METAHEURISTICA GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE (GRASP- PROCEDIMIENTO DE BUSQUEDA CODICIOSA ALEATORIZADA Y ADAPTATIVA) APLICADA AL JOB SHOP SCHEDULING PROBLEM (JSSP)

YESSICA ALEJANDRA APARICIO TORRES

CAROLINA GONZALEZ PATIÑO

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS

ESCUELA DE ESTUDIOS INDUSTRIALES Y EMPRESARIALES

BUCARAMANGA

2016

METAHEURISTICA GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE (GRASP- PROCEDIMIENTO DE BUSQUEDA CODICIOSA ALEATORIZADA Y ADAPTATIVA) APLICADA AL JOB SHOP SCHEDULING PROBLEM (JSSP)

YESSICA ALEJANDRA APARICIO TORRES

CAROLINA GONZALEZ PATIÑO

TRABAJO DE GRADO PRESENTADO COMO REQUISITO PARA OPTAR AL TÍTULO DE INGENIERA INDUSTRIAL

Director:

CARLOS EDUARDO DIAZ BOHORQUEZ

Magister en Ingeniería Industrial

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICOMECÁNICAS

ESCUELA DE ESTUDIOS INDUSTRIALES Y EMPRESARIALES

INGENIERÍA INDUSTRIAL

BUCARAMANGA, SANTANDER

2016

AGRADECIMIENTOS

A Dios por permitirnos alcanzar este logro.

A los integrantes del grupo OPALO por sus valiosos aportes y gratas palabras de motivación.

Al profesor Carlos Eduardo Díaz por su guía y apoyo durante este proceso.

A Reinaldo Pabón por su colaboración y gentileza.

A la Escuela de Estudios Industriales y Empresariales por la formación recibida.

A todos aquellos amigos que con su compañía nos regalaron los mejores recuerdos de esta etapa de nuestra vida.

DEDICATORIA

A mis padres, el motor de mi vida, mi fuerza constante, quienes con amor y paciencia me enseñaron a luchar por mis sueños y no rendirme jamás, es por ellos que hoy pude alcanzar una meta más.

A mis hermanas, por su apoyo e incondicionalidad, quienes con sus consejos me ayudan a escalar cada vez más alto.

A mis sobrinos, la alegría de mis días, que con sus ocurrencias me llenan de felicidad y motivos para seguir.

A Daniel, mi compañero de vida, por brindarme su amor, por ser el complemento de mí ser. Estamos un paso más cerca.

A todos ellos, gracias por existir.

Alejandra Aparicio Torres

DEDICATORIA

A mi mamá, quien me enseñó a dar siempre lo mejor de mí y a quien le debo la persona que soy. Esto es por ella y para ella.

A mi familia, por creer en mí, colaborarme y brindarme su apoyo en todo momento.

A mis amigos, quienes hicieron esta etapa de mi vida más sencilla y divertida.

Carolina González Patiño

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

RESUMEN

TÍTULO: METAHEURISTICA GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE (GRASP- PROCEDIMIENTO DE BUSQUEDA CODICIOSA ALEATORIZADA Y ADAPTATIVA) APLICADA AL JOB SHOP SCHEDULING PROBLEM (JSSP)[*]

AUTORES: APARICIO TORRES, Yessica Alejandra. GONZALEZ PATIÑO, Carolina.[**]

PALABRAS CLAVE: Procedimiento de Búsqueda Codiciosa Aleatorizada y Adaptativa, GRASP, Algoritmo Genético, AG, Job Shop Scheduling Problem, JSSP, Hibrído, metahurística, makespan.

DESCRIPCIÓN

La presente investigación tiene como propósito diseñar un algoritmo para dar solución al *Job Shop Scheduling Problem* tradicional bajo el objetivo de minimización del *makespan* teniendo como punto de partida la metaheurística GRASP, para ello se realiza una revisión de literatura que incluye el problema, el método de solución y la combinación de estos dos como base para la investigación. Seguido a esto, se diseña el algoritmo teniendo como resultado un hibrido compuesto por GRASP y Algoritmo Genético, el cual inicia con la construcción de la población inicial por medio de GRASP y los operadores genéticos utilizados son selección aleatoria, un operador de cruce propuesto por Park et. al. y mutación swap.

El algoritmo diseñado se desarrolla en lenguaje de programación “M” propio de matlab y es validado a través de instancias estudiadas en la literatura variando algunos de los parámetros seleccionados. Adicionalmente, se realiza un diseño factorial 2^k para cada una de las instancias probadas con el fin de identificar los factores significativos en la variable respuesta *makespan*. Finalmente, se presentan los resultados obtenidos y se comparan con otros algoritmos propuestos.

[*] Trabajo de grado

[**] Facultad de Ingenierías Físicomecánicas. Escuela de Estudios Industriales y Empresariales. Director: MSc. Carlos Eduardo Díaz Bohórquez.

ABSTRACT

TITLE: METAHEURISTIC GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE (GRASP) APPLIED TO THE JOB SHOP SCHEDULING PROBLEM (JSSP)*

AUTORS: APARICIO TORRES, Yessica Alejandra. GONZALEZ PATIÑO, Carolina.**

KEYWORDS: Greedy Randomized Adaptive Search Procedure, GRASP, Genetic Algorithm, AG, Job Shop Scheduling Problem, JSSP, Hybrid, metahurístics, makespan.

DESCRIPTION

This research aims to design an algorithm to solve the Job Shop Scheduling Problem in a basic scheme and under the single objective of minimizing the makespan, taking as its starting point the GRASP metaheuristic. First of all a literature review is done, this review includes the problem, the solution method and the combination of these as the basis for the research. Following this, the algorithm resulting is a hybrid between GRASP and Genetic Algorithm which begins with the creation of an initial population using the construction phase of GRASP and then the genetic operators are used to improve this population, these operators are: Random selection, a crossover operator proposed by Park et al. and swap mutation.

The algorithm is developed in the programming language "M" own by Matlab, then it is validated through the instances of benchmark problems varying some selected parameters. Additionally, a factorial design 2^k is performed for each tested instance in order to identify the significant factors in the variable response makespan. Finally, the results obtained are presented and compared with other proposed algorithms.

* Trabajo de grado

** Facultad de Ingenierías Físicomecánicas. Escuela de Estudios Industriales y Empresariales. Director: MSc. Carlos Eduardo Díaz Bohórquez.

INTRODUCCIÓN

La programación de operaciones (*scheduling*) es una tarea que representa un reto en las organizaciones debido a que apoya el proceso de toma decisiones de nivel operativo, por lo cual, obtener buenos resultados se verá reflejado en los objetivos organizacionales como la satisfacción del cliente y el uso eficiente de los recursos disponibles, lo que conlleva a un desempeño competitivo en el medio en que se desarrolle la actividad económica. Algunos ejemplos de problemas de *scheduling* pueden ser, operaciones en un proceso de producción, despegues y aterrizajes en un aeropuerto, etapas de un proyecto de construcción, ejecución en un programa de computador, entre otros. La importancia que se le ha dado a lo largo del tiempo a encontrar soluciones que cumplan con objetivos establecidos y que sea viable en cuanto al tiempo de obtención de la misma, ha dirigido un gran número de investigaciones cuya base son los algoritmos de aproximación.

Entre los problemas de programación se puede destacar el *Job Shop Scheduling*, el cual, a pesar del tiempo que ha pasado desde que se consideró por primera vez, actualmente continúa siendo objeto de estudio dado su nivel de complejidad y su aplicabilidad a las industrias manufactureras y de servicios; este problema consiste en la programación de n trabajos en m máquinas, en donde cada trabajo se caracteriza por tener una secuencia fija de operaciones que se procesan en máquinas especificas con tiempos de procesamiento determinados bajo una serie de restricciones, cuyo fin es cumplir uno o varios objetivos de minimización.

Los métodos de solución existentes para afrontar este problema han evolucionado desde métodos exactos hasta heurísticas y metaheurísticas, dando más importancia a llegar a un resultado de buena calidad en un tiempo razonable. Por otro lado, modelar la realidad se ha convertido en tendencia en los estudios más recientes, es por esto que se han incluido variantes en el *Job Shop* que pretenden acercarlo a dicha realidad. Sin embargo, la necesidad de

entender un problema y las vías que existen para encontrar su solución deben abordarse desde un esquema básico y así establecer un marco de referencia para las investigaciones futuras en donde pueden incluirse dichas variaciones u otros aspectos.

En esta investigación se desarrolla un algoritmo basado en la metaheurística GRASP y Algoritmo Genético para dar solución al JSSP clásico bajo el único objetivo de minimización del *makespan,* validándolo en instancias que han sido ampliamente estudiadas en la literatura, estos resultados son comparados para comprobar la efectividad del algoritmo.

1. PLANTEAMIENTO DEL PROBLEMA

En un sistema productivo tipo *Job Shop* se deben procesar un conjunto finito de n trabajos en un conjunto finito de m máquinas, en donde cada trabajo se caracteriza por tener una secuencia fija de operaciones que se procesan en máquinas específicas con tiempos de procesamiento determinados. Cada trabajo puede ser procesado solo por una máquina a la vez y cada máquina puede procesar solo un trabajo al tiempo, una vez inicia el procesamiento no puede ser interrumpido hasta completarse[1]. El problema de *scheduling* para este sistema consiste en asignar las máquinas a las tareas, establecer el orden en que se van a efectuar y determinar el instante de inicio y finalización de cada una, con el fin de aprovechar los recursos eficientemente y así alcanzar los objetivos establecidos según los criterios de optimización.

El *Job Shop Scheduling Problem* está clasificado como un problema de optimización combinatoria tipo *NP-Hard,* por lo tanto encontrar soluciones óptimas a través de métodos exactos requiere tiempos de computación que crecen de forma exponencial, siendo poco aplicables en situaciones reales. Es por esto que ha sido ampliamente estudiado en la literatura, en donde se han propuesto diferentes métodos de aproximación para obtener soluciones eficientes.

Las metaheurísticas se caracterizan por su efectividad en la solución de problemas que se consideraban intratables, como el caso del JSSP. Sin embargo, esta es una línea de investigación con oportunidades de mejora en la generación de soluciones, por lo tanto, actualmente se continúa el estudio de este problema utilizando estrategias de intensificación a las metaheurísticas existentes y creación de híbridos.

En esta investigación se busca abordar el problema del JSSP con el objetivo de minimizar el *makespan*, por medio de un hibrido teniendo como base GRASP,

[1] BŁAŻEWICZ, Jacek; DOMSCHKE, Wolfgang; PESCH, Erwin. The job shop scheduling problem: Conventional and new solution techniques. European journal of operational research, 1996, vol. 93, no 1, p. 1-33.

validando y comparando el algoritmo con algunas instancias tratadas en la literatura.

2. JUSTIFICACIÓN

La programación de operaciones (*Scheduling*) representa un factor importante en el cumplimiento de las demandas de los clientes y en la generación de beneficios en las diferentes industrias. El *Job Shop* se encuentra clasificado como un problema de Scheduling de tipo *NP-hard* dada la dificultad que existe para obtener solución por métodos convencionales de optimización. Este problema se considera de gran importancia por su aplicabilidad en los casos que se presentan en la realidad en empresas manufactureras y de servicios; es por ello que su programación se ha convertido en un tema destacado, motivando a investigadores a utilizar diferentes métodos para desarrollar algoritmos que busquen mejores soluciones en los diferentes objetivos del problema.

Entre los métodos que se han utilizado se encuentran metaheurísticas como algoritmo genético, búsqueda tabú, enjambre de partículas, optimización por colonia de hormigas, algoritmo memético, entre otros. Hasta la fecha, en la Escuela de Estudios Industriales y Empresariales se han realizado trabajos de grado utilizando algunas de estas metaheurísticas enfocadas al Job Shop, sin embargo, se identifica la oportunidad de estudiar otros métodos de solución entre los cuales se encuentra *Greedy Randomized Adaptive Search Procedure,* Procedimiento de búsqueda codiciosa aleatorizada y adaptativa, (GRASP).

GRASP es una metaheurística *multi-start* para problemas combinatorios que consiste en un proceso iterativo donde cada iteración consta de dos fases: la fase de construcción y la fase de búsqueda local. Esta ha sido aplicada exitosamente en problemas clásicos como la programación de producción, ruteo, problema de partición, ubicación y diseño de sistemas productivos, problemas de asignación cuadrática y otros problemas de asignación de optimización manufacturera.

En la literatura se encuentran aplicaciones de GRASP para el problema de *Job Shop Scheduling* que llegan a soluciones efectivas en comparación a otros métodos en diferentes instancias. Sin embargo, en la revisión literaria se evidenció que existen pocas aplicaciones en este tipo de problema. Es por esto

que el presente proyecto de grado busca combinar y aplicar este algoritmo para la minimización del *makespan* en el Job Shop clásico, ya que se considera importante como ejercicio académico y como un punto de partida para nuevas investigaciones.

3. OBJETIVOS

3.1. OBJETIVO GENERAL

Resolver el problema de *Job Shop Scheduling* para la minimización del *makespan* utilizando la metaheurística GRASP.

3.2. OBJETIVOS ESPECÍFICOS

- Revisar la literatura relacionada al *Job Shop Scheduling problem* y al método de solución propuesto.
- Combinar la metaheurística GRASP con heurísticas de mejoramiento para la solución del JSP.
- Programar el algoritmo en Matlab.
- Validar el algoritmo en diferentes instancias estudiadas en la literatura.
- Realizar un artículo de carácter publicable con los resultados, análisis y conclusiones obtenidos.

4. MARCO TEÓRICO

4.1. OPTIMIZACIÓN COMBINATORIA

Para CUNQUERO[2] la optimización combinatoria es una rama de la matemática aplicada que busca a través de la realización de ciertas operaciones, encontrar un objetivo óptimo de un conjunto finito de soluciones factibles, este objetivo por lo general se basa en criterios de minimización o maximización de una función objetivo. Dicho conjunto de soluciones tiene un número elevado de elementos que hacen muy difícil el poder explorarlos o evaluarlos todos, adicionalmente cabe resaltar que dada la naturaleza finita del mismo las variables allí tratadas son discretas.

Según BLUM[3] en esta clase de problema se tienen los siguientes elementos:

- Un conjunto de variables $X=\{x_1, x_2, x_3, \ldots, x_n\}$;
- Los dominios de las variables $D_1, D_2, D_3, \ldots, D_n$;
- Las restricciones entre variables;
- Una función objetivo *f* a ser minimizada o maximizada;

El conjunto de todas las soluciones factibles S, también llamado espacio de búsqueda o espacio de solución, en donde la combinación o permutación de parámetros que genera un elemento s del conjunto satisface todas las restricciones y es considerado un candidato para la solución se denota de la siguiente manera:

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

[2] CUNQUERO, Rafael Martí. Algoritmos Heurísticos en Optimización Combinatoria.

[3] BLUM, Christian; ROLI, Andrea. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. En: *ACM Computing Surveys (CSUR)*, 2003, vol. 35, no 3, p. 268-308.

La solución de un problema combinatorio se da cuando se encuentra una solución s* ∈ S, la cual, según el criterio de minimización o maximización, sea $f(s^*) \leq f(s) \; \forall \; s \in S$ o $f(s^*) \geq f(s) \; \forall \; s \in S$ respectivamente.

En los problemas de optimización combinatoria como se mencionó anteriormente, la solución se encuentra codificada con variables discretas, sin embargo, esto no significa que todos aquellos que tengan esta característica sean problemas de optimización combinatoria. Algunos de los problemas CO (*Combinatorial Optimization*) son: el problema del agente de ventas viajero (TSP), el problema de asignación cuadrática (QAP), horarios y programación o *scheduling;* dentro de esta última clasificación se encuentra el problema de *Job Shop*. Para la solución de estos problemas se han estudiado algoritmos exactos y de aproximación los cuales se utilizan según la complejidad de los problemas y su tamaño.

4.2. CLASIFICACIÓN DE LOS SCHEDULING PROBLEMS

Los *scheduling problems* pueden ser clasificados por varias características, una de ellas es según la forma en la cual los trabajos llegan al taller, clasificándose como estáticos, cuando cierto número de trabajos llega simultáneamente al taller que está inactivo y disponible inmediatamente para trabajar; y dinámicos, cuando llegan intermitentemente, y sus llegadas continuarán indefinidamente en el futuro[4]. Por otra parte, también existe una clasificación según la relación entre los trabajos y las operaciones, en la cual los sistemas de producción se clasifican en una sola estación, donde cada trabajo se compone de una sola operación y multi-estación, de más de una; cada operación requiere de una maquina diferente. Dentro de los sistemas multi-estación los principales tipos de *shop scheduling problems* que han sido ampliamente estudiados en la literatura son: *the flow shop, the job shop* y *the open shop scheduling problems*[5].

[4] CONWAY, R. W.; MAXWELL, W. L.; MILLER, L. W. Theory of scheduling. 1967. *Theory of Scheduling, Palo Alto-London*, 1967.
[5] LIU, Shi Qiang; ONG, Hoon Liong; NG, Kien Ming. A fast tabu search algorithm for the group shop scheduling problem. *Advances in Engineering Software*, 2005, vol. 36, no 8, p. 533-539.

4.2.1. General Shop Problem[6]*:* Se tienen n trabajos $i = 1, \dots, n$ y m máquinas $M_1, \dots, M_m$. Cada trabajo i se compone de un conjunto de operaciones O_{ij} $(j = 1, \dots, n_i)$, con tiempo de procesamiento p_{ij}. Cada Operación debe ser procesada en una maquina $\mu_{1ij} \in \{M_1, \dots, M_m\}$. En este problema es posible la existencia de relaciones de precedencia entre las operaciones de todos los trabajos. Además, cada trabajo puede ser procesado solo por una maquina al mismo tiempo y cada máquina solo puede procesar un trabajo a la vez.

4.2.2. Open Shop Problems: Es un caso especial del General Shop, en el cual cada trabajo i se compone de m operaciones O_{ij} $(j = 1, \dots, m)$, donde O_{ij} debe ser procesado en una máquina M_j y no hay relaciones de precedencia entre las operaciones. El problema es encontrar el orden de las operaciones pertenecientes al mismo trabajo y el orden de operaciones a ser procesados en la misma máquina.

4.2.3. Flow Shop Problems: Es un General Shop Problem en el cual cada trabajo i consiste de m operaciones O_{ij} con tiempo de procesamiento p_{ij} $(j = 1, \dots, m)$, donde O_{ij} debe ser procesado en la máquina M_j y hay restricciones de precedencia de la forma $O_{ij} \rightarrow O_{i,j+1}$ $(i = 1, \dots, m-1)$ para cada $i = 1, \dots, n$, es decir, cada trabajo es procesado primero en la maquina uno, luego en la dos, luego en la tres, y así sucesivamente. El problema es encontrar un orden de trabajo para cada máquina j.

[6] BRUCKER, Peter; BRUCKER, P. *Scheduling algorithms*. Berlin: Springer, 2007.

4.2.4. Job Shop Problems: Este también es un caso especial del General Shop Problem en el cual se generaliza el flow shop problem. Se tienen n trabajos $i = 1, \dots, n$ y m máquinas $M_1, \dots, M_m$. El trabajo i se compone de una secuencia de n_i operaciones $O_{i1,}\, O_{i2}, \dots, O_{in_i}$; las cuales deben ser procesadas en ese orden, es decir, existen restricciones de precedencia de la forma $O_{ij} \rightarrow O_{i,j+1}$ $(i = 1, \dots, n_i - 1)$. Hay una máquina μ_{ij} ϵ $\{M_1, \dots, M_m\}$ y un tiempo de procesamiento p_{ij} asociado con cada operación O_{ij}. O_{ij} debe ser procesada por las unidades de tiempo p_{ij} en la máquina μ_{ij}. El problema es encontrar un *schedule* que optimice el objetivo de desempeño establecido.

- ✓ **Schedule[7]:** Un *schedule* es la asignación de máquinas y posiblemente recursos a los trabajos en un tiempo determinado, para el caso del *Job shop Shceduling Problem* clásico corresponde solo a la asignación de máquinas.

Tipos de Schedule:

- *Schedule* factible: Cuando el *schedule* satisface todas las condiciones del problema.
- *Schedule* semiactivo: Se dice que un *schedule* factible es semiactivo si los trabajos en el Schedule no pueden ser desplazados para comenzar antes sin cambiar la secuencia de los trabajos.
- *Schedule* activo: Se dice que un *schedule* semiactivo es activo si los trabajos en el *schedule* no pueden ser desplazados para comenzar antes sin retrasar otros trabajos.
- *Non-delay* (Sin retraso): Son *schedules* activos, en los cuales los trabajos se ponen en el programa del tal manera que el tiempo de inactividad de

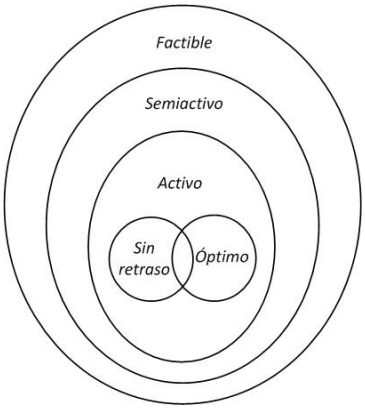
[7] GAWIEJNOWICZ, Stanislaw. Basics of the scheduling theory. En *Time-Dependent Scheduling*. Springer Berlin Heidelberg, 2008. p. 35-46.

la maquina es minimizado y que ninguna maquina se mantenga inactiva si alguna operación puede ser procesada.

