cada subgrupo de instancias.

Tabla 25.

Resúmenes resultados HGA1-VND

N	Min	Max	Media	Desviación
8	255	792	500.3333333	164.8998818
12	337	1128	732.4583333	182.2718894
16	350	1542	1012	372.6133229
24	756	2190	1478.583333	372.6133229
100	4307	7850	5711.740741	893.1920111
200	9008	13230	11769.48148	988.9844974

La Figura 51, muestra el comportamiento creciente de la desviación estándar muestral de la variable *makespan*, para las instancias procesadas con el algoritmo HGA1-VND.

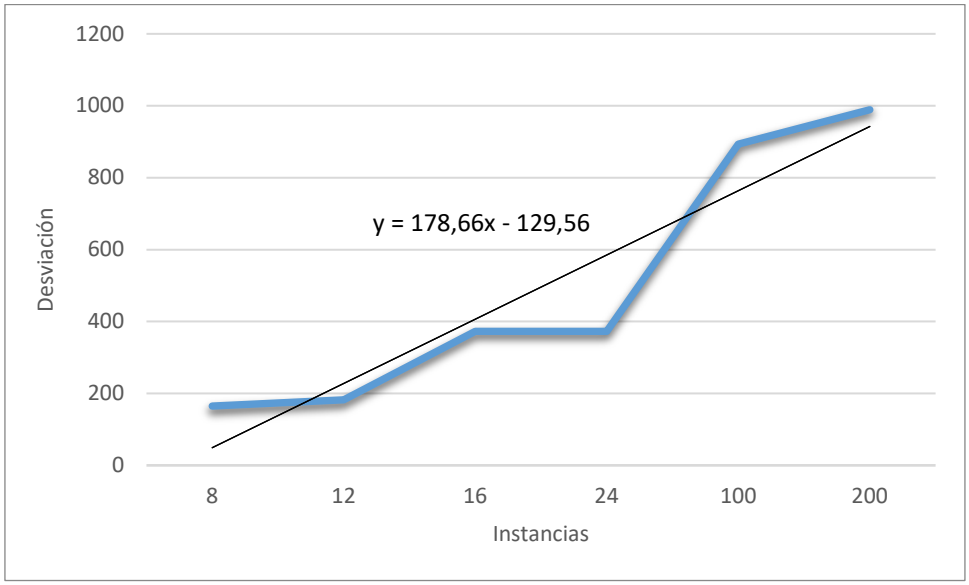


Figura 51. Desviación vs instancias para HGA1-VND

Finalmente se presenta, un pequeño resumen en la Tabla 26, de los resultados obtenidos para las 150 instancias a partir de la ejecución de estas con el algoritmo HGA1- GR

Tabla 26.

Resúmenes resultados HGA1-GR

N	Min	Max	Media	Desviación
8	366	792	572.7083333	108.20491
12	507	1129	832.1666667	180.010306
16	443	1550	1070	366.772051
24	633	2099	1517.458333	366.772051
100	4732	7088	5861.592593	570.378434
200	9875	16812	12051.33333	1631.96658

De igual manera que en los resultados conseguidos por los otros algoritmos, se observa en la Figura 52, que el comportamiento de la desviación estándar muestral del valor de *makespan* para las instancias analizadas, también aumenta proporcionalmente con respecto a la cantidad de trabajos que contienen las instancia.