- *Schedule* óptimo: Se dice que es óptimo si el valor del criterio de optimización para el *schedule* es óptimo.

En la figura 1 se ilustra las clases de *schedule* y la relación que hay entre ellas, observando que se garantiza que el *schedule* óptimo es un *Schedule* activo aunque no necesariamente *non-delay.*

Figura 1. Tipos de *Schedule* y su relación.



Fuente: Adaptado de *An Effective PSO and AIS-Based Hybrid Intelligent Algorithm for Job-Shop Scheduling*[8].

✓ **Criterios de optimización:** También conocidos como medidas de desempeño, considerando $C_1, C_2, \dots, C_n$ el tiempo de terminación en un schedule. Algunos de los criterios utilizados son:

- Tiempo máximo de terminación $C_{max} = \max\{C_j\}$, $donde\ j\ denota\ trabajo$.
- Tardanza máxima

$$L_{max} = \max\{L_j\} = \max\{C_j - d_j\};\ d_j: fecha\ de\ entrega\ del\ trabajo\ j.$$

[8] GE, Hong-Wei, et al. An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling. En: Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 2008, vol. 38, no 2, p. 358-368.

- Retraso máximo $T_{max} = \max\{T_j\} = \max\{\max\{0, L_j\}\}$
- Costo máximo $f_{max} = \max\{f_j(C_j)\}$,
  $donde\ f_1, f_2, \dots, f_n\ son\ funciones\ de\ costo\ dadas.$
- Tiempo total de terminación $\sum C_j = \sum_{j=1}^{n} C_j$
- Tiempo total de terminación ponderado
  $\sum w_j C_j = \sum_{j=1}^{n} w_j C_j$; w_j: $importancia\ relativa\ del\ trabajo\ j.$
- Carga total de la máquina $\sum C_{max}^{(k)} = \sum_{j=1}^{m} C_{max}^{(k)}$,
  $donde\ C_{max}^{(k)}\ denota\ el\ tiempo\ de\ terminación\ máxima\ sobre$
  $todos\ los\ trabajos\ asignados\ a\ la\ máquina\ M_k, 1 \leq k \leq m.$
- Número de trabajos tardíos $\sum U_j = \sum_{j=1}^{n} U_j$, donde $U_j = 0\ si\ L_j \leq 0\ y\ U_j = 1\ si\ L_j > 0$
- Costo total $\sum f_j = \sum_{j=1}^{n} f_j(C_j)$,
  $donde\ f_1, f_2, \dots, f_n\ son\ las\ funicones\ de\ costos\ dadas.$

El makespan C_{max} ha sido el principal criterio para investigadores académicos. Este es equivalente al tiempo de terminación del último trabajo para dejar el sistema, se considera que un mínimo makespan usualmente implica una buena utilización de las máquinas[9].

[9] PINEDO, Michael L. *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media, 2012.

- ✓ **Formulación Job Shop Scheduling Problem:** El modelo matemático para este problema se describe como en BINATO et al.10 Se tiene un conjunto de trabajos $\mathcal{J}$ y un conjunto de máquinas $\mathcal{M}$, donde $\sigma_1^j \prec \sigma_2^j \prec \sigma_{|\mathcal{M}|}^j$ es el conjunto ordenado de $|\mathcal{M}|$ operaciones del trabajo $\mathcal{J}$ y $\boldsymbol{\mathcal{O}}$ el conjunto de operaciones, cada operación σ_k^j está definida por dos parámetros: $\mathcal{M}_k^j$ es la máquina en la cual σ_k^j es procesada y $p(\sigma_k^j)$ es el tiempo de procesamiento de la operación σ_k^j; y el tiempo de inicio de la k-esima operación está dado por $t\,(\sigma_k^j)$.

$$Minimizar\ Cmax$$

$$Cmax\ \geq\ t\,(\sigma_k^j) + p(\sigma_k^j),\ \ para\ todo\ \sigma_k^j \in\ \boldsymbol{\mathcal{O}},$$

Una operación del mismo trabajo inicia cuando se ha finalizado la operación precedente a esta

$$t\,(\sigma_k^j)\ \geq\ t\,(\sigma_l^j) + p(\sigma_l^j),\ \ para\ todo\ \sigma_l^j\ \prec\ \sigma_k^j,$$

Las operaciones de diferentes trabajos que se realizan en la misma máquina no se pueden hacer al mismo tiempo

$$t\,(\sigma_k^j)\ \geq\ t\,(\sigma_l^i) + p(\sigma_l^i)\ \ \vee\ \ t\,(\sigma_l^j)\ \geq\ t\,(\sigma_k^j) + p(\sigma_k^j),$$
$$para\ todo\ \sigma_l^i,\ \ \sigma_k^j\ tal\ que\ M_{\sigma_l^i} =\ M_{\sigma_k^j}$$

El tiempo de inicio de cada operación pertenece a los reales positivos

$$t\,(\sigma_k^j)\ \geq\ 0, para\ todo\ \sigma_k^j\ \in\ \boldsymbol{\mathcal{O}}$$

[10] BINATO, S., et al. A GRASP for job shop scheduling. En: Essays and surveys in metaheuristics. Springer US, 2002. p. 59-79.

- ✓ **Representación de la solución:** Dentro de los métodos de representación más comunes para la solución del JSP se encuentran el diagrama de Gantt descrito en Gantt (1919), Clark (1922) y Porter (1968) y el grafo disyuntivo propuesto por Roy y Sussman (1964).

  - Diagrama de Gantt: Se caracteriza por su facilidad de interpretación y consiste básicamente en un diagrama de barras, donde el eje horizontal corresponde al tiempo y el eje vertical a los recursos disponibles, en este caso, máquinas. Las barras representan el tiempo de inicio y final de una actividad.

  - Grafo disyuntivo: Considerado un buen método para representar el problema de minimización del makespan[11]. Se define:

$$G = (N, A, E)$$

Siendo N el conjunto de nodos que representa las operaciones

$$\{O, O_{11}, O_{12}, \dots n \times m + 1\}$$

Donde O denota el inicio ficticio y $n \times m + 1$ final ficticio.

A es el conjunto de arcos ordinarios (conjuntivos) que conectan operaciones consecutivas del mismo trabajo.

E es el conjunto de arcos disyuntivos que conectan operaciones procesadas por la misma máquina.

El largo de un arco denota el tiempo de procesamiento procesado en la maquina correspondiente.

Se presenta un ejemplo de Job Shop con tres trabajos y tres máquinas para observar los métodos de representación, en la figura 2 se puede observar el diagrama de Gantt y en la figura 3 el grafo disyuntivo.

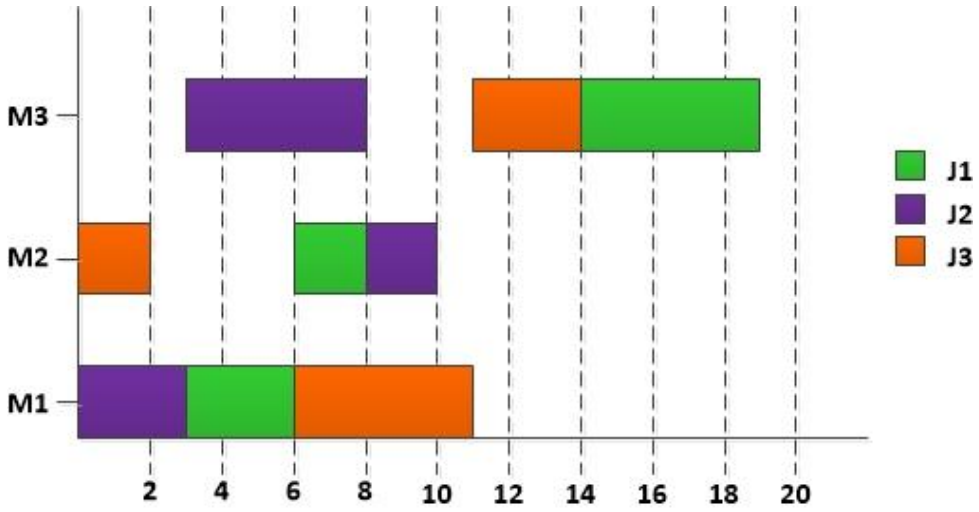
[11] HERRMANN, Jeffrey W. A history of production scheduling. En *Handbook of Production Scheduling*. Springer US, 2006. p. 1-22.

Tabla 1. Ejemplo de un Job Shop con tres trabajos y tres máquinas.

Trabajos	***Secuencia de máquinas***	***Tiempo de procesamiento***
J1	1-2-3	3-2-5
J2	1-3-2	3-5-2
J3	2-1-3	2-5-3

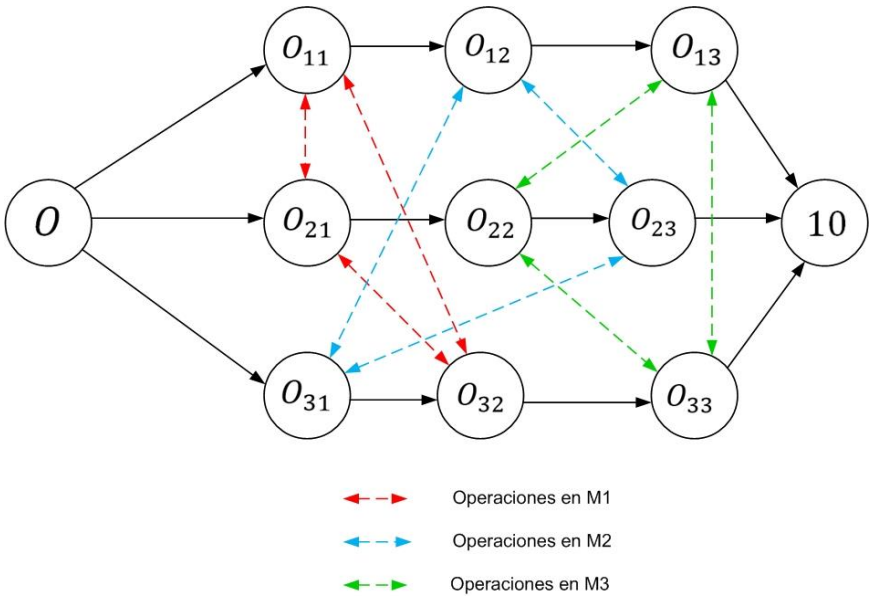
Fuente: Tomado de *An efficient memetic algorithm for solving the job shop scheduling problem*[12]

Figura 2. Diagrama de Gantt para un Job Shop con tres trabajos y tres máquinas.



[12] GAO, Liang, et al. An efficient memetic algorithm for solving the job shop scheduling problem. Computers & Industrial Engineering, 2011, vol. 60, no 4, p. 699-705.

Figura 3. Grafo Disyuntivo para un Job Shop con tres trabajos y tres maquinas



Fuente: Adaptado de *An efficient memetic algorithm for solving the job shop scheduling problem*[13].

4.3. COMPLEJIDAD COMPUTACIONAL

Para PAPADIMITRIOU (2003)[14] el tener que definir la complejidad computacional nació dado el empeño de muchos investigadores por encontrar algoritmos que mejoraran cada vez más cierto problema, encontrando tras varios intentos que esto parecía imposible. Sin embargo, esta definición no podía darse de manera arbitraria, lo cual llevó al desarrollo de técnicas matemáticas que formalmente probaran que no había algún algoritmo que tuviera mejor desempeño en algunos problemas que los ya conocidos; es ahí donde se establece que el dominio de los modelos matemáticos y las técnicas utilizadas para establecer pruebas de tal imposibilidad se llama complejidad computacional.

[13] Ibid.
[14] PAPADIMITRIOU, Christos H. *Computational complexity*. John Wiley and Sons Ltd., 2003.

4.3.1. Clases de complejidad computacional: Las clases de complejidad, por uniformidad y facilidad, no están compuestas por problemas que tengan alguna característica en especial, se forman por un lenguaje el cual es una secuencia de caracteres que puede ser la representación de un problema dado si logra captar su complejidad.

- **Complejidad P:** Es la clase de complejidad más importante y comprende todos los problemas que pueden ser resueltos en un algoritmo de tiempo polinomial.
- **Complejidad NP:** Es una clase más amplia que la tipo P y se caracteriza porque cualquier lenguaje en esta clase puede ser determinado por una máquina de *Turing* no determinística polinomial; una máquina de *Turing* o *Turing machine* es un artículo hipotético que tiene la capacidad de simular la lógica de un algoritmo, en otras palabras, según SIPSER[15], es un modelo matemático definido como máquina.
- **Complejidad NP-completo:** Continuando con lo dicho por PAPADIMITRIOU[16], la clase NP-completo representa el conjunto de todos los problemas z en NP para los cuales es posible reducir cualquier otro problema NP y a z a los mismos en tiempo polinomial.
- **Complejidad NP-Hard (complejo):** Para JAIN y MEERAN[17] Son los problemas que están fuera del orden polinomial, es decir que su requerimiento de tiempo es de orden $O(y^x)$ donde *x* es el tamaño de las entradas y *y* es una constante.

El problema de *Job Shop Scheduling* está dentro de la categoría de *NP-Hard* y algunos de sus casos especiales se pueden considerar como *Strongly* NP-Hard

[15] SIPSER, Michael. *Introduction to the Theory of Computation*. Cengage Learning, 2012.

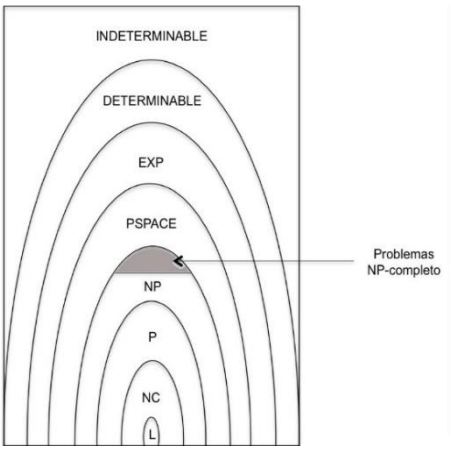
[16] PAPADIMITRIOU. Op. cit.

[17] JAIN, Anant Singh; MEERAN, Sheik. Deterministic job-shop scheduling: Past, present and future. En: European journal of operational research, 1999, vol. 113, no 2, p. 390-434.

o por su traducción literal Fuertemente NP-Hard, lo cual hace que este sea uno de los problemas más duros de esta clase.[18]

En la figura 4 se puede observar las diferentes clases de complejidad representadas como regiones; aquí se muestra un planteamiento en donde las clasificaciones de problemas pueden contener en sí otras clasificaciones, por ejemplo, los problemas tipo P están contenidos en EXP y así mismo EXP está contenido entre los problemas clasificados como determinables.

Figura 4. Clases de complejidad representadas como regiones.



Fuente. Adaptado de *Computational complexity.*[19]

[18] NAKANO, Ryohei; YAMADA, Takeshi. Conventional Genetic Algorithm for Job Shop Problems. En: *Proceedings of the 4th International Conference on Genetic Algorithms, San Diego, CA, USA, July 1991*. 1991. p. 474-479. Citado por: JAIN, Anant y MEERAN, Sheik. Op. cit.

[19] PAPADIMITRIOU. Op. cit.

4.4. METODOS DE SOLUCIÓN PARA EL JSP

4.4.1. Métodos exactos: Branch and Bound es la principal estrategia enumerativa, siendo durante muchos años la técnica más popular para la solución del JSP. Consiste en la construcción dinámica de un árbol que representa todos los schedules factibles, por medio de reglas y procedimientos de esta técnica se eliminan iterativamente largas porciones del árbol hasta encontrar una solución óptima. Este método no puede resolver instancias con más de 250 operaciones en tiempos razonables[20].

4.4.2. Métodos aproximados: Los métodos aproximados surgen como alternativa a las limitaciones de las técnicas exactas, proporcionando buenas soluciones en tiempos racionales.

- ✓ **Heurísticas:** Según Barr et al.[21] "un método heurístico es un conjunto bien conocido de pasos para identificar rápidamente una solución de alta calidad para un problema dado". La principal limitación de las heurísticas se encuentra en la posibilidad de quedar atrapados en un óptimo local, debido a que no cuentan con mecanismos para continuar buscando el óptimo[22]. Los algoritmos heurísticos se clasifican en:

- Métodos constructivos: Son procedimientos que son capaces de construir una solución paso a paso. Esta construcción depende de la estrategia que se sigue, entre las cuales se encuentra: estrategia voraz, estrategia de descomposición, métodos de reducción y métodos de manipulación del modelo.

[20] JAIN; MEERAN. Deterministic job-shop scheduling: Past, present and future. Op. cit.
[21] BARR, Richard S., et al. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1995, vol. 1, no 1, p. 9-32. Citado por: DUARTE MUÑOZ, Abraham; PANTRIGO FERNÁNDEZ, J. J.; GALLEGO CARRILLO, M. Metaheurísticas. *Madrid: Dykinson*, 2007.
[22] DUARTE MUÑOZ, Abraham; PANTRIGO FERNÁNDEZ, J. J.; GALLEGO CARRILLO, M. Metaheurísticas. *Madrid: Dykinson*, 2007.

- Métodos de búsqueda: Estos métodos a diferencia de los de construcción parten de una solución factible dada e intentan mejorarla. Las estrategias que utilizan son, estrategia de búsqueda local 1, estrategia de búsqueda local 2 y estrategia aleatorizada.

Las Reglas de despacho o de prioridad hacen parte de las heurísticas más aplicadas para dar solución al JSP debido a su facilidad de implementación y sus requerimientos computaciones reducidos. Sin embargo, se considera que la calidad de solución es pobre en calidad y más cuando aumenta el tamaño del problema, a causa de que solo consideran el estado actual de la máquina y sus alrededor inmediato.[23] Consisten en asignar una prioridad a las operaciones disponibles para ser programadas en cada paso y se selecciona la operación con la prioridad más alta para ser secuenciada[24].

Se puede encontrar en las más comunes:

- Shortest Task Time (STT): Programa la tarea con el tiempo de procesamiento más pequeño.
- Longest Task Time (LTT): Programa la tarea con el tiempo de procesamiento más largo.
- Shortest Processing Time (SPT): Programa la tarea con el tiempo de procesamiento total más pequeño.
- Longest Processing Time (LPT): Programa la tarea con el tiempo de procesamiento total más largo.
- Shortest Remaining Processing Time (SRPT): Programa la tarea con el tiempo de procesamiento del trabajo pendiente más corto
- Longest Remaining Processing Time (LRPT): Programa la tarea con el tiempo de procesamiento total pendiente más largo.

[23] XIA, Wei-jun; WU, Zhi-ming. A hybrid particle swarm optimization approach for the job-shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 2006, vol. 29, no 3-4, p. 360-366.

[24] JAIN, Anant Singh; MEERAN, Sheik. *A state-of-the-art review of job-shop scheduling techniques*. Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998.

- Last Buffer First Serve (LBFS): Programa la tarea con el número más corto pendiente de tareas posteriores
- First Buffer First Serve (FBFS): Programa la tarea con el número más largo pendiente de tareas posteriores.

El *shifting bottleneck procedure* SBP es otra de las heurísticas destacadas para el JSP, se considera que su principal contribución es la relajación a problemas de una máquina para decidir el orden en que se deben programar los trabajos. Este procedimiento se caracteriza por las siguientes etapas, identificación del subproblema, seguido de la selección del cuello de botella, solución del subproblema y por ultimo reoptimización del *schedule*. Sin embargo, el problema del SBP es la dificultad en el desempeño de la reoptimización y la generación de soluciones infactibles.

- ✓ **Metaheurísticas:** Según Abraham Duarte[25] una metaheurística es un procedimiento que pretende ir más allá de lo que otros lo hacen, modificando y guiando otras heurísticas para librarse de la optimalidad local. Según lo definió J.P Kelly et al.[26] "Las metaheurísticas están diseñadas para resolver problemas difíciles de optimización combinatoria en los que los heurísticos clásicos no son efectivos." En general lo más importante es que estas permiten crear combinaciones o algoritmos híbridos dado que proveen un framework general del cual se puede partir. Las metaheurísticas toman ideas de diferentes campos de investigación como lo son:
  - Técnicas de diseño de algoritmos.
  - Algoritmos específicos.
  - Fuentes de inspiración (del mundo real).
  - Métodos estadísticos.

[25] DUARTE MUÑOZ, Abraham. Op. cit.

[26] KELIY, James P. META-HEURISTICS: Theory & Applications. 1996. Citado por: Ibid.

Básicamente para muchas metaheurísticas no hay un marco teórico en el cual se haya encontrado el diseño de las mismas, sin embargo, esto no quiere decir que no tengan un fundamento sólido que las apoye; por el contrario existen muchos resultados experimentales que sustentan su efectividad y justifican su estudio.

Para poder ser aplicables las metaheurísticas deben tener dos partes, las cuales garanticen su uso en varios problemas y a su vez su efectividad en un problema determinado, esto es posible a una estructura sencilla y general utilizada para el primer objetivo y una estructura que dependa de los parámetros o características del problema. Existen algunos tipos de metaheurísticas que encierran de alguna manera en grupos algunos algoritmos conocidos, estas son:

- Metaheurísticas trayectoriales
- Metaheurísticas poblacionales
- Metaheurísticas constructivas

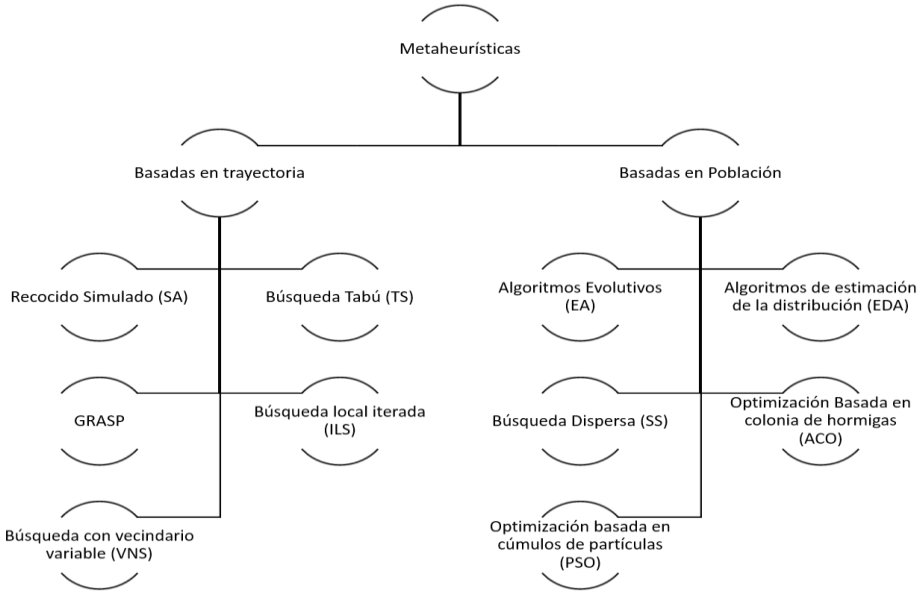
Sin embargo, otros autores resumen de otra manera la clasificación de las metaheurísticas, por ejemplo José García[27] habla de las características seleccionadas de donde se pueden obtener diferentes taxonomías como lo son: basadas y no basadas en la naturaleza, con memoria o sin ella, con una o varias estructuras de vecindario, entre otras. A diferencia de J.P Kelly et al. resume la clasificación en metaheurísticas trayectoriales o basadas en trayectoria y metaheurísticas basadas en poblaciones.

Como se mencionó anteriormente estos métodos de solución permiten la combinación de diferentes esquemas, por lo tanto en los últimos años, para problemas de optimización combinatoria que están clasificados como *NP-Hard* y demás, se ha recurrido a la resolución de estos mediante híbridos entre metaheurísticas llegando a resultados experimentales de mejor calidad.

[27] GARCÍA, José Francisco Chicano. *Metaheurísticas e ingeniería del software*. 2007. Tesis Doctoral. Universidad de Málaga.

En la figura 5 se muestran algunas de las metaheurísticas consideradas como representativas en la clasificación según se basan en la trayectoria o en la población.

Figura 5. Clasificación de las metaheurísticas.



Fuente. Adaptado de Metaheurísticas e ingeniería del software[28].

4.5. GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE (GRASP)

GRASP es un método iterativo, en el cual cada iteración tiene dos fases, una de construcción y una de búsqueda local, en donde la primera se encarga de la construcción de soluciones factibles y la segunda de encontrar un mínimo (o máximo dependiendo del criterio de optimización) a partir de la solución construida. Este procedimiento se repite varias veces hasta que un criterio de parada determine hasta qué punto se continúa iterando y la mejor solución encontrada se asume como resultado.[29]

[28] Ibíd.

[29] RESENDE, Mauricio GC; VELARDE, Jose Luis González. GRASP: Procedimientos de búsquedas miopes aleatorizados y adaptativos. En: Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 2003, vol. 7, no 19, p. 61-76.

En la fase de construcción la solución parcial se obtiene agregando elementos de un conjunto y cada elemento se selecciona a través de una función *greedy* de evaluación. La evaluación que se lleva a cabo permite la construcción de una lista restringida de candidatos o RCL (*restricted candidate list*) la cual contiene los mejores elementos que se hayan encontrado, posteriormente la selección de los mismos desde la RCL se hace de manera aleatoria, luego se incorpora dicho elemento a la solución parcial y el incremento en el costo es reevaluado. Este es el componente codicioso y aleatorio de la meta-heurística.

Después de realizar este procedimiento aún no se garantiza que la solución obtenida sea optima, por lo tanto se recurre a un algoritmo de búsqueda local cuya función es mejorar la solución dada por la fase de construcción reemplazándola por una mejor en el vecindario; finalmente y como se mencionó anteriormente, un criterio de parada se establece, por lo general se refiere a cierto número de iteraciones sin encontrar una mejor solución, para obtener la respuesta final.

Figura 6. Pseudo-código de la metaheurística GRASP

```
procedure GRASP(Max_Iterations,Seed)
1     Read Input();
2     for k = 1,…,Max Iterations do
3          Solution←Greedy Randomized Construction (Seed);
4          Solution←Local Search(Solution);
5          Update Solution(Solution,Best Solution);
6     end;
7     return Best Solution;
end GRASP.
```

Fuente. Adaptado de *Greedy Randomized Adaptive Search Procedures.*[30]

La calidad de la solución depende de muchos aspectos, entre los cuales se encuentran: la estructura de vecindario que se utilice en la fase de búsqueda local, la rápida evaluación de las funciones de costos y la solución principal construida. La implementación de la metaheurística se considera sencilla dado que solo se necesitan dos parámetros para darle inicio al algoritmo; el criterio de

[30] RESENDE, Mauricio GC, et al. Handbook of Metaheuristics. Editado por Glover, F. e Kochenberger, G., Capítulo Greedy Randomized Adaptive Search Procedures, Ed. Kluwer Academic Publishers, pg, 2003, p. 219-249.

parada, como se mencionó anteriormente, y aquello relacionado con la calidad de los elementos de la lista de candidatos restringida. Cabe resaltar que en este procedimiento el tiempo de computación, y por lo tanto la calidad, dependen del número máximo de iteraciones que se establezca.

Uno de los componentes más importantes del GRASP es la ya mencionada RCL (lista de candidatos restringida), pues de aquí se seleccionan los elementos para construir nuevas soluciones, por lo tanto, la construcción de esta debe asegurarse de contemplar aquellos elementos que, sin destruir la factibilidad de la solución, generen los menores incrementos en la función de costo, los cuales, posteriormente se evaluarán. Algunos elementos a tener en cuenta son:

- $c(e)$: Incremento en el costo por agregar el elemento e a la solución bajo construcción.
- E: Conjunto de elementos candidatos a ser incluidos y $e \in E$.
- $c^{min}\ y\ c^{max}$ Mínimo y máximo incremento en el costo respectivamente.
- p: Número máximo de elementos en la RCL (parámetro).
- $\alpha \in [0,1]$: Parámetro de umbral en donde $\alpha = 0$ corresponde a un algoritmo codicioso puro y $\alpha = 1$ corresponde a una construcción aleatoria.

Figura 7. Pseudo-código para el procedimiento de construcción del GRASP.

```
Procedure Greedy_Randomized_Construction(Seed)
1   Solition←∅;
2   Evaluate the incremental costs of the candidate elments;
3   while Solution is not a complete solution do
4       Build the restricted candidate list (RCL);
5       Select an element s from the RCL at random;
6       Solution←Solution ∪ {s};
7       Reevaluate the incremental costos;
8   end;
9   return Solution;
end Greedy Randomized Construction.
```

Fuente:Adaptado de *Greedy Randomized Adaptive Search Procedures*[31]

[31] Ibid., p. 222

La RCL está conformada por los elementos $e \in E$ que tengan los mejores incrementos en el costo $c(e)$, es decir, los que representan el menor cambio. La lista de candidatos restringida puede limitarse de dos formas, una es estableciendo un parámetro p y la otra forma está basada en la calidad de los mismos; aquí los elementos a incluir en la lista a parte de conservar la factibilidad de la solución, también deberán tener mayor calidad en la función de costo que el valor umbral α, esto significa:

$$c(e) \in [c^{min}, c^{min} + \alpha(c^{max} - c^{min})]$$

Figura 8. Pseudo-código para la fase de búsqueda local del GRASP.

```
Procedure Local_Search(Solution)
1    while Solution is not locally optimal do
2        Find s' ∈ N (Solution) with f'(s) < f(Solution);
3        Solution←s';
4    end;
5    return solution;
end Local_Search.
```

Fuente. Adaptado de *Greedy Randomized Adaptive Search Procedures*[32]

Figura 9. Pseudo-código refinado de la fase de construcción del GRASP

```
Procedure Greedy_Randomized_Construction(α,Seed)

1    Solution←∅;
2    Initialize the candidate set: C←E;
3    Evaluate the incremental cost c(e) for all e ∈ C;
4    while C≠ ∅ do
5        c^min←min{c(e)| e ∈ C};
6        c^max←max{c(e)| e ∈ C};
7        RCL← {e ∈ C| c(e) ≤ c^min + α(c^max-c^min)};
8        Select an element s from the RCL at random;
9        Solution←Solution ∪ {s};
10       Update the candidate set C;
11       Reevaluatethe incremental costs c(e) for all e ∈ C;
12   end;
13   return Solution;
end Greedy_Randomized_Construction.
```

Fuente: Adapatado de *Greedy Randomized Adaptive Search Procedures*[33]

[32] Ibíd., p. 223.
[33] Ibid., p. 224.

5. METODOLOGÍA

Para dar cumplimiento a los objetivos planteados, la metodología a seguir en el desarrollo de este proyecto se encuentra estructurada de la siguiente forma:

- Revisión de la literatura: Se realiza la búsqueda en las bases de datos disponibles de la Universidad Industrial de Santander, principalmente en la Web of science, ELSEVIER y SpringerLink, dividiéndose en tres fases, revisión literaria del problema Job Shop Scheduling tradicional, seguido de la revisión literaria de la metaheurística GRASP y por último, revisión de la aplicación del método GRASP en el Job Shop Scheduling Problem, con el fin de conocer y recopilar el estudio de los temas más relevantes para la investigación a realizar.

- Combinación de la metaheurística GRASP: A partir de los resultados de la etapa anterior, se identifican las posibles mejoras, adaptaciones, y/o combinaciones aplicables a la metaheurística GRASP que permita dar solución al problema del Job Shop Scheduling.

- Programación del algoritmo: En esta etapa es necesario, inicialmente, conocer las bases para la programación en el software Matlab y posteriormente realizar el desarrollo del algoritmo.

- Validación del algoritmo: Una vez desarrollo el algoritmo se seleccionan algunas de las instancias más estudiadas en la literatura con el fin de comparar los resultados generados por este.

6. REVISIÓN DE LA LITERATURA

6.1. PROBLEMA: JOB SHOP SCHEDULING PROBLEM

El *Job Shop Scheduling Problem* se considera dentro de los problemas de programación clásicos como el más general, el cual ha sido ampliamente estudiado en la literatura y en donde el principal criterio para investigadores académicos ha sido la minimización del *makespan.* Aunque no hay certeza del origen de este problema, Jain y Meeran (1999)[34] al igual que otros autores lo atribuyen a S.M. Jhonson (1954)[35]. Además, el libro "*Industrial scheduling*" editado por Muth & Thompson (1963)[36] fue aceptado como base para investigaciones futuras, debido a que reúne los principales hallazgos de la época.

Diferentes enfoques de solución han sido estudiados en este problema, en primera instancia se trataron métodos exactos para hallar soluciones óptimas, sin embargo esto tiene un sentido matemático que no siempre es útil en la práctica y adicionalmente, dichas soluciones son difíciles de encontrar debido a que fue demostrado por *Garey, Johnson,* y *Sethi (1976)[37] que el JSSP* está clasificado como *NP-Hard.* Por otro lado, existen métodos de aproximación que arrojan soluciones cercanas al óptimo, los cuales se ajustan mejor a problemas prácticos y son más efectivos para aquellos de gran tamaño puesto que emplean un tiempo de computación moderado. No obstante, resultados de investigaciones han evidenciado que los métodos híbridos, que combinan diferentes enfoques de aproximación, generan mejores soluciones en tiempos computacionales razonables.

[34] JAIN; MEERAN. Deterministic job-shop scheduling: Past, present and future. Op. cit.
[35] JOHNSON, Selmer Martin. Optimal two-and three-stage production schedules with setup times included. En: Naval research logistics quarterly, 1954, vol. 1, no 1, p. 61-68. Citado por: Ibid.
[36] MUTH, John F.; THOMPSON, Gerald Luther (ed.). Industrial scheduling. Prentice-Hall, 1963. Citado por: Ibid.
[37] GAREY, Michael R.; JOHNSON, David S.; SETHI, Ravi. The complexity of flowshop and jobshop scheduling. En: Mathematics of operations research, 1976, vol. 1, no 2, p. 117-129. Citado por: SHA, D. Y.; LIN, Hsing-Hung. A multi-objective PSO for job-shop scheduling problems. En: Expert Systems with Applications, 2010, vol. 37, no 2, p. 1065-1070.

A continuación se encuentran algunas de las investigaciones realizadas para la minimización del *makespan* en el *Job Shop Scheduling Problem.*

BERTSIMAS y SETHURAMAN (2002)[38] Diseñaron un algoritmo llamado *fluid synchronization algorithm* (FSA), para el JSP con el objetivo de minimizar el *makespan*. En el cual se consideró una relajación para el JSP, remplazando los trabajos discretos con un flujo continuo. Los resultados computacionales concluyen que el FSA es un algoritmo práctico para la solución del JSP, debido a las soluciones de alta calidad incluso en los problemas de multiplicidad moderada, que están presentes en los ambientes manufactureros.

AZIZI y ZOLFAGHARI (2004)[39] Muestran un estudio comparativo del algoritmo recocido simulado para la minimización del *makespan* en el JSSP, esta comparación se hace para tres enfoques; el recocido simulado convencional, un recocido simulado adaptativo y un recocido simulado tabú adaptativo. La diferencia entre estos métodos se basa en cómo se maneja la temperatura de enfriamiento dado que ésta afecta el comportamiento del algoritmo y su solución obtenida, en el tercer enfoque después de la modificación que se hace, se agrega una lista Tabú como elemento de memoria que no posee el SA (recocido simulado). Se comparan estas tres propuestas en instancias del *benchmark* dando como resultado un mejor desempeño para el algoritmo adaptativo tabú de recocido simulado *(Adaptive Tabu-Simulated Annealing).*

WENQI y AIHUA (2004)[40] presentan un algoritmo denominado *improved shifting bottleneck procedure* (ISB) y una versión refinada con el cual prueban que es posible conseguir soluciones factibles para cualquier instancia del JSP. La principal diferencia entre el SB e ISB se encuentra en el procedimiento de

[38] BERTSIMAS, Dimitris; SETHURAMAN, Jay. From fluid relaxations to practical algorithms for job shop scheduling: the makespan objective. En: Mathematical Programming, 2002, vol. 92, no 1, p. 61-102.
[39] AZIZI, Nader y ZOLFAGHARI, Saeed. Adaptive temperature control for simulated annealing: a comparative study. En: computers & operations research, 2004. Vol. 31, no. 14, p. 2439-2451.
[40] WENQI, Huang; AIHUA, Yin. An improved shifting bottleneck procedure for the job shop scheduling problem. En: Computers & Operations Research, 2004, vol. 31, no 12, p. 2093-2110.

solución para el problema de una máquina, mientras que el SB lo resuelve con un algoritmo exacto, el ISB lo hace con un algoritmo de aproximación que aplica una perturbación en las reglas de prioridad. Probado en diferentes instancias prueban que con menor tiempo de computación el ISB encuentra mejores soluciones en casi todas las instancias en comparación con el SB, siendo ambos algoritmos efectivos para solucionar el JSP. Además, recomiendan en estudios futuros combinar el ISB con otras estrategias para crear algoritmos metaheurísticos eficientes y efectivos para el JSP.

CHANDRASEKARAN et al.(2006)[41] Proponen un algoritmo de sistema inmune artificial AIS para minimizar el *makespan* en el JSP el cual fue probado en problemas de diferentes tamaño y comparados con el *Tabu search with bottleneck procedure* (TSSB) y *Guided Local Search with Shifting Bottleneck procedure* (SB-GLS1) de Balas y Vazacopoulos[42], siendo una buena técnica para la solución de este tipo de problemas.

XIA y WU (2006)[43] Presentan un algoritmo híbrido basado en la optimización por enjambre de partículas (PSO) combinado con un algoritmo de recocido simulado (SA) para la minimización del *makespan* en el JSSP. Esta idea surgió dada la necesidad de mejorar las soluciones arrojadas por el PSO en su esquema básico, utilizando el SA en la búsqueda global para lograr este objetivo. El HPSO (*Hybrid Particle Swarm Optimization*) mostró ser efectivo, eficiente y de fácil implementación.

LIAN, JIAO y GU (2006)[44] Proponen un algoritmo SPSO *(Similar Particle Swarm Optimization)* en la solución del JSSP para la minimización del *makespan.* Este

[41] CHANDRASEKARAN, M., et al. Solving job shop scheduling problems using artificial immune system. En: The International Journal of Advanced Manufacturing Technology, 2006, vol. 31, no 5-6, p. 580-593.
[42] BALAS, Egon; VAZACOPOULOS, Alkis. Guided local search with shifting bottleneck for job shop scheduling. Management science, 1998, vol. 44, no 2, p. 262-275. Citado por: Ibid.
[43] Xia; Wu. Op. cit.
[44] LIAN, Zhigang; JIAO, Bin. Y GU, Xingsheng. A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. En: applied mathematics and computation, 2006. Vol. 183, no. 2, p. 1008-1017.

algoritmo se ha aplicado antes en la programación de un sistema *Flow Shop* y ha mostrado ser efectivo al integrar algunos operadores de algoritmos genéticos en el sistema. El algoritmo se evalúa en tres instancias representativas de *Taillard*[45] y se compara con la eficiencia del algoritmo genético convencional, obteniendo mejores resultados.

ZHANG, et al. (2007)[46] Presentan un algoritmo de búsqueda Tabú (TS) con una nueva estructura de vecindario para la minimización del *makespan* en el JSSP, esta nueva estructura mejorada está diseñada para obtener nuevos vecindarios de soluciones aplicando pequeñas perturbaciones a la solución inicial, demostrando ser muy eficaz en comparación a las estructuras existentes dado que tiene la capacidad de analizar un espacio de solución más amplio en el marco del algoritmo presentado.

GE et al. (2008)[47] Presentan un algoritmo inteligente híbrido que combina Optimización por enjambre de particulas (PSO) con Sistema Inmune Artificial (AIS) para solucionar el problema de JSSP con el objetivo de minimizar el *makespan*. Arrojando mejores soluciones respecto a otros algoritmos, excepto en el Nowicki and Smutnicki, demostrando que el algoritmo propuesto resuelve el JSP eficientemente.

ZHANG, et al. (2008)[48] Ahora presentan un híbrido entre búsqueda Tabú y Recocido Simulado para la minimización del *makespan* en el JSSP, en donde el algoritmo SA se analiza para afrontar el problema de dependencia en la solución inicial que tiene el algoritmo de TS y a su vez encuentra soluciones elite en el *Big Valley* dando la oportunidad de re-intensificar la búsqueda Tabú en soluciones prometedoras. El recocido simulado en su esquema básico carece de

[45] TAILLARD, Eric. Benchmarks for basic scheduling problems. european journal of operational research, 1993, vol. 64, no 2, p. 278-285. Citado por: Ibid.
[46] ZHANG, et al. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. En: computers & operations research, 2007. Vol. 34, no. 11, p. 3229-3242.
[47] GE, Hong-Wei, et al. Op. cit.
[48] ZHANG, et al. A very fast TS/SA algorithm for the job shop scheduling problem. En: computers & operations research, 2008. Vol. 35, no. 1, p. 282-294.

memoria lo que puede afectar su efectividad, sin embargo, la combinación de estos dos algoritmos provee mejoras en las soluciones.