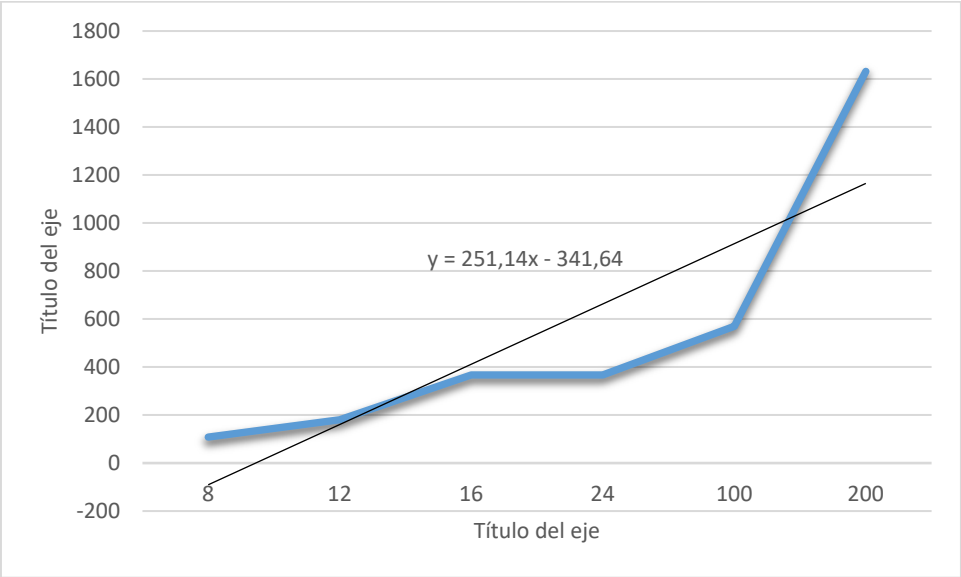


Figura 52. Desviación vs Instancias para HGA1-GR

Estos resultados se ejecutan con el objetivo de dejar bases de comparación para investigaciones futuras sobre el mismo problema de optimización (DAPFSP-SDST). Asimismo, en el Apéndice J, se presentan las soluciones completas, con su respectivo *makespan*.

12. Conclusiones

A partir del análisis bibliométrico se identifica el aumento significativo en la participación de múltiples autores en publicaciones relacionadas con el DAPFSP-SDST, donde el porcentaje de participación de este tipo de publicaciones se encuentra aproximadamente en el 34% desde el

2016. Por otra parte, la revisión de literatura valida la necesidad de proponer diferentes métodos de solución y estructuras para el tratamiento del problema de investigación como el DAPFSP-SDST con fábricas heterogéneas.

Del diseño de experimentos realizado para los tres algoritmos se logra concluir que los factores, tamaño de población, probabilidad de cruce, probabilidad de mutación y número de iteraciones, presentan efectos significativos en los resultados que se obtienen para la variable respuesta *makespan*. Estos factores presentan una tendencia de aumento en los valores de sus niveles, a medida que aumenta el tamaño de las instancias analizadas.

A partir de la calibración de los algoritmos, se obtiene que las instancias pequeñas de 8 trabajos mejoran en un 60% su respuesta con niveles de tamaño de población y números de iteraciones bajos y probabilidad de cruce y mutación alta y media, respectivamente. Mientras que para las instancias con 12 trabajos se consiguen mejoras del 2% en los resultados cuando se trabaja con un nivel alto para la probabilidad de cruce, y niveles medios para la probabilidad de mutación, tamaño de población y número de iteraciones.

Las instancias medianas de 16 trabajos mejoran las soluciones en un 1,8% con respecto a la media cuando el tamaño de la población trabaja con un nivel alto, mientras que la probabilidad de mutación, probabilidad de cruce y número de iteraciones utilizan niveles medios. Por otra parte, las instancias de 24 trabajos presentan 2,5 % de mejoras en el rendimiento de los resultados obtenidos, cuando la probabilidad de cruce utiliza un nivel medio y los demás factores se ejecutan con su respectivo nivel alto.

Para las instancias grandes de 100 trabajos, los resultados obtenidos presentan mejoras del 10% al trabajar con niveles medios para el tamaño de población y número de iteraciones y niveles

altos para las probabilidades de mutación y cruce. Sin embargo, las instancias de 200 trabajos consiguen 6% de mejoras en las soluciones de la variable *makespan*, cuando los niveles de cada factor son altos.

Del análisis del desempeño de GA1, se concluye que, para instancias pequeñas y medianas, el porcentaje de instancias que presentan mejores soluciones es en promedio del 54 %. Por otra parte, para instancias grandes presentan en promedio mejoras del 30% en las soluciones generadas.

Para el caso del HGA1-GR se tiene que alrededor del 50% de las instancias pequeñas y medianas mejoran sus resultados al ser ejecutadas con este algoritmo. Para las instancias grandes de 100 trabajos se observa únicamente una mejora del 38% y para las de 200 una mejora del 17%.

El algoritmo HGA1-VND presenta un desempeño superior al 4% comparado con los resultados obtenidos al ejecutar las instancias con los otros dos algoritmos diseñados para la presente investigación y los resultados existentes haciendo uso del MILP y el algoritmo VND.