Según Sha y Lin[49], Xia y Wu[50] y Zhang, et al.[51] en los algoritmos más utilizados para la solución del JSSP se destacan el recocido simulado (SA), algoritmo genético (GA), búsqueda Tabú (TS) los cuales son combinados entre sí y con otros algoritmos para generar híbridos. Esto se evidencia en la revisión literaria realizada.

Por otra parte, con el fin de acercar más el problema a la realidad de las empresas se han realizado investigaciones con algunos de los algoritmos mencionados anteriormente que consideran más de un objetivo, como Sha y Lin (2010)[52] quienes propusieron un algoritmo PSO modificado para el problema del Job Shop Scheduling multi-objetivo, considerando los criterios de minimización del makespan, tardanza total y tiempo total de inactividad de las máquinas, al igual que LEI y WU (2006)[53] quienes buscan minimizar el makespan y tiempo de tardanza total con el algoritmo evolutivo crowding-measure-based multiobjective. También SURESH y MOHANASUNDARAM (2006)[54] con el objetivo de minimizar el *makespan* y el *mean flow time* proponen una meta-heurística llamada *Pareto archived simulated annealing* (PASA).

Para finalizar, en las últimas investigaciones (2015) del tema se consideran factores como la minimización del consumo de energía Tang y Dai[55], la

[49] SHA, D. Y.; LIN, Hsing-Hung. A multi-objective PSO for job-shop scheduling problems. En: Expert Systems with Applications, 2010, vol. 37, no 2, p. 1065-1070.
[50] XIA; WU. Op. cit.
[51] ZHANG. Op. Cit.
[52] SHA; LIN. Op. cit.
[53] LEI, Deming; WU, Zhiming. Crowding-measure-based multiobjective evolutionary algorithm for job shop scheduling. En: The International Journal of Advanced Manufacturing Technology, 2006, vol. 30, no 1-2, p. 112-117.
[54] SURESH, R. K.; MOHANASUNDARAM, K. M. Pareto archived simulated annealing for job shop scheduling with multiple objectives. En: The International Journal of Advanced Manufacturing Technology, 2006, vol. 29, no 1, p. 184-196.
[55] TANG, Dunbing; DAI, Min. Energy-efficient approach to minimizing the energy consumption in an extended job-shop scheduling problem. En: Chinese Journal of Mechanical Engineering, 2015, vol. 28, no 5, p. 1048-1055.

restricción de operarios Mencia, et ál[56]; Sierra, Mencia y Varela.[57]; Mencia, et ál[58] y el estudio de problemas Job Shop específicos como lo hacen Jia, Jiang y Li[59] y Jong y Lai[60] buscando involucrar factores representativos de la realidad.

6.2. MÉTODO DE SOLUCIÓN: GRASP

GRASP *Greedy Randomized Adaptive Search Procedure* es una metaheurística iterativa, en este procedimiento cada iteración consta de dos fases, una de construcción en la cual se obtiene una solución factible que posteriormente es mejorada en la fase de búsqueda local. Varios autores como Duarte y Martí (2006)[61], Alvarez-Valdes, Parreño y Tamarit (2008)[62] atribuyen el desarrollo de esta a Feo y Resende (1989)[63] y el acrónimo a Feo, Resende y Smith (1994)[64]. Según Marinakis y Migdalas (2004)[65] el GRASP se ha popularizado para la solución de problemas de optimización combinatoria, algunos de estos son: problemas de *scheduling* y localización.

[56] MENCÍA, Raúl, et al. Memetic algorithms for the job shop scheduling problem with operators. En: Applied Soft Computing, 2015, vol. 34, p. 94-105.
[57] SIERRA, María R.; MENCÍA, Carlos; VARELA, Ramiro. New schedule generation schemes for the job-shop problem with operators. En: Journal of Intelligent Manufacturing, 2013, vol. 26, no 3, p. 511-525.
[58] MENCÍA, Carlos, et al. Solving the job shop scheduling problem with operators by depth-first heuristic search enhanced with global pruning rules. En: AI Communications, 2015, vol. 28, no 2, p. 365-381.
[59] JIA, Wenyou; JIANG, Zhibin; LI, You. Scheduling to minimize the makespan in large-piece one-of-a-kind production with machine availability constraints. En: Expert Systems with Applications, 2015, vol. 42, no 23, p. 9174-9182.
[60] JONG, Wen-Ren; LAI, Po-Jung. The navigation process of mould-manufacturing scheduling optimisation by applying genetic algorithm. En: International Journal of Computer Integrated Manufacturing, 2014, no ahead-of-print, p. 1-19.
[61] DUARTE, Abraham; MARTÍ, Rafael. Tabu search and GRASP for the maximum diversity problem. En: European Journal of Operational Research, 2007, vol. 178, no 1, p. 71-84.
[62] ALVAREZ-VALDÉS, Ramón; PARREÑO, Francisco; TAMARIT, José Manuel. Reactive GRASP for the strip-packing problem. En: Computers & Operations Research, 2008, vol. 35, no 4, p. 1065-1083.
[63] FEO, Thomas A.; RESENDE, Mauricio GC. A probabilistic heuristic for a computationally difficult set covering problem. En: Operations research letters, 1989, vol. 8, no 2, p. 67-71. Citado por DUARTE, Op cit., Ibid.
[64] FEO, Thomas A.; RESENDE, Mauricio GC; SMITH, Stuart H. A greedy randomized adaptive search procedure for maximum independent set. En: Operations Research, 1994, vol. 42, no 5, p. 860-878. Citado por: DUARTE, Op cit.
[65] MARINAKIS, Yannis; MIGDALAS, Athanasios; PARDALOS, Panos M. Expanding neighborhood GRASP for the traveling salesman problem. En: Computational Optimization and Applications, 2005, vol. 32, no 3, p. 231-257.

Algunas investigaciones en las cuales se ha estudiado el GRASP son las siguientes:

FEO Y RESENDE (1995)[66] Definen varios componentes que comprenden el GRASP y el paso a paso para el desarrollo de esta heurística en problemas de optimización combinatoria. Se menciona que la clave para el éxito de un algoritmo de búsqueda local consiste de una elección adecuada de la estructura de vecindario, técnicas de búsqueda eficiente y la solución de partida. También se resalta como característica atractiva del GRASP la facilidad con la cual este puede ser implementado. Además realizan revisión de las aplicaciones del GRASP y dos implementaciones industriales en dos sistemas de soporte de decisiones a gran escala INSITES y OASIS.

AHUJA, Ravindra K.; ORLIN, James B.; TIWARI, Ashish[67] **(2000)** En su investigación *A Greedy Genetic Algorithm for the Quadratic Assignment Problem* proponen un método de solución que utiliza la fase de construcción del GRASP como generador de la población inicial y los pasos correspondientes al procedimiento del Algoritmo Genético. En el paso de selección se elimina la evaluación de los candidatos y se realiza de manera aleatoria. Con respecto al operador de cruce, en este documento se proponen dos alternativas que, según los autores, arrojan mejores resultados que lo operadores clásicos dado que se adaptan mejor a las condiciones que se obtienen cuando se utiliza una metaheurística para la generación de la población inicial; los operadores mencionados anteriormente son: *path crossover scheme* y *optimized crossover scheme.* En el paso de mutación los autores decidieron trabajar con un proceso denominado inmigración, el cual consiste en reemplazar a los candidatos más pobres de la población para obtener nuevos candidatos. En esta investigación

[66] FEO, Thomas A.; RESENDE, Mauricio GC. Greedy randomized adaptive search procedures. En: Journal of global optimization, 1995, vol. 6, no 2, p. 109-133.

[67] AHUJA, Ravindra K.; ORLIN, James B.; TIWARI, Ashish. A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 2000, vol. 27, no 10, p. 917-934.

se destaca la importancia de mantener la diversidad en la población inicial evidenciando resultados computacionales en los cuales los peores resultados se obtienen cuando se limitaba dicha diversidad en la población.

RIBEIRO, UCHOA y WERNECK (2000)[68] Desarrollan un GRASP híbrido con perturbaciones para el *Steiner problem in graphs*. En esta investigación se propone un cambio en la fase de construcción de la meta-heurística GRASP combinando otras heurísticas de construcción las cuales utilizan una estrategia de perturbaciones de peso. Estas perturbaciones de peso se llevan a cabo bajo una oscilación estratégica que combina elementos de intensificación y diversificación. También se utiliza el conocido *Path Relinking* como intensificación de la optimización, para posteriormente evaluar sus resultados en instancias del *benchmark* demostrando ser superior en la calidad de las soluciones.

RESENDE y RIBEIRO (2001)[69] Tratan el problema de *Private Virtual Circuit Routing* con la meta-heurística GRASP mejorado con *Path relinking*. La metodología que estos autores propusieron fue una comparación de los resultados que se obtienen mediante variantes de la meta-heurística. Una variante es la utilización del GRASP sin ninguna mejora, las siguientes incluyen el *Path Relinking* como estrategia de intensificación, el cual crea un grupo elite de soluciones en la fase de búsqueda local comparándolas con las obtenidas en la fase de construcción; estas variantes se diferencian en el punto de partida y la solución guía intercambiando una solución elite escogida al azar y el óptimo local. Por otra parte, la última variante combina estas dos propuestas. Como conclusión final se obtiene la variante que combina GRASP con *Path Relinking* y que inicia desde la solución elite al azar guiada por la solución óptima obtiene mejores resultados que las demás.

[68] RIBEIRO, Celso; UCHOA, Eduardo; WERNECK, Renato F. A hybrid GRASP with perturbations for the Steiner problem in graphs. En: informs journal on computing, 2002. Vol. 14, no. 3, p. 228-246.
[69] RESENDE, Mauricio GC; RIBEIRO, Celso C. A GRASP with path-relinking for private virtual circuit routing. En: Networks, 2003, vol. 41, no 2, p. 104-114.

BAHIENSE, et al. (2001)[70] Resuelven un modelo llamado *Mixed Integer Disjunctive Model for Transmission Network Expansion* usando *Branch and Bound* y la meta-heurística GRASP. El modelo convencional tiene la característica de tener restricciones no lineales, sin embargo, esto se trata trabajando con la forma disyuntiva equivalente a las mismas; la meta-heurística GRASP se utiliza con el fin de obtener una cota superior y reducir el tiempo de búsqueda en el árbol de soluciones minimizando la brecha entre las soluciones obtenidas y la solución óptima, lo cual resulta útil en la evaluación de instancias de gran tamaño y de carácter más real.

RESENDE Y REBEIRO (2002)[71] Describen los parámetros del GRASP, siendo los dos principales los que están relacionados con el criterio de parada y la calidad de los elementos de la lista restringida de candidatos o RCL. Además, presentan mejoras y técnicas alternativas para la fase de construcción, las cuales incluyen: GRASP reactivo, perturbaciones de costos en lugar de selección aleatoria, funciones *bias*, memoria y aprendizaje y búsqueda local en soluciones parcialmente construidas, también se estudia el efecto de *path relinking* en el GRASP, con cuatro variantes de este. Aplicaciones e híbridos también son tratados.

BARD, KONTORAVDIS y YU (2002)[72] Proponen un *Branch-and-Cut Procedure* para el problema de ruteo de vehículos con ventanas de tiempo; La solución a encontrar en este problema es minimizar la cantidad de vehículos que se requieren para las condiciones del mismo, sujeto a algunas restricciones de capacidad y de tiempo. En el proceso de solución se utiliza la meta-heurística GRASP como medio para la creación de soluciones factibles que serán

[70] BAHIENSE, Laura, et al. A mixed integer disjunctive model for transmission network expansion. En: Power Systems, IEEE Transactions, 2001, vol. 16, no 3, p. 560-565.
[71] RESENDE, et al. Handbook of Metaheuristics. Op. cit.
[72] BARD, Jonathan F.; KONTORAVDIS, George; YU, Gang. A branch-and-cut procedure for the vehicle routing problem with time windows. En: Transportation Science, Mayo 2002, vol. 36, no 2, p. 250-269.

mejoradas con el procedimiento propuesto. Se evalúa en instancias del problema a tratar, evidenciando un comportamiento típico en las instancias de 50 nodos.

ALVAREZ-VALDES, et al. (2002)[73] Consideran el problema *large-scale guillotine (un)constrained two-dimensional cutting* en donde se utilizan diferentes algoritmos para su solución. Este problema como se menciona se trata con restricciones y sin restricciones lo cual hace que diferentes algoritmos puedan ser aplicados y evaluados, en primera instancia se utilizan algoritmos de construcción como el *Upper Bound,* después se utiliza el GRASP el cual parte de la construcción hecha por los procedimientos anteriores y realiza su procedimiento normal. Una búsqueda Tabú también se evalúa en esta investigación partiendo de algunos resultados obtenidos con él.

PIÑANA et al. (2004)[74] Proponen un método heurístico basado en la metodología GRASP que genera soluciones de calidad para el problema de minimizar el ancho de banda de la matriz, el cual busca encontrar una permutación de filas y columnas de una matriz dada, que mantenga los elementos distintos de cero en una banda que es lo más cercana posible a la diagonal principal. Además, al ser acoplada con *path relinking* mejora los resultados, resultando competitivo frente a un algoritmo de búsqueda tabú, considerado uno de los mejores para este problema.

MARINAKIS, MIGDALAS y PARDALOS (2005)[75] Se presenta una versión modificada del GRASP al TSP (Traveling Salesman Problem), el cual en la fase de búsqueda local utiliza dos diferentes estrategias de búsqueda basadas en métodos 2-opt y 3-opt. La diferencia más importante es la forma en la que la RCL

[73] ALVAREZ-VALDÉS, Ramón, et al. A tabu search algorithm for large-scale guillotine (un) constrained two-dimensional cutting problems. En: Computers & Operations Research, 2002, vol. 29, no 7, p. 925-947.
[74] PINANA, Estefanıa, et al. GRASP and path relinking for the matrix bandwidth minimization. En: European Journal of Operational Research, 2004, vol. 153, no 1, p. 200-210.
[75] MARINAKIS; MIGDALAS y PARDALOS. Op. cit. p. 231-257.

es construida en la primera fase y la estrategia usada para explorar el vecindario durante la búsqueda local en la segunda fase.

AIEX et al. (2005)[76] Proponen variantes del GRASP con *path relinking* para problemas de asignación de tres índices. Los resultados de los experimentos computacionales arrojan mejor rendimiento del GRASP con *path relinking* en comparación con el GRASP puro. Además, se consideran siete variantes, GRASP, GPR(RAND), GPR(ALL), GPR(RAND,POST), GPR(ALL,POST), GPR(RAND,POST,INT), GPR(ALL,POST,INT).

GUPTA Y SMITH (2006)[77] El problema de *single machine total tardiness scheduling with sequence dependent setups* es tratado aquí y evaluado con dos algoritmos; una búsqueda local basada en el espacio y GRASP. En este caso, como en los anteriores, se opta por hacer algunas modificaciones al GRASP básico, no obstante, en la presente investigación se alteran las dos fases de la meta-heurística. En la fase de construcción se evalúan tres funciones de costo, mientras que en la fase de búsqueda local se implementa una búsqueda en el vecindario variable o VNS por sus siglas en inglés y una búsqueda desendiente basada en entornos variables o VND. Una operación post-optimización se considera para mejorar el algoritmo, en este caso se utiliza *Path Relininking*. Finalmente, se evalúan los algoritmos y se comparan con otros, mostrando que el GRASP propuesto, aunque tiene mayor tiempo de computación que otros algoritmos provee soluciones más robustas y de mejor calidad.

BOUDIA, LOULY y PRINS (2006)[78] Presentan la meta-heurística GRASP como método de solución para el problema de producción y distribución combinados.

[76] AIEX, Renata M., et al. GRASP with path relinking for three-index assignment. En: INFORMS Journal on Computing, 2005, vol. 17, no 2, p. 224-247.
[77] GUPTA, Skylab R.; SMITH, Jeffrey S. Algorithms for single machine total tardiness scheduling with sequence dependent setups. En: European Journal of Operational Research, 2006, vol. 175, no 2, p. 722-739.
[78] BOUDIA, Mourad; LOULY, Mohamed Aly Ould; PRINS, Christian. A reactive GRASP and path relinking for a combined production–distribution problem. En: Computers & Operations Research, 2007, vol. 34, no 11, p. 3402-3419.

En este caso abordan tres maneras de resolverlo; una de ellas es el GRASP en su esquema básico, el cual se escoge por su estructura simple y por el pequeño número de parámetros que necesita en su desarrollo, estas ventajas se ven opacadas en la fase de búsqueda local propia del algoritmo, dado a la combinación de dichos parametros, estos inconvenientes tratan de resolverse mediante las otras dos propuestas. El GRASP reactivo pretende optimizar el tamaño de la lista restringida de candidatos que se forma en la fase de construcción del algoritmo y el GRASP con *Path Relinking* utiliza un grupo de soluciones elite de una "piscina de soluciones" para mejorar la búsqueda local. Los resultados demuestran que las dos mejoras propuestas son superiores al algoritmo básico en instancias más grandes.

DUARTE y MARTÍ (2007)[79] Desarrollan un nuevo procedimiento heurístico para el problema de máxima diversidad *maximum diversity problem (MDP)*. Proponen cuatro métodos constructivos basados en búsqueda tabú que agregan estructura de memoria y cuatro construcciones de menos memoria basados en la metodología GRASP, concluyendo de los experimentos que el uso de memoria es efectivo en la construcción y mejora para dar solución a este problema. Además, en instancias medianas y grandes supera a las mejores heurísticas.

ALVAREZ, PARREÑO y TAMARIT (2008)[80] Presentan un GRASP para el *strip packing problem* que arrojan mejores resultados en comparación a otras metaheurísticas, lo cual se le atribuye a tres aspectos; uno de los cuales es que el GRASP se adapta particularmente a este problema.

PAOLA FESTA AND MAURICIO G.C. RESENDE (2008)[81] Presentan un resumen de las formas y los algoritmos utilizados para mejorar los resultados que se obtienen mediante la aplicación de la meta-heurística GRASP en su

[79] DUARTE; MARTÍ. Op. cit. p. 71-84.
[80] ALVAREZ-VALDÉS, Op cit.
[81] FESTA, Paola; RESENDE, Mauricio GC. Hybrid GRASP heuristics. En: Foundations of Computational Intelligence Volume 3. Springer Berlin Heidelberg, 2009. p. 75-100.

esquema básico. En la fase de construcción se mencionan algunos como: GRASP reactivo, el cual pretende mejorar la selección del tamaño de la RCL; perturbaciones de costos, permite agregar un poco de ruido a los costos de las soluciones; funciones de *Bias*, estas funciones permiten dar una distribución de probabilidad a cada elemento de la RCL, lo cual permite conocer un poco cómo se seleccionan dichos elementos. Posteriormente se presenta la combinación de GRASP con *Path Relinking* la cual en la literatura ha tenido un buen desempeño, se mencionan las dos maneras de utilizarlo, como método de post-optimización o como estrategia de intensificación de la búsqueda local, enfocando su atención a la última ya que demuestra ser la mejor. La Búsqueda Tabú se ha utilizado para reemplazar la búsqueda local estándar que utiliza GRASP al igual que la búsqueda local iterativa, la denominada *very-large scale neighborhood search* y Recocido Simulado. Para meta-heurísticas evolutivas se utiliza el GRASP para obtener la solución inicial. Finalmente se aborda el tema de las estructuras de vecindario como VNS y VND y otras hibridaciones.

TASGETIREN, PAN y LIANG (2009)[82] Desarrollan un *discrete differential evolution algorithm* para el problema de *Single machine total weighted tardiness* o Tardanza promedio ponderada total en una máquina con tiempos de ajuste dependientes de una secuencia. El algoritmo se mejora en el esquema de la población inicial con algunas heurísticas constructivas entre las cuales se encuentra GRASP, también algunos métodos de aceleración fueron incluidos en este procedimiento, los cuales se usaron por primera vez en la literatura. Este procedimiento tuvo excelentes resultados en comparación con los existentes, demostrando que las mejoras que se pensaron fueron una alternativa acertada, es decir, la utilización del GRASP y otras heurísticas contribuyeron en la generación de mejores soluciones.

[82] TASGETIREN, M. Fatih; PAN, Quan-Ke; LIANG, Yun-Chia. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. En: Computers & Operations Research, 2009, vol. 36, no 6, p. 1900-1915.

RESENDE et al. (2010)[83] Proponen un método heurístico basado en GRASP y *path relinking* para el problema de diversidad max-min, el cual busca solucionar un subconjunto de elementos de un conjunto dado de tal manera que la diversidad entre los elementos seleccionados sea maximizada. Se estudiaron y probaron variantes de estos híbridos, dando como resultado que las variantes dinámicas de GRASP con *greedy* PR y GRASP con PR evolutivo son los mejores métodos para solucionar el MMDP en las instancias probadas.

MARINAKIS y MARINAKI (2010)[84] Proponen un híbrido entre algoritmo Genético y PSO, el cual utiliza dos métodos para mejorar su calidad; el *Expanding Neighborhood Search* (ENS) se utiliza como estrategia de velocidad para el algoritmo. Por otro lado, también se maneja un método para generar la población inicial, que en este caso es una variación del GRASP llamada MPNS-GRASP por sus siglas en inglés de *Multiple Phase Neighborhood Search* GRASP. La diferencia de este algoritmo con el GRASP convencional radica en que brinda flexibilidad a cada iteración aplicando diferentes funciones *"Greedy".* Esta combinación de nuevas estructuras permite a los autores comparar con el híbrido básico dando como resultado una mejora en los tiempos de computación y calidad de la solución.

MARINAKIS y MARINAKI (2010)[85] Vuelven a abordar un problema de ruteo, como lo es *probabilistic traveling salesman problem* utilizando el algoritmo propuesto en el mismo año con la diferencia de ser multi-objetivo; al igual que en el problema anterior se utilizó la meta-heurística GRASP con búsqueda en el vecindario expandida para crear una población inicial y en su evaluación se

[83] RESENDE, Mauricio GC, et al. GRASP and path relinking for the max–min diversity problem. En: Computers & Operations Research, 2010, vol. 37, no 3, p. 498-508.
[84] MARINAKIS, Yannis; MARINAKI, Magdalene. A hybrid genetic–Particle Swarm Optimization Algorithm for the vehicle routing problem. En: Expert Systems with Applications, 2010, vol. 37, no 2, p. 1446-1455.
[85] MARINAKIS, Yannis; MARINAKI, Magdalene. A hybrid multi-swarm particle swarm optimization algorithm for the probabilistic traveling salesman problem. En: Computers & Operations Research, 2010, vol. 37, no 3, p. 432-442.

obtienen muy buenos resultados dejando como evidencia la eficiencia de este híbrido.

MARINAKIS, MARINAKI y DOUNIAS (2010)[86] Aquí estos autores proponen, a diferencia de las investigaciones previamente mencionadas, un híbrido que excluye el algoritmo genético y se enfoca solamente en PSO como base, usando de la misma manera la meta-heurística GRASP con una fase de búsqueda en el vecindario múltiple. Estas investigaciones permiten concluir que GRASP ayuda a mejorar la fase de construcción de la población inicial, sin embargo, es importante resaltar que en las tres investigaciones se combinó con una estructura de vecindario mejorada lo cual garantizó la eficiencia en los tres diferentes problemas de ruteo presentados.

MARINAKIS, MARINAKI y DOUNIAS (2011)[87] Presentan un nuevo algoritmo híbrido, que combina *Honey Bees Mating Optimization* con *Multiple Phase Neighborhood Search-Greedy Randomized Adaptive Search Procedure (MPNS-GRASP)* y *Expanding Neighborhood Search Strategy.* El MPNS-GRASP, el cual es una versión modificada del GRASP, es utilizado para crear la solución inicial, la diferencia más importante es la construcción de la lista RCL y la aplicación de las funciones codiciosas alternativas en cada iteración frente al GRASP clásico que solo tiene una única función codiciosa simple. Los resultados de las instancias son satisfactorios, tanto en calidad como eficiencia computacional.

[86] MARINAKIS, Yannis; MARINAKI, Magdalene; DOUNIAS, Georgios. A hybrid particle swarm optimization algorithm for the vehicle routing problem. En: Engineering Applications of Artificial Intelligence, 2010, vol. 23, no 4, p. 463-472.

[87] MARINAKIS, Yannis; MARINAKI, Magdalene; DOUNIAS, Georgios. Honey bees mating optimization algorithm for the Euclidean traveling salesman problem. En: Information Sciences, 2011, vol. 181, no 20, p. 4684-4698.

6.3. GRASP EN JOB SHOP

El primer GRASP que se propuso para la minimización del *makespan* en el JSSP fue por parte de **BINATO et al (2002)[88].** En esta investigación a diferencia del GRASP convencional los candidatos de la RCL son seleccionados con la función aleatoria de Bresina[89] para restringir los elementos. Además de esto, se utiliza una estrategia de intensificación que busca a través de un grupo de soluciones élite guiar la fase de construcción junto con el concepto de *Proximate Optimality Principle*, principio de optimalidad próxima POP, el cual evalúa los elementos seleccionados en un *scheduling* parcial con el fin de mejorar el *makespan* del mismo. Las soluciones encontradas por el algoritmo mostraron ser cercanas al óptimo en las instancias evaluadas, adicionalmente recomendaron que para instancias que representan mayor dificultad sea utilizado un esquema de intensificación como el *path relinking.*

Posteriormente, las investigaciones realizadas que han estudiado la aplicación de la meta-heurística GRASP al problema del *Job Shop Scheduling* son las siguientes:

AIEX, BINATO y RESENDE (2003)[90] en su investigación *Parallel GRASP with path-relinking for job shop scheduling* con el objetivo de minimizar el *makespan* muestran el desarrollo de un algoritmo híbrido en donde se utiliza el mencionado *Path Relinking* como estrategia de intensificación y post optimización. En la fase de construcción se utilizó *two exchangued local search* basado en la gráfica disyuntiva al igual que en 2002 por Resende. Lo que hace paralelo este procedimiento es que se evalúa en dos condiciones denominadas no colaborativa y colaborativa, las cuales se diferencian en la comunicación que tienen los procesos para intercambiar información con respecto a sus grupos elite. Al evaluar el algoritmo en las instancias y compararlo con el GRASP clásico

[88] BINATO, S., et al. Op. cit.
[89] BRESINA, John L. Heuristic-biased stochastic sampling. En: AAAI/IAAI, Vol. 1. 1996. p. 271-278. Citado por: BINATO, S., et al. A GRASP for job shop scheduling. En: Essays and surveys in metaheuristics. Springer US, 2002. p. 59-79.
[90] AIEX, Renata M.; BINATO, Silvio; RESENDE, Mauricio GC. Parallel GRASP with path-relinking for job shop scheduling. En: Parallel Computing, 2003, vol. 29, no 4, p. 393-430.

y el GRASP con intensificación POP demuestra que, aunque implica mayor tiempo computacional la respuesta es de mejor calidad.

FERNANDES y LOURENÇO (2007)[91] Presentan un algoritmo simple para el JSP que combina GRASP con *branch and bound,* el cual es utilizado para resolver subproblemas de programación de una máquina. Comparado con otros procedimientos similares se logran mejores soluciones y de manera mas rápida.

CHASSAING, et al. (2014)[92] proponen un enfoque de GRASP combinado con ELS o *Evolutionary Local Search* para el problema de *job-shop with a web service paradigm packaging.* En esta investigación La fase de construcción del GRASP se mantiene igual mientras que en la fase de búsqueda local se intensifica con una ELS, la cual crea muchas soluciones a partir de una principal en cada iteración, esta meta-heurística se escogió por su velocidad y por su capacidad de generar muchas soluciones en cada iteración, lo que permite tener mayor espacio explorado y evita evaluar una solución más de una vez. Su evaluación se hace en las diferentes escalas de instancias demostrando efectividad y eficiencia en las soluciones arrojadas.

[91] FERNANDES, Susana; LOURENÇO, Helena R. A GRASP and branch-and-bound metaheuristic for the job-shop scheduling. En: Evolutionary Computation in Combinatorial Optimization. Springer Berlin Heidelberg, 2007. p. 60-71.

[92] CHASSAING, Maxime, et al. A GRASP× ELS approach for the job-shop with a web service paradigm packaging. En: Expert Systems with Applications, Febrero, 2014, vol. 41, no 2, p. 544-562.

7. MARCO DE ANTECEDENTES

En la Universidad Industrial de Santander, específicamente en la Escuela de Estudios Industriales y Empresariales se han realizado tres proyectos de grado relacionados al *Job Shop Scheduling Problem.* De un lado, se encuentran Aguilar Karin y Pérez Yuleiny[93], quienes en su investigación **"Un algoritmo memético para la minimización del makespan en el problema del Job Shop Scheduling**" definen la problemática del *Job Shop* y los métodos que se han utilizado para su solución, siendo aquellos de aproximación los más utilizados en los últimos tiempos. También, se plantea que la existencia de estos métodos de solución no ha estado limitada al desarrollo único del algoritmo original, con el paso del tiempo y la necesidad creciente de acercarse a la solución óptima se han llegado a combinar los métodos para mejorarlos o complementarlos, adaptándose a los criterios de optimización, complejidad y estructura del problema. Esto se tomó en cuenta para el desarrollo del algoritmo, realizando una combinación entre el proceso genético y búsqueda local, dicha combinación representa una ventaja para resolver problemas de gran tamaño. Además, se utilizó la estructura de vecindarios propuesta por Nowicki y Smutnicki[94], también denominado N_5, para obtener mejores soluciones. Para la validación del algoritmo se utilizaron las instancias más estudiadas en la literatura demostrando ser competitivos en comparación con otros métodos.

Por otro lado, Cindy Sarmiento[95] en el desarrollo de su proyecto de grado titulado **"Método *Particle Swarm Optimization* (PSO – Enjambre de partículas)**

[93] AGUILAR, Karin y PÉREZ, Yuleiny. Un algoritmo memético para la minimización del makespan en el problema del Job shop scheduling. Bucaramanga, 2012, 223p. Tesis (ingeniería industrial). Universidad industrial de Santander. Facultad de ingeniería físicomecánicas. Escuela de estudios industriales y empresariales.

[94] NOWICKI, Eugeniusz y SMUTNICKI, Czeslaw. A fast taboo search algorithm for the job shop problem. En: Management Science Vol. 42.(1996),p.797-813 citado por AGUILAR, Karin y PÉREZ, Yuleiny. Un algoritmo memético para la minimización del makespan en el problema del Job shop scheduling. Bucaramanga, 2012, 223p. Tesis (ingeniería industrial). Universidad industrial de Santander. Facultad de ingeniería físicomecánicas. Escuela de estudios industriales y empresariales.

95 SARMIENTO, Cindy Johanna. Método particle swarm optimization (PSO-Enjambre de partículas) aplicado al problema de múltiples objetivos del Job Shop Scheduling (JSP) o secuenciamiento de máquinas. Bucaramanga, 2012, 134p. Tesis (ingeniería industrial). Universidad industrial de Santander. Facultad de ingeniería físicomecánicas. Escuela de estudios industriales y empresariales.

aplicado al problema de múltiples objetivos del *Job Shop Scheduling* (JSP) o secuenciamiento de máquinas" realiza una recopilación sobre los métodos para resolver el JSP, profundizando en el método de optimización por enjambre de partículas, el cual busca la optimización de dos objetivos, la minimización del *makespan* y el tiempo de tardanza total. Además, para la aplicación eficiente del PSO en el JSP fue necesario realizar modificaciones tomando en cuenta lo propuesto por Sha y Lin (2010), presentando como resultado el pseudocódigo para su solución.

Finalmente, Juan Gómez y Edwin Orduz[96], en la investigación realizada buscaron la **Minimización del *makespan* en el problema de Job Shop Flexible con restricciones de transporte utilizando algoritmo genético**. Para el cual se debe considerar primero la asignación de cada operación a cada una de las maquinas disponibles y luego la programación de estas, la cual minimice el *makespan.* Adicionalmente, se tienen en cuenta restricciones de transporte entre máquinas considerando recursos idénticos de transporte y que el tiempo de este es dependiente de la trayectoria entre las máquinas. Como resultado se genera un pseudocódigo que al probarlo evidencia ser una metaheurística efectiva para este tipo de problemas.

Siendo el JSP el problema a solucionar, el marco teórico contenido en estos proyectos son una base para la contextualización del problema, adicionalmente, la biblioteca de *benchmark problems* conformada en uno de estos es de gran utilidad para probar el algoritmo a desarrollar y medir su efectividad.

[96] GÓMEZ, Juan y ORDUZ, Edwin. Minimización del makespan en el problema de Job Shop Flexible con restricciones de transporte utilizando algoritmo genético. Bucaramanga, 2015, 126p. Tesis (ingeniería industrial). Universidad industrial de Santander. Facultad de ingeniería físicomecánicas. Escuela de estudios industriales y empresariales.

8. COMBINACIÓN DE LA METAHEURÍSTICA GRASP

Como se evidenció y se mencionó en la revisión de literatura enfocada al *Job Shop Scheduling Problem*, autores como Sha y Lin, Xia y Wu y Zhang, et al. destacan que los algoritmos más utilizados para dar solución al JSSP son, el recocido simulado (SA), algoritmo genético (GA), búsqueda Tabú (TS), e hibridaciones de estos mismos. Teniendo en cuenta esto y la investigación encontrada en la revisión acerca del método de solución GRASP titulada "A Greedy Genetic Algorithm for the Quadratic Assignment Problem" (Un algoritmo genético codicioso para el problema de asignación cuadrático) se toma la decisión de realizar la combinación de la metaheurística GRASP con algoritmo genético, dado al éxito que ha tenido este último en la solución de problemas de este tipo. Por esta razón, se realiza una breve revisión del algoritmo genético aplicado al JSSP, como contextualización y análisis para el diseño del algoritmo a proponer.

8.1. REVISIÓN DE LITERATURA ALGORITMO GENÉTICO

LI, Ye; CHEN, Yan (2010)[97] Proponen una codificación para el problema del *Job Shop Scheduling* que se basa en la secuenciación del trabajo y la distribución de las máquinas, en donde se garantiza el orden de las operaciones, es decir, la factibilidad del cromosoma representante, y que cada operación se asigne a las máquinas que pueden procesarlas. Por otro lado la selección se basa en la función *Fitness,* que para este caso, será el inverso de la función objetivo, dando como resultado valores más altos en la función para los valores más pequeños de *makespan.* Se reconoce la importancia del operador de mutación para evitar la convergencia temprana en óptimos locales, finalmente este proceso se valida en una problema de cinco máquinas y ocho procesos.

[97] LI, Ye; CHEN, Yan. A genetic algorithm for job-shop scheduling. *Journal of software*, 2010, vol. 5, no 3, p. 269-274.

ZHOU, Hong; FENG, Yuncheng; HAN, Limin (2001)[98] En esta investigación los autores modifican las propiedades del Algoritmo Genético tradicional incluyendo algunas heurísticas que restringen el espacio de soluciones y direccionan el proceso de búsqueda. Esta modificación conlleva a un proceso de construcción de soluciones más complejo, en donde las primeras operaciones de cada máquina se programaran de acuerdo a un Algoritmo Genético y las demás de acuerdo a reglas de prioridad que, para este caso, son *Shortest Processing Time (STP)* y *Most Work Remaining (MWKR);* como su nombre lo infiere estas reglas dan prioridad a algunas operaciones, siendo las aquí mencionadas aquellas con el menor tiempo de procesamiento y aquellas con el mayor trabajo restante, respectivamente. En cuanto a la reproducción de las soluciones obtenidas en esta investigación se seleccionó un operador de cruce de dos puntos y una probabilidad de mutación que se reduce gradualmente según la diversidad de la población, dada por una función del valor *fitness* de sus integrantes, dando así estructura a la metaheurística.

FALKENAUER, Emanuel; BOUFFOUIX, S. (1991)[99] Estos autores realizan una descripción detallada del procedimiento del Algoritmo Genético y de los operadores que se deben utilizar en el mismo. Partiendo de la importancia que tiene la codificación del problema, en esta investigación se ejemplariza de manera clara cómo abordan el problema en cuanto a los operadores de cruce y mutación, definiendo dos operadores de cruce, *OX (Order Crossover*) y *LX (Linear Order Crossover)* cuya diferencia radica en cómo se mantiene la información de prioridad de las operaciones durante el proceso de cruce; en cuanto al proceso de mutación se incluye un nuevo concepto llamado Inversión, que básicamente se enfoca en la permutación de posiciones en el cromosoma

[98] ZHOU, Hong; FENG, Yuncheng; HAN, Limin. The hybrid heuristic genetic algorithm for job shop scheduling. *Computers & Industrial Engineering*, 2001, vol. 40, no 3, p. 191-200.
[99] FALKENAUER, Emanuel; BOUFFOUIX, S. A genetic algorithm for job shop. En *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*. IEEE, 1991. p. 824-829.

en vez de los valores que en ellos se contienen, reduciendo el posible impacto destructivo de un intercambio.

BIERWIRTH, Christian (1995) [100] En esta investigación, el autor propone un nuevo método de permutación que respeta la factibilidad en los cromosomas representantes de una solución, la cual es denominada «Permutación con repetición». Sin embargo aquí se aclara que este tipo de codificación requiere operadores de cruce que se ajusten a la misma, por lo tanto se presenta un operador llamado *GOX (Generalized Order Crossover),* el cual es una generalización del ya mencionado *OX*. Este operador al tener una pareja de padres, establece un índice para cada gen que no es más que un contador de repetición en el cromosoma; de un padre se selecciona una cadena aleatoria que será implantada en el otro padre eliminando los genes repetidos; el orden en que se implantan depende de donde se encuentre la cadena en el padre donador.

PARK, Byung Joo; CHOI, Hyung Rim; KIM, Hyun Soo (2003) [101] Deciden trabajar con la codificación propuesta por BIERWIRTH[102] dada su efectividad en cuanto a factibilidad se trata. En este *paper* la población inicial se construye con el algoritmo de Gliffler-Thompson.[103], también se muestran algunos operadores de cruce que se ajustan a la codificación presentada y se describen los demás pasos del GA. Por otra parte en este documento se establece una variante del algoritmo, llamado *Parallel Genetic Algorithm,* el cual divide la población inicial en varias sub-poblaciones que serán sujetas al procedimiento del Algoritmo Genético cada una, lo que puede evitar convergencias tempranas en óptimos locales; en este nuevo procedimiento los individuos pertenecientes a

[100] BIERWIRTH, Christian. A generalized permutation approach to job shop scheduling with genetic algorithms. *Operations-Research-Spektrum*, 1995, vol. 17, no 2-3, p. 87-92

[101] PARK, Byung Joo; CHOI, Hyung Rim; KIM, Hyun Soo. A hybrid genetic algorithm for the job shop scheduling problems. *Computers & industrial engineering*, 2003, vol. 45, no 4, p. 597-613.

[102] BIERWIRTH, Christian. Op cit. Citado por: PARK, Byung Joo; CHOI, Hyung Rim; KIM, Hyun Soo. A hybrid genetic algorithm for the job shop scheduling problems. *Computers & industrial engineering*, 2003, vol. 45, no 4, p. 597-613.

[103] GIFFLER, Bernard; THOMPSON, Gerald Luther. Algorithms for solving production-scheduling problems. *Operations research*, 1960, vol. 8, no 4, p. 487-503. Citad por: PARK, Byung Joo; CHOI, Hyung Rim; KIM, Hyun Soo. A hybrid genetic algorithm for the job shop scheduling problems. *Computers & industrial engineering*, 2003, vol. 45, no 4, p. 597-613.

subpoblaciones en algunos intervalos pueden migrar de una a otra dependiendo de su desempeño. Finalmente se comparan los resultados obtenidos en diferentes instancias.

GONÇALVES, Jose Fernando; DE MAGALHÃES MENDES, Jorge José; RESENDE, Maurício GC (2005) [104] En su investigación el Algoritmo Genético se combina con un procedimiento generador de *schedules* y una búsqueda local para mejorar los resultados obtenidos; en este nuevo enfoque el Algoritmo Genético es responsable de transformar los cromosomas dados en otros que tengan reglas de prioridad, el generador de *schedules* utiliza los resultados obtenidos anteriormente y los convierte en *schedules* parametrizados activos y finalmente la búsqueda local mejora dichos *schedules* mediante el procedimiento «*Two Exchange Local Search*» basado en el modelo de grafica disyuntiva de Roy y Sussman[105]. Los resultados finalmente se comparan con otros enfoques y se evidencia la superioridad de este procedimiento en cuanto a calidad de la solución.