El tamaño de las instancias genera un impacto significativo en las soluciones conseguidas para el DAPFSP-SDST con fábricas heterogéneas, debido a que dependiendo de la cantidad trabajos N, productos P, fábricas F y máquinas M, la variable *makespan* presenta una alta variación en sus resultados. Sin embargo, se observa que existe mayor variabilidad en las soluciones a medida que el número trabajos N aumenta.

13. Recomendaciones

Para futuras investigaciones se recomienda:

Solucionar el DAPFSP-SDST teniendo en cuenta otras consideraciones que lo acerquen más a la realidad de la industria, como lo son los costos de tiempos de espera, capacidad de almacenamiento y transporte entre fábricas.

Contemplar una función multiobjetivo para la solución del DAPFSP-SDST, la cual podría estar relacionada con la minimización de costos (producción, transporte, recursos, almacenamiento, etc.), y a su vez valorar los beneficios de entregas prontas. De esta manera se podría evaluar el problema teniendo presente la utilidad generada.

Implementar el algoritmo genético con otra función fitness o de adaptación, para permitir la selección de individuos con resultados de *makespan* desfavorables, y así evitar la convergencia anticipada a los óptimos locales.

Diseñar un algoritmo hibrido entre genético y VND, que contenga más de una estructura de VND, como lo puede ser el intercambio de trabajos dentro de una misma fábrica.

Estudiar, implementar o crear nuevos operadores para el algoritmo genético que permitan la búsqueda de mejores resultados para el DAPFSP-SDST.

Referencias Bibliográficas

Alberto, E., & Hernandez, G. (2017). Revisión de la Literatura Sobre el Problema de Programación de “ Flow Shop ” Híbrido con Máquinas Paralelas no Relacionadas Literature Review on the Hybrid Flow Shop Scheduling Problem with Unrelated Parallel Machines, 9–22.

Allahverdi, A., & Soroush, H. M. (2008). The significance of reducing setup times/setup costs. *European Journal of Operational Research*, *187*(3), 978–984. https://doi.org/10.1016/j.ejor.2006.09.010

Bargaoui, H., Belkahla Driss, O., & Ghédira, K. (2017). A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion. *Computers and Industrial Engineering*, *111*, 239–250. https://doi.org/10.1016/j.cie.2017.07.020

Bargaoui, H., Driss, O. B., & Ghédira, K. (2017). Towards a Distributed Implementation of Chemical Reaction Optimization for the Multi-factory Permutation Flowshop Scheduling Problem. *Procedia Computer Science*, *112*, 1531–1541. https://doi.org/10.1016/j.procs.2017.08.057

Basir, S. A., Mazdeh, M. M., & Namakshenas, M. (2018). Bi-level genetic algorithms for a two-stage assembly flow-shop scheduling problem with batch delivery system. *Computers and Industrial Engineering*, *126*(July), 217–231. https://doi.org/10.1016/j.cie.2018.09.035

Benbouzid-Si Tayeb, F., Bessedik, M., Benbouzid, M., Cheurfi, H., & Blizak, A. (2017). Research on Permutation Flow-shop Scheduling Problem based on Improved Genetic Immune Algorithm with vaccinated offspring. *Procedia Computer Science*, *112*, 427–436.

https://doi.org/10.1016/j.procs.2017.08.055

Chakravorty, A., & Laha, D. (2017). A heuristically directed immune algorithm to minimize makespan and total flow time in permutation flow shops. *International Journal of Advanced Manufacturing Technology*, *93*(9–12), 3759–3776. https://doi.org/10.1007/s00170-017-0679-1

Chen, P., Huang, H. kuan, & Dong, X. Y. (2010). Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem. *Expert Systems with Applications*, *37*(2), 1620–1627. https://doi.org/10.1016/j.eswa.2009.06.047

Correa González, M. E., Ortíz Delgado, D. F., & Garavito Hernández, E. A. (2017). Solución del Problema de Flowshop Distribuido y Permutado con Etapa de Ensamble considerando tiempos de alistamiento dependientes de la secuencia (DAPFSP-SDST) y fábricas heterogéneas a través de un algoritmo basado en VND.