WANG, Yuping, et al. (2012) [106] Resaltan la importancia de una codificación que se ajuste a el enfoque que se va a trabajar, por este motivo y basados en una gráfica dirigida la representación utilizada se divide en sub-cadenas, las cuales son iguales a la cantidad de máquinas presentes en el problema, donde cada una representa la secuencia de procesamiento en cada máquina. El operador de cruce en esta investigación fue diseñado para acoplarse a las características del JSP; este operador realiza un intercambio entre las sub-cadenas armando grupos complementarios, dando así como resultado un par de cromosomas hijos. Para el operador de mutación se calcula el camino crítico en uno de los padres y si hay dos operaciones consecutivas en la misma máquina se

[104] GONÇALVES, Jose Fernando; DE MAGALHÃES MENDES, Jorge José; RESENDE, Maurício GC. A hybrid genetic algorithm for the job shop scheduling problem. *European journal of operational research*, 2005, vol. 167, no 1, p. 77-95.
[105] ROY, Bernard; SUSSMANN, B. Les problemes d'ordonnancement avec contraintes disjonctives. *Note ds*, 1964, vol. 9. Citado por: Ibid.
[106] WANG, Yuping, et al. A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Operations Research*, 2012, vol. 39, no 10, p. 2291-2299.

consideran las permutaciones, dando vida a nuevas generaciones. Finalmente, se realiza una búsqueda local que mejora las soluciones obtenidas.

BIERWIRTH, Christian; MATTFELD, Dirk C.; KOPFER, Herbert (1996)[107] Estos autores proponen trabajar con la representación propuesta por BIERWIRTH[108] (1995) cuya ventaja, ya mencionada, es la eliminación de cromosomas que representen soluciones no factibles; por otro lado y de acuerdo con la codificación utilizada, los autores comparan tres tipos de operadores de cruce cuya diferencia radica en la tendencia de mantener el orden de los cromosomas padres, el *GOX,* ya mencionado, mantiene un orden relativo en los cromosomas hijos; el *GPMX (Generalized Partially Mapped Crossover)* que mantiene las posiciones en la permutación y el *PPX (Precedence Preservative Crossover)* que mantiene un orden absoluto de los cromosomas generando una cadena de números aleatorios entre 1 y 2, estableciendo así el orden de llenado con respecto al padre 1 y al padre 2. La comparación entre los tres tipos de operadores arrojó que manteniendo el orden completo de los padres también es posible llegar a soluciones de alta calidad.

MEISEL, José David; PRADO, Liliana Katherine (2010) [109] En este trabajo presentan un algoritmo genético hibrido mejorado para la solución del programación del JSP, el cual parte de la representación planteada por BIERWIRTH et al.[110] Y PARK et al.[111] A partir de esta codificación la generación de la población inicial se realiza con base en el algoritmo G&T.[112] y

[107] BIERWIRTH, Christian; MATTFELD, Dirk C.; KOPFER, Herbert. On permutation representations for scheduling problems. En *Parallel Problem Solving from Nature—PPSN IV*. Springer Berlin Heidelberg, 1996. p. 310-318.

[108] BIERWIRTH, Christian. A generalized permutation approach to job shop scheduling with genetic algorithms. Op cit. Citado por: BIERWIRTH, Christian; MATTFELD, Dirk C.; KOPFER, Herbert. On permutation representations for scheduling problems. En *Parallel Problem Solving from Nature—PPSN IV*. Springer Berlin Heidelberg, 1996. p. 310-318

[109] MEISEL, José DaviD; PRADO, Liliana Katherine. Un algoritmo genético híbrido y un enfriamiento simulado para solucionar el problema de programación de pedidos job shop. *Revista EIA*, 2010, no 13, p. 31-51.

[110] BIERWIRTH et al. Op cit. Citado por: Ibid.

[111] PARK et al Op cit. Citado por: MEISEL, José DaviD; PRADO, Liliana Katherine. Op. cit.

[112] GIFFLER, Bernard; THOMPSON, Gerald Luther. Op. cit. citado por: MEISEL, José DaviD; PRADO, Liliana Katherine. Op. cit.

posteriormente realiza una estrategia que escoge una sub-población, todo esto con el fin optimizar el proceso del Algoritmo Genético Mejorado (AGM); La selección de individuos se realiza por torneo en un tamaño fijo de grupos, posteriormente una estrategia de cruce llamada *«Re-insert»* selecciona dos posiciones aleatorias de un cromosoma padre y un trabajo diferente para ser insertado entre las dos posiciones anteriormente mencionadas, este tipo de cruce respeta la factibilidad de la solución. Uno de los puntos más importantes de este documento está en el paso de mutación, ya que los autores mencionan la gran importancia del mismo en los resultados finales, para afrontar esta situación en este *paper* se recurre al enfriamiento simulado para diversificar la población en este paso. Finalmente para mejorar las soluciones obtenidas se realiza un procedimiento de búsqueda local que intercambia trabajos diferentes y evalúa su función de desempeño.

8.2. ALGORITMO GENÉTICO

Los algoritmos genéticos (AG) son técnicas de búsqueda aleatoria para optimización propuesta por primera vez por J. Holland (1975)[113] la cual se basa en el mecanismo de selección natural del proceso evolutivo.[114] El AG se ejecuta con una población de soluciones a diferencia de otros métodos de optimización que solo trabajan con una sola solución; a cada individuo de esta población le es asignado un valor de acuerdo a la función objetivo del problema en que este siendo aplicado, luego se seleccionan los individuos más aptos de esta población para efectuar la reproducción en la cual se aplican operadores de cruce y/o de mutación y así obtener una nueva población más apta que la anterior.[115] Los algoritmos genéticos han sido aplicados a una amplia variedad de problemas obteniendo resultados exitosos, entre ellos problemas de optimización *NP-Hard* como el *Job Shop Scheduling Problem.*

[113] HOLLAND, John H. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press, 1975. Citado por: ZHOU, Hong; FENG, Yuncheng; HAN, Limin. Op. cit.
[114] ZHOU, Hong; FENG, Yuncheng; HAN, Limin. Op. cit.
[115] PARK, Byung Joo; CHOI, Hyung Rim; KIM, Hyun Soo. Op. cit.

8.2.1. Representación genética: Como primer paso en la construcción del algoritmo genético se debe definir una representación genética o codificación adecuada, siendo esta de gran importancia debido a que afectará los siguientes pasos del algoritmo genético.[116] Esta representación se realiza como una cadena llamada cromosoma, en donde las posiciones de cada cromosoma se denominan genes; cada cromosoma representa una solución para el problema. La representación del algoritmo genético clásico se realiza mediante una cadena de números binarios[117], sin embargo, esta representación no se adapta a cierto tipo de problemas como el Job Shop Scheduling y por esto mismo se han propuesto nuevas representaciones de cromosomas.

8.2.2. Generación de la población inicial: La población inicial es de gran importancia para la calidad del algoritmo genético, en la mayoría de las investigaciones esta se obtiene aleatoriamente[118], sin embargo, existen métodos para la construcción de la población inicial que buscan generar mejores individuos y así mismo mejores soluciones, algunos de los que se han utilizado son, el algoritmo G&T[119], OG&T[120] y GRASP.

[116] LI, Ye; CHEN, Yan. Op. cit.
[117] PARK, Byung Joo; CHOI, Hyung Rim; KIM, Hyun Soo. Op. cit.
[118] FALKENAUER, Emanuel. Op. cit.
[119] PARK, Byung Joo; CHOI, Hyung Rim; KIM, Hyun Soo. Op. cit.
[120] MENCÍA, Raúl, et al. A genetic algorithm for job-shop scheduling with operators enhanced by weak lamarckian evolution and search space narrowing.*Natural Computing*, 2014, vol. 13, no 2, p. 179-192.

8.2.3. Fitness y selección: Se define la función de desempeño que permite determinar la calidad de los individuos, el fitness más alto indica mejor desempeño de los individuos, siendo estos individuos los de mayor probabilidad de sobrevivir; seguido a esto se encuentra la etapa de selección en donde se eligen los dos padres sobre los cuales se efectuará el operador de cruce.[121] Existen diferentes mecanismos de selección, entre ellos se destacan, selección aleatoria, en donde aumenta la probabilidad de ser seleccionados según el valor de su fitness; ranking, en el cual los individuos son clasificados en orden ascendente de su valor fitness y los que se encuentran en las posiciones más altas tienen una mayor probabilidad de ser seleccionados; selección por torneo, en este se eligen dos individuos de la población aleatoriamente y al compararlos se selecciona el más apto, es decir, el de mayor valor fitness para ser reproducido; selección de ruleta, en este caso el mejor cromosoma de la anterior generación es copiado a la siguiente generación, aquí su probabilidad de ser seleccionado es proporcional a su fitness.

8.2.4. Crossover: En esta etapa se realiza la reproducción de los padres seleccionados por medio del operador de cruce para generar dos nuevos cromosomas intercambiando el material genético de los padres. Entre los operadores de crossover tradicionales se encuentra el cruce de un punto y el de dos puntos. Además existen otros operadores de crossover propuestos, especialmente para codificaciones de permutación, algunos como el Partially Matched Crossover (PMX) y el crossover basado en orden Order Crossover (OX), los cuales se derivan del crossover de dos puntos.[122]

[121] ZHOU, Hong; FENG, Yuncheng; HAN, Limin. Op. cit.

[122] CHENG, Runwei; GEN, Mitsuo; TSUJIMURA, Yasuhiro. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers & Industrial Engineering*, 1999, vol. 36, no 2, p. 343-364.

8.2.5. Mutación: Su importancia radica en que se realiza para diversificar la población y evitar la convergencia en un óptimo local[123], para ellos se introduce cambios aleatorios en algunos miembros de la población. También suele ser usada la inmigración por algunos investigadores con el mismo propósito en lugar de mutación, y esta se encarga es de reemplazar individuos pobres de la población actual por nuevos individuos.[124] Entre los operadores de mutación tradicionales se encuentra, la mutación swap, la cual consiste en seleccionar dos puntos aleatorios del cromosoma e intercambiar estos genes que representan operaciones de distinto trabajos. Adicional a este se han propuesto distintos operadores cuya base es el intercambio de genes, diferenciándose en la forma de realizar dicho intercambio; por otro lado, también se ha planteado la implementación de heurísticas en esta etapa.

[123] LI, Ye; CHEN, Yan. Op. cit.
[124] AHUJA, Ravindra K.; ORLIN, James B.; TIWARI, Ashish. Op. cit.

9. DISEÑO DEL ALGORITMO

9.1. REPRESENTACIÓN GENÉTICA

Teniendo en cuenta la importancia de seleccionar una representación adecuada de los individuos por su influencia en la eficiencia del algoritmo genético, para la presente investigación se utiliza la representación genética basada en las operaciones; propuesta por Bierwirth[125] y utilizada por autores como MENCÍA, Raúl, et al; PARK, Byung Joo; CHOI, Hyung Rim; KIM, Hyun Soo; GAO, Jie; GEN, Mitsuo; SUN, Linyan, entre otros; esta codificación está diseñada con base en la permutación con repetición y genera solo soluciones factibles, debido a que mantiene la información acerca de la secuencia de operaciones y las restricciones de precedencia, en esta codificación cada trabajo aparece tantas veces en la permutación como la frecuencia que hay de operaciones asociadas a él.

Para un problema de tres trabajos y tres máquinas la representación de un cromosoma se da de la siguiente manera: [1 2 2 1 3 1 2 3 3], en la cual al leer de izquierda a derecha la ocurrencia del k-ésimo trabajo se refiere a la k-ésima operación del mismo, es decir, el sexto gen de este cromosoma es el 1, indicando la tercera operación del trabajo uno dado que es la tercera vez que se repite este número. En la figura 10 se presenta el cromosoma con un índice de ocurrencia, que indica la operación que se está representando del respectivo trabajo.[126]

Figura 10. Representación de la solución.

Cromosoma	1	2	2	1	3	1	2	3	3
Índice	1	1	2	2	1	3	3	2	3
$O_{ij}: Operación\ i\ del\ trabajo\ j$	O_{11}	O_{12}	O_{22}	O_{21}	O_{13}	O_{31}	O_{32}	O_{23}	O_{33}

[125] BIERWIRTH, Christian. A generalized permutation approach to job shop scheduling with genetic algorithms. Op. cit.
[126] PARK, Byung Joo; CHOI, Hyung Rim; KIM, Hyun Soo. Op. cit.

9.2. GENERACIÓN DE LA POBLACIÓN INICIAL

Cuando en la construcción de la población inicial se generan individuos factibles, diversos y superiores, se pueden lograr soluciones de mejor calidad; además, cuando esta es generada aleatoriamente, como suele realizarse en la mayoría de las investigaciones, puede conllevar a más tiempo de búsqueda para alcanzar una solución óptima y a su vez la posibilidad de encontrar una solución de este tipo se reduce.[127] Por ello, en la presente investigación se utiliza la metaheurística GRASP o *Greedy Randomized Adaptive Search Procedure* para generar la población inicial. Este procedimiento consta de dos fases, construcción y búsqueda local, en la construcción se utiliza una función greedy de evaluación que permite crear una lista restringida de candidatos cuyo tamaño está definido por un valor determinado α, el cual, en el presente trabajo se establece aleatorio para cada iteración del GRASP con el fin de introducir diversidad en la población; esta lista denominada RCL contiene los mejores elementos candidatos de donde se seleccionarán los individuos por medio de una función lineal de sesgo propuesta por BRESINA[128].

En esta función se define:

- $r(\sigma)$: $rango\ del\ elemento\ \sigma = C_{max}; donde\ \sigma$: $operación$
- $Función\ de\ sesgo$: $bias(r) = {}^{1}/_{r(\sigma)}$
- $Probabilidad\ de\ ser\ seleccionado$: $\pi(\sigma) = \frac{bias(r)}{\sum_{\sigma' \in RCL}\left(bias\left(r(\sigma')\right)\right)}$

En la fase de búsqueda local el objetivo es mejorar las soluciones obtenidas en la etapa anterior, sin embargo teniendo en cuenta los resultados encontrados por AHUJA, ORLIN y TIWARI[129] no se implementará esta fase ya que conllevaría a un incremento en el tiempo computacional sin conducir a mejores resultados.

Como se mencionó anteriormente el procedimiento consta de una selección codiciosa y aleatorizada de las operaciones candidatas, en donde, el valor α

[127] Ibid.
[128] BRESINA, John L. Op. cit.
[129] AHUJA, Ravindra K.; ORLIN, James B.; TIWARI, Ashish. Op. cit.

determina qué tan aleatorizada o codiciosa es dicha selección. El proceso se describe de la siguiente manera:

- Se determinan las operaciones candidatas o factibles a ser programadas.
- Se evalúa el *makespan* de cada operación candidata según las operaciones que ya han sido programadas.
- Con estos valores y según la ecuación $[c^{min}, c^{min} + \alpha(c^{max} - c^{min})]$ se determina el intervalo entre el cual deben estar las candidatos para pertenecer a la RCL.
- Teniendo esta lista se selecciona aleatoriamente según la función de sesgo propuesta por BRESINA[130] que genera probabilidades para cada uno de los elementos de la RCL.
- El proceso se repite hasta programar todas las operaciones del problema.
- Formar el cromosoma de la solución construida según la codificación propuesta.
- Según el tamaño de la población inicial se determina el número de iteraciones.

Adicionalmente, con el propósito de introducir diversidad a la población inicial el 20% de esta se genera aleatoriamente.

A continuación, se muestra un ejemplo gráfico de la fase de construcción, en donde se realizan la programación de las dos primeras operaciones. Este ejemplo pertenece a un *Job Shop Scheculing Problem* de 3x3.

Notación general:

- ✓ $\sigma_k^j: k - \text{ésima operación del trabajo } j.$
- ✓ $\mathcal{M}_k^j: \mathcal{M}(\sigma_k^j)$ ó $\text{Máquina en la cual la operación } \sigma_k^j \text{ será procesada.}$

[130] BRESINA, John L. Op. cit.

- ✓ $p_k^j: p(\sigma_k^j)$ ó $tiempo\ de\ procesamietno\ de\ la\ operación\ \sigma_k^j$.
- ✓ $\mathcal{O}_c: Conjunto\ de\ operaciones\ candidatas\ a\ ser\ programadas.$
- ✓ $\mathcal{O}_s: Conjunto\ de\ operacines\ ya\ programadas.$
- ✓ $RCL: Lista\ de\ candidatos\ restringida.$
- ✓ $\alpha: Parametro\ que\ da\ el\ tamaño\ de\ RCL.$
- ✓ $h(\sigma): Función\ Greedy\ adapatativa \rightarrow\ \mathcal{C}_{max}\ para\ \mathcal{O} = \{\mathcal{O}_s \cup\ \sigma\}$
- ✓ $\bar{h}: Valor\ máximo\ de\ la\ función\ greedy\ adapatativa.$
- ✓ $\underline{h}: Valor\ minimo\ de\ la\ fumción\ greedy\ adaptativa.$
- ✓ $r(\sigma): Ranking\ de\ la\ función\ deBresina.$
- ✓ $\pi(\sigma): Probabilidad.$

Tabla 2. Tiempos de operación y precedencia de un JSSP 3x3.

	$\mathcal{M}1$	$\mathcal{M}2$	$\mathcal{M}3$	**Precedencia**
$\boldsymbol{\sigma_1^1}$	4			
$\boldsymbol{\sigma_2^1}$		5		σ_1^1
$\boldsymbol{\sigma_3^1}$			10	σ_2^1
$\boldsymbol{\sigma_1^2}$	15			
$\boldsymbol{\sigma_2^2}$		20		σ_1^2
$\boldsymbol{\sigma_3^2}$			10	σ_2^2
$\boldsymbol{\sigma_1^3}$		5		
$\boldsymbol{\sigma_2^3}$	10			σ_1^3
$\boldsymbol{\sigma_3^3}$			2	σ_2^3

Parámetros iniciales:

$$\alpha = 0{,}4$$

$$RCL = \{\sigma \in \mathcal{O} \mid \underline{h} \leq h(\sigma) \leq \underline{h} + \alpha(\bar{h} - \underline{h})$$

$$Función\ de\ sesgo\ para\ la\ RCL = {}^{1}/_{r(\sigma)}$$

Operación 1:

$$\mathcal{O}_c = \{\sigma_1^1, \sigma_1^2, \sigma_1^3\}$$

$$h(\sigma_1^1) = 4, \qquad h(\sigma_1^2) = 15, \qquad h(\sigma_1^3) = 5$$

$$\underline{h} = 4, \qquad \bar{h} = 15 \implies (\bar{h} - \underline{h}) = 11$$

$$4 \leq h(\sigma) \leq 8{,}4 \implies RCL = \{\sigma_1^1, \sigma_1^3\}$$

	$r(\sigma)$	$1/r(\sigma)$	$\pi(\sigma)$	$\pi(\sigma)$ *acum.*
σ_1^1	*4*	*0,25*	*0,556*	*0,556*
σ_1^3	*5*	*0,2*	*0,444*	*1*
	Total	*0,45*		

#Rnd= 0,155 $\implies$ *operación a programar* σ_1^1

$$\mathcal{O}_s = \{\sigma_1^1\} \rightarrow \mathcal{C}_{max} = 4$$

Operación 2:

$$\mathcal{O}_c = \{\sigma_2^1, \sigma_1^2, \sigma_1^3\}$$

$$h(\sigma_2^1) = 9, \qquad h(\sigma_1^2) = 19, \qquad h(\sigma_1^3) = 5$$

$$\underline{h} = 5, \qquad \bar{h} = 19 \implies (\bar{h} - \underline{h}) = 14$$

$$5 \leq h(\sigma) \leq 10{,}6 \implies RCL = \{\sigma_1^3\}$$

Operación a programar σ_1^3

$$\mathcal{O}_s = \{\sigma_1^1, \sigma_1^3\} \rightarrow \mathcal{C}_{max} = 5$$

9.3. SELECCIÓN

Como se mencionó anteriormente, por lo general, la selección de los individuos a reproducir se basa en dar una mayor probabilidad a los más aptos, sin embargo, en la presente investigación los cromosomas padres se seleccionan en dos grupos, de tal manera que se garantice que la generación tendrá el mismo número de individuos de la población inicial, un grupo denominado "mejores" que contiene, según el tamaño de la población entre 5 y 10 soluciones, lo que corresponde al 5% o 10%, las cuales son las mejores del grupo. Del segundo grupo denominado "restantes" se escogen los cromosomas padres faltantes aleatoriamente, es decir, todos los individuos de dicho grupo cuentan

con igual probabilidad de ser seleccionados. La selección aleatoria se realiza teniendo como base la investigación de AHUJA, ORLIN y TIWARI[131], en donde se obtuvieron mejores resultados cuando la selección no estaba sesgada hacia los individuos más adaptados. Además, al utilizar este tipo de selección proporcional a su valor *fitness*, se puede conducir a una convergencia "prematura" debido a que la población se completa con individuos similares.[132]

9.4. CROSSOVER

En la literatura se encuentran diferentes opiniones acerca del uso de operadores de cruce para el JSSP, por ejemplo, QI et al. [133] afirma que los operadores tradicionales no son efectivos en este tipo de problemas, por esto se han propuesto nuevos operadores que se ajustan a las representaciones planteadas, en el caso de la codificación de permutación existen operadores basados en crossover de dos puntos los cuales se enfocan en el orden o en la posición de los genes; entre los más nombrados se encuentra el PMX dada su eficiencia en la transmisión de esquemas de orden y a partir del mismo se han realizado modificaciones como el GPMX como es el caso de los operadores propuestos por Park et al.[134]

En esta investigación el operador de crossover que se utiliza corresponde al denominado «*Crossover 4*» por PARK et. al[135], el cual es una de las modificaciones mencionadas anteriormente y aquel que obtuvo la mayor cantidad de soluciones optimas en todas las instancias en la investigación realizada por el ya mencionado autor. Ese operador funciona de la siguiente manera.

---

[131] Ibid.

[132] WANG, Yuping, et al. Op. cit.

[133] QI, J. G.; BURNS, G. R.; HARRISON, D. K. The application of parallel multipopulation genetic algorithms to dynamic job-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 2000, vol. 16, no 8, p. 609-615. Citado por: KURDI, Mohamed. A new hybrid island model genetic algorithm for job shop scheduling problem. *Computers & Industrial Engineering*, 2015, vol. 88, p. 273-283.

[134] PARK, Byung Joo; CHOI, Hyung Rim; KIM, Hyun Soo. Op. cit.

[135] Ibid.

a. Se selecciona una sub-cadena aleatoriamente en el padre 1.
b. Se indican los elementos de la sub-cadena en el padre 2 según su índice de ocurrencia.
c. Se ubican los genes del padre dos uno por uno manteniendo el orden hasta llegar a la posición en que la sub-cadena inicia en el padre 1, se inserta la sub-cadena y se continua agregando los genes del padre 2.
d. Se eliminan los genes indicados en el numeral b dando como resultado el primer hijo.
e. Se repite el proceso intercambiando los padres.

La sub-cadena que se selecciona tiene un tamaño que se escoge aleatoriamente entre $^1/_4$, $^1/_3$ y $^1/_2$ del tamaño total del cromosoma, valor que se escoge aleatoriamente al igual que la posición donde dicha cadena empezará en el padre seleccionado.

En la figura 11 se representa gráficamente los pasos descritos anteriormente. Se observa la diferencia entre el hijo y los padres.

9.5. MUTACIÓN

Como se mencionó anteriormente la mutación busca diversificar la población y evitar la convergencia en un óptimo local. Para esta investigación se selecciona el operador de mutación *swap*, el cual consiste en seleccionar dos genes aleatorios del cromosoma e intercambiar sus posiciones, ajustándose a la representación adoptada selecciona para el intercambio solo operaciones de distintos trabajos.

Este proceso no se realiza en todos los hijos resultantes del cruce, se selecciona una probabilidad de mutación, que para este caso será menor o igual al 10%.

En la figura 12 se muestra de manera gráfica la mutación *swap* en un problema de tres trabajos y tres máquinas.

Figura 11. Representación gráfica del operador de cruce.

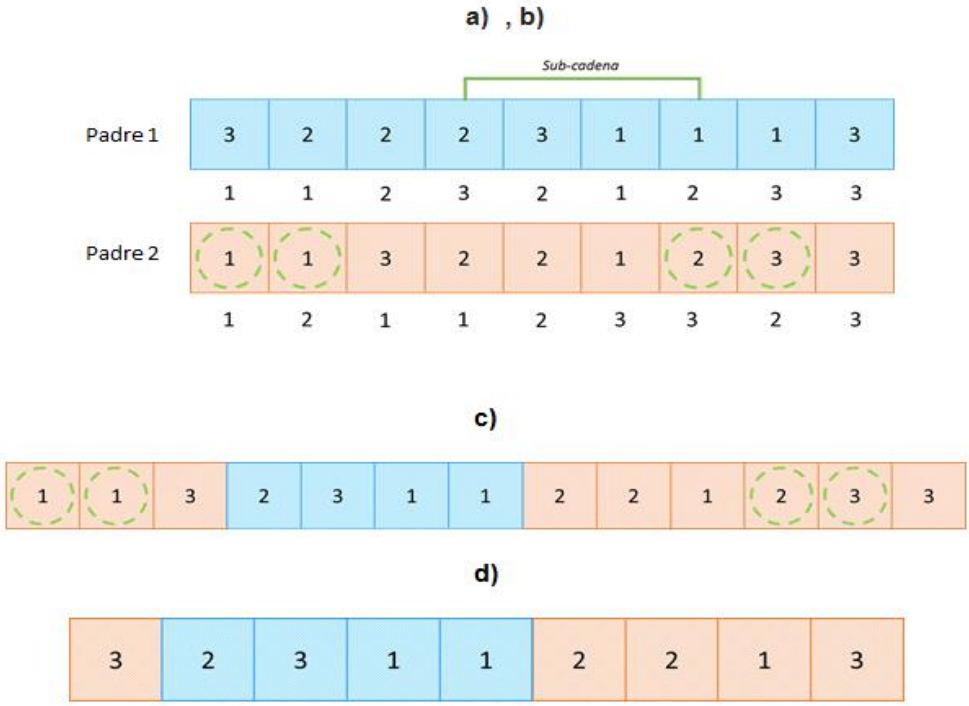
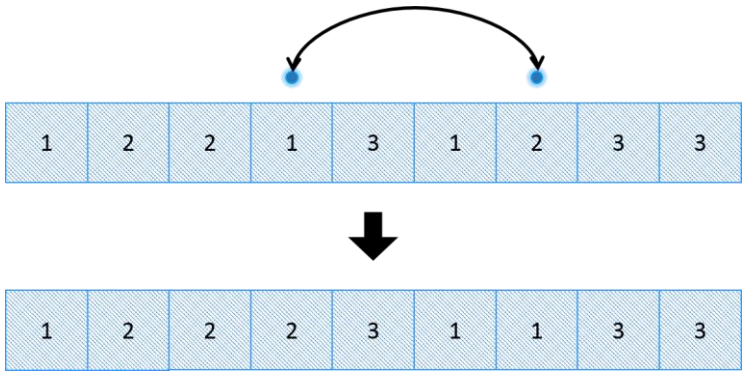


Figura 12. Representación gráfica de la mutación *Swap*.



Finalmente, el algoritmo diseñado se lleva al lenguaje de programación Matlab. Ver anexo A.

10. VALIDACIÓN DEL ALGORITMO

Para validar el algoritmo diseñado se utilizaron instancias que han sido previamente estudiadas en la literatura, entre las más destacadas se encuentran las de Fisher and Thompson (Ft) y Lawrence (La). Para la selección de estas se toma la decisión de clasificarlas por su tamaño, teniendo como base la clasificación realizada por CHASSAING, Maxime, et al.[136] quienes dividen las instancias en cuatro clases según el número de operaciones, estas son pequeñas, medianas, grandes y extra grandes. Para el presente trabajo se realizan dos grupos, en el grupo uno se encuentran las pequeñas y medianas las cuales abarcan desde Ft06 (6X6) a La15 (20X5) y en el grupo dos desde Ft10 y La16 (10X10) a Ft20 y La40 (15X15), las cuales son de tamaño mayor o igual a diez y algunas de estas se aproximan más a ser matrices cuadradas o lo son. Con la intención de probar el algoritmo en al menos una de cada tamaño existente, las instancias finalmente seleccionadas en la validación son: Ft06, La01, La02, La03, La04, La06, La09 y La11, del grupo uno y Ft10, Ft20, La16, La21, La28, La31 y La37 del grupo dos.

Para las corridas realizadas se variaron los parámetros número de generaciones (grupo uno 100 y 500; grupo dos 500 y 1000), tamaño de población (grupo uno 50 y 100; grupo dos 100 y 200) y porcentaje de mutación (en ambos casos 5% y 10%). Otros parámetros, como α, el cual se utiliza en el GRASP para la creación de la población inicial se mantiene aleatorio de 0-1 en cada iteración; de igual manera el tamaño de la sub-cadena para realizar el cruce en el algoritmo genético es aleatorio entre 1/2, 1/3 y 1/4 para cada cruce; además el porcentaje de selección se mantiene fijo en 50% y la probabilidad de cruce es siempre del 100% debido a que todos los padres seleccionados se cruzan.

Se realizaron las ocho combinaciones posibles variando los parámetros ya mencionados, para cada combinación se ejecutaron diez corridas en las instancias pertenecientes al grupo uno y tres a las del grupo; en las tablas se

[136] CHASSAING, Maxime, et al. Op cit.

muestra la comparación del *makespan* con el mejor conocido, la media y desviación estándar de las corridas realizadas en cada combinación.

$Generación$: Número de generaciones.

$Población$: Tamaño de la población.

$\%\ Mutación$: Porcentaje de mutación

MC: Makespan mejor conocido

MO: Makespan mejor observado

$Media$: Media del makespan en las corridas realizadas

s: Desviación estándar del makespan en las corridas realizadas

10.1. INSTANCIAS DEL GRUPO 1

Tabla 3. Validación en la instancia FT06

FT06 (6X6)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
100	50	5	55	55	56,4	1,26	0
500	50	5	55	55	55,0	0,00	0
100	100	5	55	55	55,5	1,08	0
500	100	5	55	55	55,0	0,00	0
100	50	10	55	55	55,7	1,16	0
500	50	10	55	55	55,0	0,00	0
100	100	10	55	55	55,3	0,95	0
500	100	10	55	55	55,0	0,00	0

Para la instancia Ft06 se observa que el valor de *makespan* mejor conocido es alcanzado en todas las combinaciones realizadas, sin embargo, en algunas de las corridas en donde el parámetro número de generaciones es 100 no se logró siempre este valor.

En el anexo B se muestra una asignación de tareas obtenida con el método de solución en esta instancia, representada en un diagrama de Gantt.

Tabla 4. Validación en la instancia LA01

LA01 (10X5)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
100	50	5	666	673	686,7	13,66	1,0511
500	50	5	666	666	666,5	1,58	0,0000
100	100	5	666	668	679,4	9,50	0,3003
500	100	5	666	666	666,6	1,90	0,0000
100	50	10	666	666	682,5	13,44	0,0000
500	50	10	666	666	666	0,00	0,0000
100	100	10	666	678	684,6	8,58	1,8018
500	100	10	666	666	666,6	1,90	0,0000

Para la instancia La01 se observa que el valor del *makespan* mejor conocido se obtiene en el 62.5% de las combinaciones, además las combinaciones que presentan mayor dispersión en los datos son las realizadas para el numero de generaciones igual a 100, es decir, que en estas las respuestas obtenidas del *makespan* variaron más.

Tabla 5. Validación en la instancia LA02

LA02 (10X5)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**S**	**%DIF**
100	50	5	655	694	723,5	14,68	5,9542
500	50	5	655	701	708,5	5,60	7,0229
100	100	5	655	699	711,4	7,90	6,7176
500	100	5	655	689	697,2	5,83	5,1908
100	50	10	655	693	718,9	11,93	5,8015
500	50	10	655	693	710,8	10,89	5,8015
100	100	10	655	701	714,2	11,50	7,0229
500	100	10	655	692	705,9	10,44	5,6489

Para la instancia La02 se puede observar que en ninguna combinación de las realizadas se alcanza el *makespan* mejor conocido, sin embargo, el valor que más se aproxima a este difiere en un 5.2% y se obtiene con la combinación de 500 generaciones, tamaño de población 100 y porcentaje de mutación 5%.

Tabla 6. Validación en la instancia LA03

LA03 (10X5)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
100	50	5	597	637	656	11,93	6,7002
500	50	5	597	636	648,3	5,77	6,5327
100	100	5	597	644	655	6,75	7,8727
500	100	5	597	629	642,5	5,91	5,3601
100	50	10	597	645	661	8,91	8,0402
500	50	10	597	633	648,9	8,61	6,0302
100	100	10	597	635	647,5	6,80	6,3652
500	100	10	597	638	644,5	4,50	6,8677

Para la instancia La03 no se logra alcanzar el mejor valor conocido de *makespan* y más cercano a este se encuentra a un 5.4% de diferencia, este es obtenido con la combinación de 500 generaciones, tamaño de población 100 y porcentaje de mutación 5%.

Tabla 7. Validación en la instancia LA04

LA04 (10X5)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
100	50	5	590	617	632	9,94	4,5763
500	50	5	590	611	620,5	6,92	3,5593
100	100	5	590	625	634,8	8,92	5,9322
500	100	5	590	619	624	3,33	4,9153
100	50	10	590	624	630,4	7,11	5,7627
500	50	10	590	612	620,1	6,71	3,7288
100	100	10	590	622	635,8	9,04	5,4237
500	100	10	590	618	622,7	4,60	4,7458

Para la instancia La04 se observa que tampoco es posible alcanzar el mejor *makespan* conocido y el más próximo a este se encuentra a un 3.56% con la combinación de 500 generaciones, tamaño de población 100 y porcentaje de mutación 5%

Tabla 8. Validación en la instancia LA06

LA06 (15X5)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
100	50	5	926	926	927,9	4,33	0,0000
500	50	5	926	926	926	0,00	0,0000
100	100	5	926	926	926,4	0,97	0,0000
500	100	5	926	926	926	0,00	0,0000
100	50	10	926	926	928,2	2,57	0,0000
500	50	10	926	926	926	0,00	0,0000
100	100	10	926	926	929,4	6,26	0,0000
500	100	10	926	926	926	0,00	0,0000

Para la instancia La06 se logra alcanzar el mejor *makespan* conocido en todas las combinaciones posibles, sin embargo, en las combinaciones en las cuales todas las corridas arrojaron este valor coinciden en que el número de generaciones utilizado es 500, puesto que para los otros casos si se observa un valor de dispersión en las corridas.

Tabla 9. Validación en la instancia LA09

LA09 (15X5)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
100	50	5	951	956	962,9	5,13	0,5258
500	50	5	951	951	951,1	0,32	0,0000
100	100	5	951	952	964,3	6,17	0,1052
500	100	5	951	951	951,4	1,26	0,0000
100	50	10	951	951	960,1	5,47	0,0000
500	50	10	951	951	951,5	1,58	0,0000
100	100	10	951	951	955,2	4,34	0,0000
500	100	10	951	951	951,8	1,48	0,0000

Para la instancia La09 se logra obtener el mejor *makespan* conocido en el 75% de las combinaciones y en los casos en los que no se coincide con este valor, la diferencia es mínima pues se observa que en ambos casos esta es menor al 1%.

Tabla 10. Validación en la instancia LA11

LA11 (15X5)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
100	50	5	1222	1227	1248,9	10,75	0,4092
500	50	5	1222	1222	1222,9	1,91	0,0000
100	100	5	1222	1230	1243,6	13,29	0,6547
500	100	5	1222	1222	1227,1	6,30	0,0000
100	50	10	1222	1236	1249,4	7,47	1,1457
500	50	10	1222	1222	1226,6	6,74	0,0000
100	100	10	1222	1230	1249,4	12,94	0,6547
500	100	10	1222	1222	1225,6	5,02	0,0000

Para la instancia La11 se observa que en el 50% de las combinaciones se logra obtener el mejor *makespan* conocido, en estas coincide que el número de generaciones es de 500; además, aunque las otras no alcancen este valor, la diferencia se encuentra por debajo del 1%.

En las ocho instancias seleccionadas de este grupo el algoritmo logra llegar al 62.5% de estas al mejor valor conocido, siendo este un buen indicador de la efectividad del algoritmo diseñado para las instancias de este tamaño, teniendo en cuenta que en las que no alcanzó este valor, la diferencia no sobrepasa el 5%. Además, se observa que en la mayoría de estas los mejores valores coinciden cuando el número de generaciones se encuentra en 500.

10.2. INSTANCIAS DEL GRUPO 2.

Tabla 11. Validación en la instancia FT10

FT10 (10X10)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
500	100	5	930	1081	1093,0	12,00	16,2366
1000	100	5	930	1051	1081,0	28,62	13,0108
500	200	5	930	1059	1071,0	18,25	13,8710
1000	200	5	930	1085	1088,0	2,65	16,6667
500	100	10	930	1071	1090,3	17,79	15,1613
1000	100	10	930	1083	1096,0	13,00	16,4516
500	200	10	930	1083	1089,3	7,09	16,4516
1000	200	10	930	1081	1093,0	12,00	16,2366

Para la instancia Ft10 se observa que en ninguna de las combinaciones se obtiene el *makespan* mejor conocido; además, la dispersión de cada combinación muestra que en cada corrida la variación en la respuesta es considerable en la mayoría de las combinaciones. El mejor *makespan* se obtuvo con un número de generaciones de 1000, tamaño de la población de 100 y porcentaje de mutación del 5%, y la diferencia con el mejor es de 13%.

Tabla 12. Validación en la instancia FT20

FT20 (20X5)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
500	100	5	1165	1279	1304,0	21,79	9,7854
1000	100	5	1165	1322	1343,3	18,58	13,4764
500	200	5	1165	1288	1294,3	5,69	10,5579
1000	200	5	1165	1273	1306,7	36,83	9,2704
500	100	10	1165	1304	1314,0	15,62	11,9313
1000	100	10	1165	1294	1309,3	13,32	11,0730
500	200	10	1165	1280	1296,0	16,52	9,8712
1000	200	10	1165	1279	1286,0	6,08	9,7854

Para la instancia Ft20 se observa que el mejor *makespan* conocido no se alcanza y la respuesta que más se aproxima se obtuvo con la combinación número de generaciones 1000, tamaño de la población 200 y 5% como porcentaje de mutación; con una diferencia del 9.27%. Además, se observa que la dispersión en todos los datos es considerable.

Tabla 13. Validación en la instancia LA16

LA16 (10X10)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
500	100	5	945	985	1002,0	15,39	4,2328
1000	100	5	945	993	1005,0	11,14	5,0794
500	200	5	945	1015	1017,7	3,06	7,4074
1000	200	5	945	999	1004,7	9,81	5,7143
500	100	10	945	1001	1010,0	8,54	5,9259
1000	100	10	945	1005	1012,3	6,35	6,3492
500	200	10	945	994	1012,7	16,20	5,1852
1000	200	10	945	998	1001,3	4,93	5,6085

Para la instancia La16 se puede observar que no se logra obtener el mejor *makespan* conocido; sin embargo, el valor más próximo se encuentra a un 4.23% con la combinación número de generaciones 500, tamaño de la población 100 y porcentaje de mutación del 5%

Tabla 14. Validación en la instancia LA21

LA21 (15X10)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
500	100	5	1046	1258	1293,7	32,96	20,2677
1000	100	5	1046	1260	1283,3	23,50	20,4589
500	200	5	1046	1252	1272,3	18,45	19,6941
1000	200	5	1046	1265	1276,0	14,18	20,9369
500	100	10	1046	1293	1303,0	11,79	23,6138
1000	100	10	1046	1279	1280,7	2,89	22,2753
500	200	10	1046	1277	1281,3	3,79	22,0841
1000	200	10	1046	1286	1291,7	9,81	22,9446

Para la instancia La21 se observa que no se alcanza el mejor *makespan* conocido y la mejor respuesta se encuentra a un 19.69% de este, y se obtuvo con la combinación número de generaciones 500, tamaño de la población 200 y porcentaje de mutación del 5%. Sin embargo, la diferencia frente al mejor conocido de todas las combinaciones es similar.

Tabla 15. Validación en la instancia LA28

LA28 (20X10)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
500	100	5	1216	1505	1527,7	21,59	23,7664
1000	100	5	1216	1474	1496,3	25,42	21,2171
500	200	5	1216	1509	1519,0	12,49	24,0954
1000	200	5	1216	1493	1511,7	16,17	22,7796
500	100	10	1216	1507	1529,7	22,03	23,9309
1000	100	10	1216	1533	1544,3	13,32	26,0691
500	200	10	1216	1481	1500,0	16,64	21,7928
1000	200	10	1216	1503	1523,0	17,32	23,6020

Para la instancia La28 se observa no se logra el mejor *makespan* conocido y la respuesta más cercana se encuentra a un 21.22% de diferencia, obtenida con la combinación número de generaciones 1000, tamaño de la población 100 y porcentaje del 5%; también se observa que las otras combinaciones tienen una diferencia similar.