Deng, J., Wang, L., Wang, S. Y., & Zheng, X. L. (2016). A competitive memetic algorithm for the distributed two-stage assembly flow-shop scheduling problem. *International Journal of Production Research*, *54*(12), 3561–3577. https://doi.org/10.1080/00207543.2015.1084063

Dowsland, K. A., & Diaz, A. (2003). Heuristic design and fundamentals of the Simulated Annealing. *Inteligencia Artificial*, *7*(19). https://doi.org/10.4114/ia.v7i19.718

Fernandez-Viagas, V., Valente, J. M. S., & Framinan, J. M. (2018). Iterated-greedy-based algorithms with beam search initialization for the permutation flowshop to minimise total tardiness. *Expert Systems with Applications*, *94*(November), 58–69. https://doi.org/10.1016/j.eswa.2017.10.050

Fuchigami, H. Y., Sarker, R., & Rangel, S. (2018). Near-optimal heuristics for just-in-time jobs maximization in flow shop scheduling. *Algorithms*, *11*(4), 1–17. https://doi.org/10.3390/a11040043

Gibergans Bàguena, J. (2009). El análisis de la varianza (ANOVA). *Módulo 12 En Estadística*, 1–6.

González-Neira, E. M., Montoya-Torres, J. R., & Barrera, D. (2017). Flow-shop scheduling problem under uncertainties: Review and trends. *International Journal of Industrial Engineering Computations*, *8*(4), 399–426. https://doi.org/10.5267/j.ijiec.2017.2.001

Hansen, P., Mladenovic, N., & Moreno Perez, J. A. (2003). Búsqueda de Entorno Variable Inteligencia Artificial . Revista Iberoamericana de Inteligencia Artificial Asociación Española para la Inteligencia Artificial Asociación Española para la Inteligencia Artificial Valencia , España, (January).

Jian, J., Pan, S., & Yang, L. (2019). Solution for short-term hydrothermal scheduling with a logarithmic size mixed-integer linear programming formulation. *Energy*, *171*, 770–784. https://doi.org/10.1016/j.energy.2019.01.038

Jiang, H., Li, S., Hu, J., Hu, J., & Wei, H. (2018). Hybrid Genetic Simulated Annealing Algorithm for Improved Flow Shop Scheduling with Makespan Criterion. *Applied Sciences*, *8*(12), 2621. https://doi.org/10.3390/app8122621

Johnson, S. M. (1954). Optimal Two-And Three-Stage Production Schedules With Setup Times Included. *Naval Research Logistics Quarterly*, *1*, 61–68.

Kazemi, H., Mazdeh, M. M., & Rostami, M. (2017). The two stage assembly flow-shop scheduling

problem with batching and delivery. *Engineering Applications of Artificial Intelligence*, *63*, 98–107. https://doi.org/10.1016/j.engappai.2017.05.004

Komaki, G. M., Teymourian, E., Kayvanfar, V., & Booyavi, Z. (2017). Improved discrete cuckoo optimization algorithm for the three-stage assembly flowshop scheduling problem. *Computers and Industrial Engineering*, *105*, 158–173. https://doi.org/10.1016/j.cie.2017.01.006

Li, X., Yang, Z., Ruiz, R., Chen, T., & Sui, S. (2018). An iterated greedy heuristic for no-wait flow shops with sequence dependent setup times, learning and forgetting effects. *Information Sciences*, *453*, 408–425. https://doi.org/10.1016/j.ins.2018.04.038

Lin, J., Wang, Z. J., & Li, X. (2017). A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem. *Swarm and Evolutionary Computation*, *36*(January), 124–135. https://doi.org/10.1016/j.swevo.2017.04.007

Liu, S.-C., Lai, K., Wu, C.-C., Yu, P.-W., Lin, W.-C., & Chen, J.-Y. (2018). A two-stage three-machine assembly flow shop scheduling with learning consideration to minimize the flowtime by six hybrids of particle swarm optimization. *Swarm and Evolutionary Computation*, *41*(January), 97–110. https://doi.org/10.1016/j.swevo.2018.01.012

Low, C., Hsu, C. J., & Su, C. T. (2008). A two-stage hybrid flowshop scheduling problem with a function constraint and unrelated alternative machines. *Computers and Operations Research*, *35*(3), 845–853. https://doi.org/10.1016/j.cor.2006.04.004

Lv, C., & Lei, D. (2018). Hybrid flow shop scheduling with assembly operations and key objectives: A novel neighborhood search. *Proceedings of the 30th Chinese Control and Decision Conference, CCDC 2018*, *61*, 4885–4890.

https://doi.org/10.1109/CCDC.2018.8407977

Mansouri, S. A., Aktas, E., & Besikci, U. (2016). Green scheduling of a two-machine flowshop: Trade-off between makespan and energy consumption. *European Journal of Operational Research*, *248*(3), 772–788. https://doi.org/10.1016/j.ejor.2015.08.064

Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers and Operations Research*, *37*(4), 754–768. https://doi.org/10.1016/j.cor.2009.06.019

Peña-tibaduiza, E., Garavito-hernández, E., Moratto-chimenty, E., & Pérez-figueredo, L. (2014). A meta-heuristic based on the imperialist competitive algorithm (ICA) for solving Hybrid Flow Shop (HFS) scheduling problem with unrelated parallel machines, *81*(184), 1–2.

Peng, K., Wen, L., Li, R., Gao, L., & Li, X. (2018). An Effective Hybrid Algorithm for Permutation Flow Shop Scheduling Problem with Setup Time. *Procedia CIRP*, *72*, 1288–1292. https://doi.org/10.1016/j.procir.2018.03.258

Rahman, H. F., Sarker, R., & Essam, D. (2018). Multiple-order permutation flow shop scheduling under process interruptions. *International Journal of Advanced Manufacturing Technology*, *97*(5–8), 2781–2808. https://doi.org/10.1007/s00170-018-2146-z

Ramos, A., Sánchez, P., Ferrer, J. M., Barquín, J., & Linares, P. (2010). *Modelos Matemáticos de Optimización*. *Universidad Pontifica de Comillas*.

Rifai, A. P., Nguyen, H. T., & Dawal, S. Z. M. (2016). Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling. *Applied Soft Computing Journal*, *40*, 42–57. https://doi.org/10.1016/j.asoc.2015.11.034

Ruiz, R. (2003). Metaheuristicas para la programación flexible de la producción.

Selman, B., Kautz, H., & Cohen, B. (1994). Noise strategies for local search. *AAAI/IAAI Proceedings*, (1990), 337–343.

Shao, W., Pi, D., & Shao, Z. (2017). Optimization of makespan for the distributed no-wait flow shop scheduling problem with iterated greedy algorithms. *Knowledge-Based Systems*, *137*, 163–181. https://doi.org/10.1016/j.knosys.2017.09.026

Shao, Z., Pi, D., & Shao, W. (2018). A novel discrete water wave optimization algorithm for blocking flow-shop scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation*, *40*(November 2017), 53–75. https://doi.org/10.1016/j.swevo.2017.12.005

Sheikh, S., Komaki, G. M., & Kayvanfar, V. (2018). Multi objective two-stage assembly flow shop with release time. *Computers and Industrial Engineering*, *124*(June), 276–292. https://doi.org/10.1016/j.cie.2018.07.023

Sioud, A., & Gagné, C. (2018). Enhanced migrating birds optimization algorithm for the permutation flow shop problem with sequence dependent setup times. *European Journal of Operational Research*, *264*(1), 66–73. https://doi.org/10.1016/j.ejor.2017.06.027

Takano, M. I., & Nagano, M. S. (2017). A branch-and-bound method to minimize the makespan in a permutation flow shop with blocking and setup times. *Cogent Engineering*, *4*(1), 1–16. https://doi.org/10.1080/23311916.2017.1389638

Valencia, P. E. (2014). Optimización Mediante Algoritmos Genéticos. *Instituto de Ingenieros de Chile*, (May), 83–92. https://doi.org/10.1016/j.annemergmed.2013.01.025

Xu, J., Wu, C. C., Yin, Y., & Lin, W. C. (2017). An iterated local search for the multi-objective permutation flowshop scheduling problem with sequence-dependent setup times. *Applied Soft Computing Journal*, *52*, 39–47. https://doi.org/10.1016/j.asoc.2016.11.031

Ying, K. C., & Lin, S. W. (2018). Minimizing makespan for no-wait flowshop scheduling problems with setup times. *Computers and Industrial Engineering*, *121*(May), 73–81. https://doi.org/10.1016/j.cie.2018.05.030

Zhang, X., Tong, J., & Ma, Y. (2014). An Effective Hybrid Ant Colony Optimization for Permutation Flow-Shop Scheduling. *The Open Automation and Control Systems Journal*, *6*(1), 62–68. https://doi.org/10.2174/1874444301406010062

Zobolas, G. I., Tarantilis, C. D., & Ioannou, G. (2009). Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers and Operations Research*, *36*(4), 1249–1267. https://doi.org/10.1016/j.cor.2008.01.007