Tabla 16. Validación de la instancia LA31

LA31 (30X10)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
500	100	5	1784	1988	2011,0	19,92	11,4350
1000	100	5	1784	1984	1994,7	15,14	11,2108
500	200	5	1784	2032	2044,0	10,82	13,9013
1000	200	5	1784	1996	2012,0	14,42	11,8834
500	100	10	1784	1991	2011,0	26,46	11,6031
1000	100	10	1784	1977	1991,0	13,53	10,8184
500	200	10	1784	1964	2000,3	43,82	10,0897
1000	200	10	1784	1986	1997,3	16,29	11,3229

Para la instancia La31 se puede observar que no se logra alcanzar el mejor valor conocido y la respuesta más próxima fue obtenida con la combinación número de generaciones 500, tamaño de la población 200 y porcentaje de mutación del 10%

Tabla 17. Validación en la instancia LA37

LA37 (1X15)							
Generación	**Población**	**%Mutación**	**MC**	**MO**	**Media**	**s**	**%DIF**
500	100	5	1397	1616	1653,7	32,81	15,6764
1000	100	5	1397	1665	1668,7	5,51	19,1840
500	200	5	1397	1594	1616,7	21,20	14,1016
1000	200	5	1397	1571	1585,7	16,17	12,4553
500	100	10	1397	1637	1646,0	11,53	17,1797
1000	100	10	1397	1588	1594,3	5,69	13,6722
500	200	10	1397	1606	1614,0	8,00	14,9606
1000	200	10	1397	1596	1603,0	9,64	14,2448

Para la instancia La37 se puede observar que no se alcanza el mejor valor conocido, y la solución observada más próxima está a un 12.45% y se obtuvo con la combinación número de generaciones 1000, tamaño de la población 200 y porcentaje de mutación del 5%.

De este grupo se pudo observar que de las siete instancias probadas, en ninguna de estas el algoritmo logró obtener el *makespan* mejor conocido, sin embargo, en algunas de estas no dista mucho de este valor, como es el caso de la La16, el cual es el caso en el que más se aproxima, junto a esta la Ft20, y La31 en donde difieren en un 10% o menos. Las otras cuatro instancias analizadas se encuentran entre un 10% y 21%, siendo la más lejana al mejor conocido la La28 con un 21,2%.

11. DISEÑO EXPERIMENTAL

Con el fin de analizar que parámetros tienen un efecto significativo en la variable respuesta *makespan* se realiza un diseño factorial 2^k, siendo tres los factores seleccionados, número de generaciones, tamaño de la población y porcentaje de mutación, los cuales son los parámetros de entrada variables en el algoritmo genético propuesto, se eligen estos factores como los posibles efectos principales debido a que estos mismos fueron utilizados en la validación y los demás parámetros se mantienen aleatorios con la intención de garantizar diversidad en los individuos.

Para cada una de las instancias seleccionadas en la validación del algoritmo, se realizó su respectivo diseño, estos fueron creados y analizados en el software estadístico Minitab 17. Dado que el diseño seleccionado es un 2^3 completo, todos los tratamientos posibles son los que se muestran en la tabla 18. Adicionalmente, el número de réplicas para las instancias del grupo uno fueron diez, mientras para las del grupo dos se utilizaron tres replicas. Los niveles utilizados para cada factor se muestran en la tabla 19.

Tabla 18. Tratamientos del diseño factorial 2^3 completo

	A (Generación)	**B (Población)**	**C (Mutación)**
1	-	-	-
2	+	-	-
3	-	+	-
4	+	+	-
5	-	-	+
6	+	-	+
7	-	+	+
8	+	+	+

$Generación$: Número de generaciones.

$Población$: Tamaño de la población.

$\%\ Mutación$: Porcentaje de mutación.

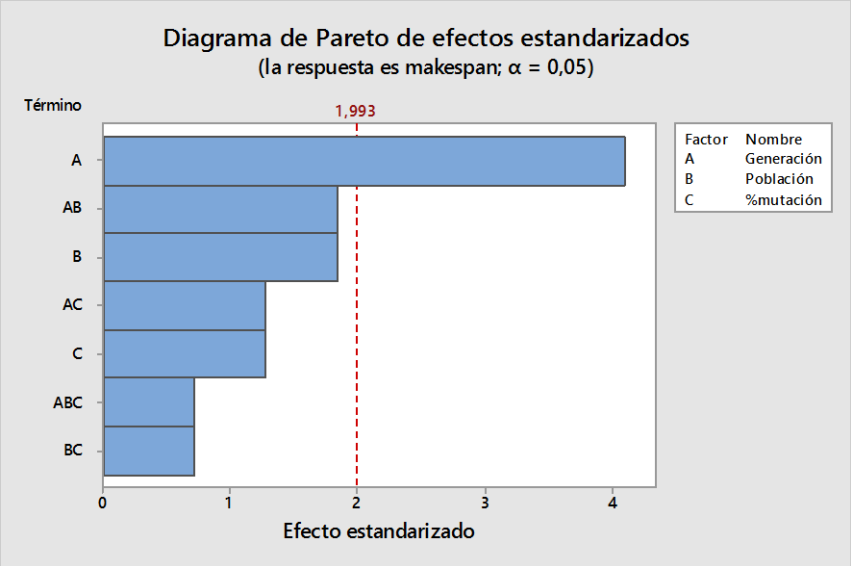
Tabla 19. Niveles de los factores para el diseño factorial 2^3 completo

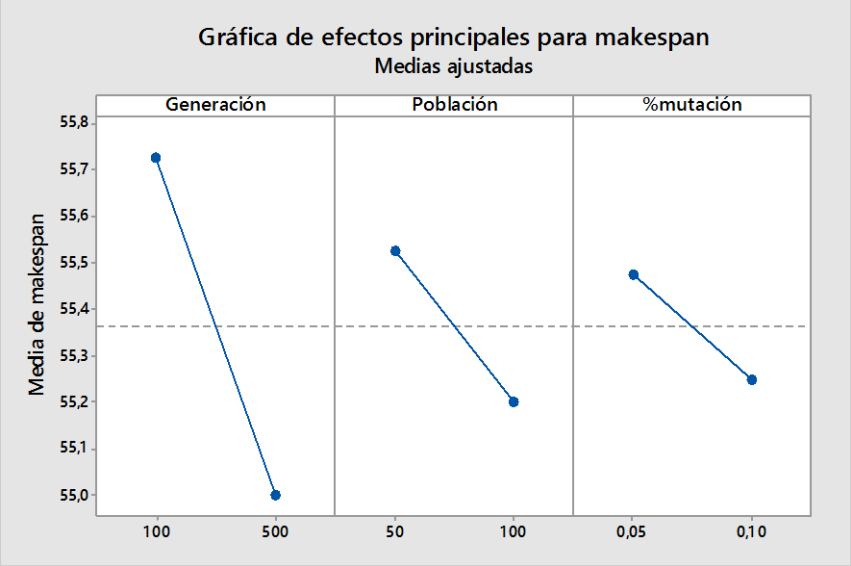
FACTORES	GRUPO 1		GRUPO 2	
	Nivel bajo	Nivel alto	Nivel bajo	Nivel alto
Generación	100	500	500	1000
Población	50	100	100	200
%mutación	5	10	5	10

El nivel de significancia considerado para el análisis de varianza es del 5%, los factores que evidencia un efecto significativo en la variable *makespan* son aquellos en los cuales su valor p es menor a 0.05; en el análisis de cada instancia se muestra el diagrama de Pareto de efectos estandarizados en el cual los factores o interacciones que sobrepasan la línea de referencia son los que tienen un efecto significativo; también se puede observar la gráfica de efectos principales, la cual muestra en qué nivel la variable respuesta se minimiza. Las tablas ANOVA se adjuntan en el anexo C, considerando que estos mismos resultados se resumen en las gráficas de cada instancia.

11.1. INSTANCIAS DEL GRUPO1

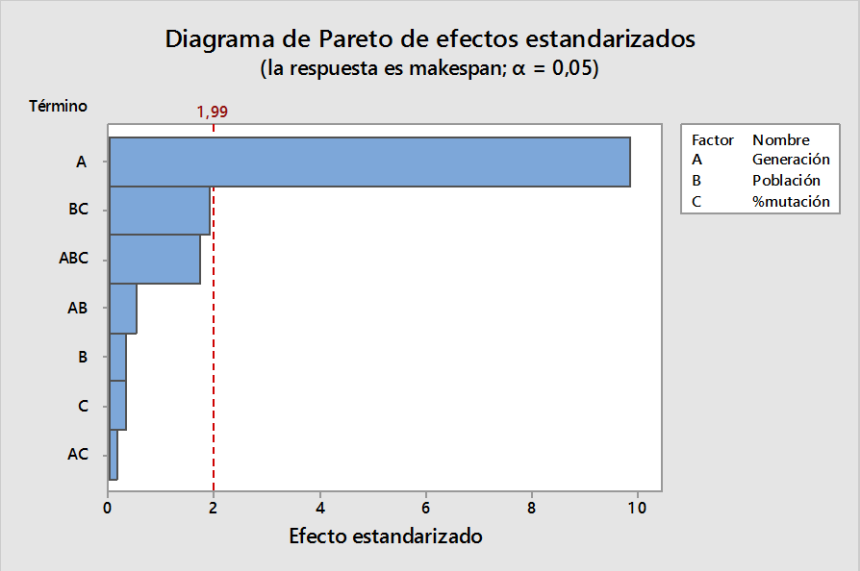
11.1.1. Análisis de varianza para la instancia FT06

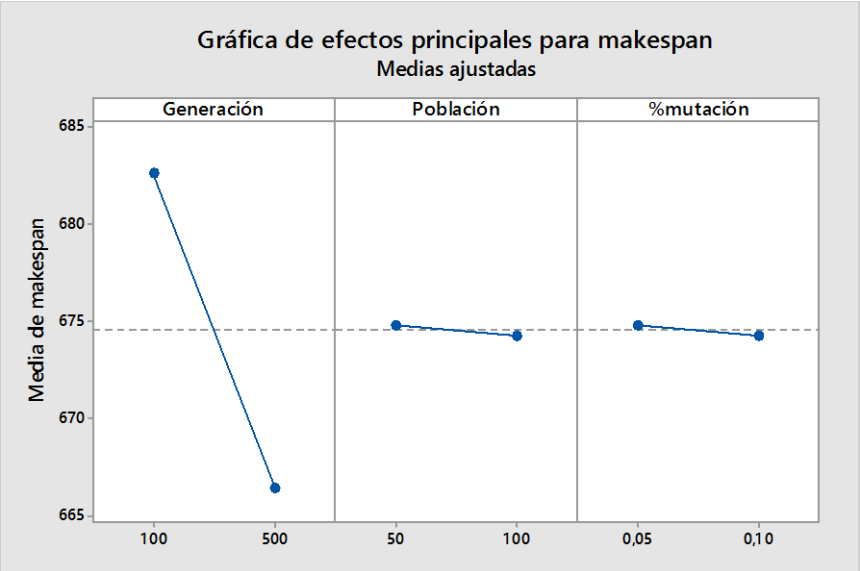




El factor que evidencia un efecto significativo en la variable respuesta *makespan* es el número de Generaciones. En la gráfica de efectos principales se observa que el nivel alto de número de generaciones arroja un valor menor del *makespan*. Los demás factores y las interacciones de estos no evidencian un efecto significativo.

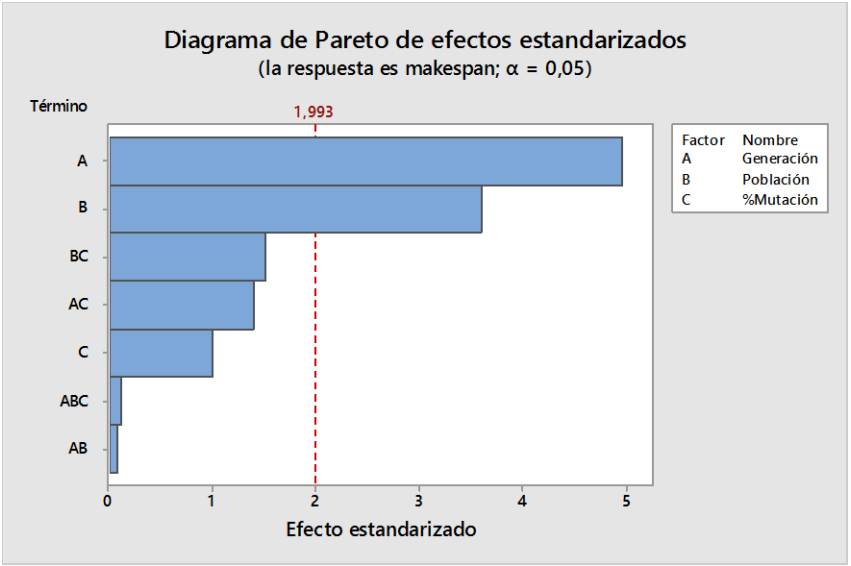
11.1.2. Análisis de varianza para la instancia LA01

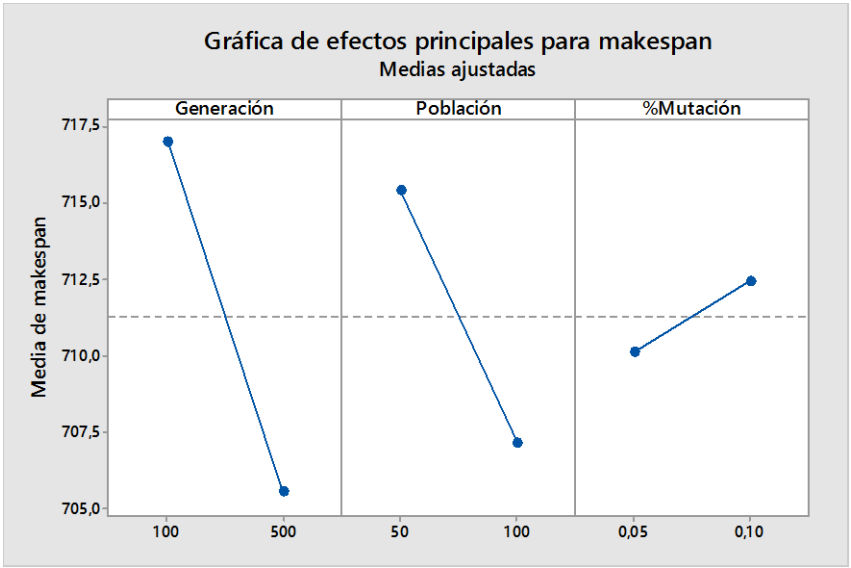




En esta instancia se puede observar que el factor con efecto significativo es el número de generaciones y que el nivel alto de este conduce a una disminución del *makespan*. Los demás factores y las interacciones de estos no evidencian un efecto significativo.

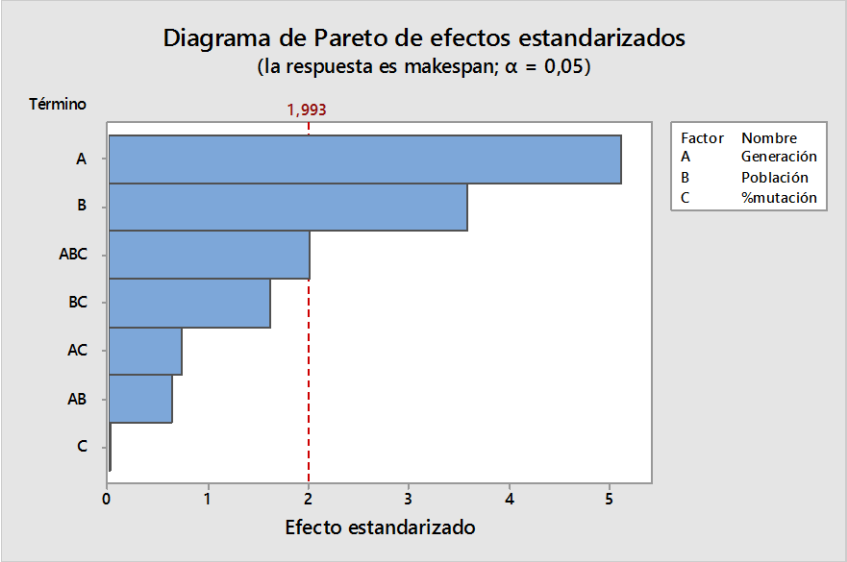
11.1.3. Análisis de varianza para la instancia LA02

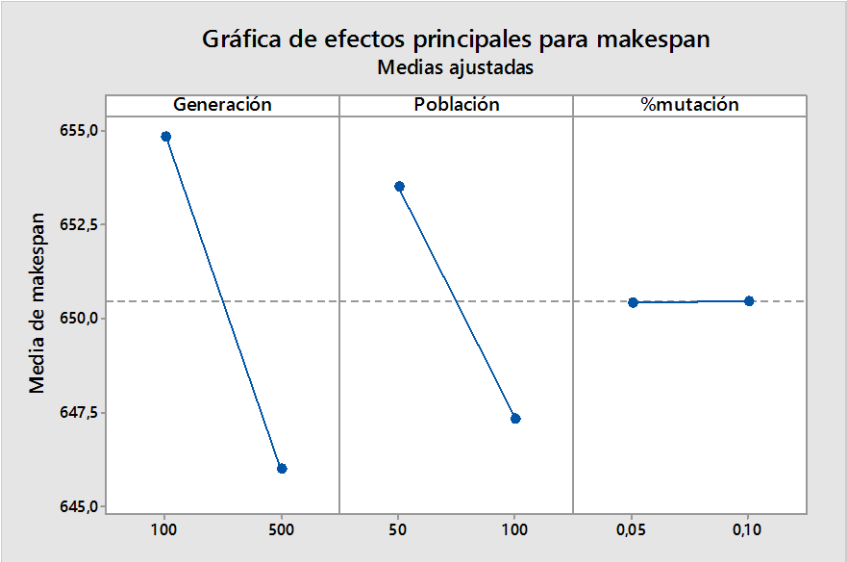




Para este caso los factores con efectos significativos son el número de generaciones y el tamaño de la población, los cuales en el nivel alto arrojaron respuestas de un *makespan* menor. Los demás factores y las interacciones de estos no evidencian un efecto significativo.

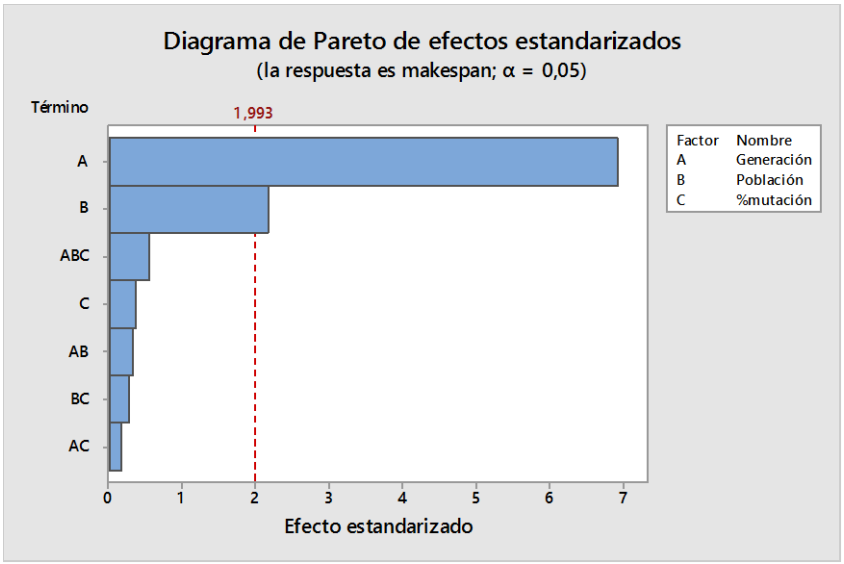
11.1.4. Análisis de varianza para la instancia LA03

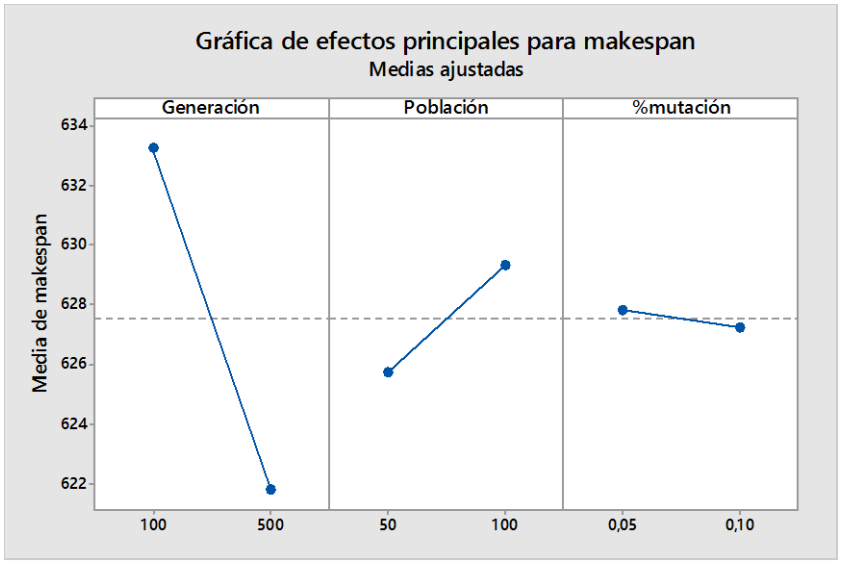




En esta instancia los factores con efectos significativos son el número de generaciones, el tamaño de la población, también la interacción de los tres factores analizados arroja un efecto significativo; los niveles altos de cada factor son los que dieron como resultado un menor valor del *makespan*.

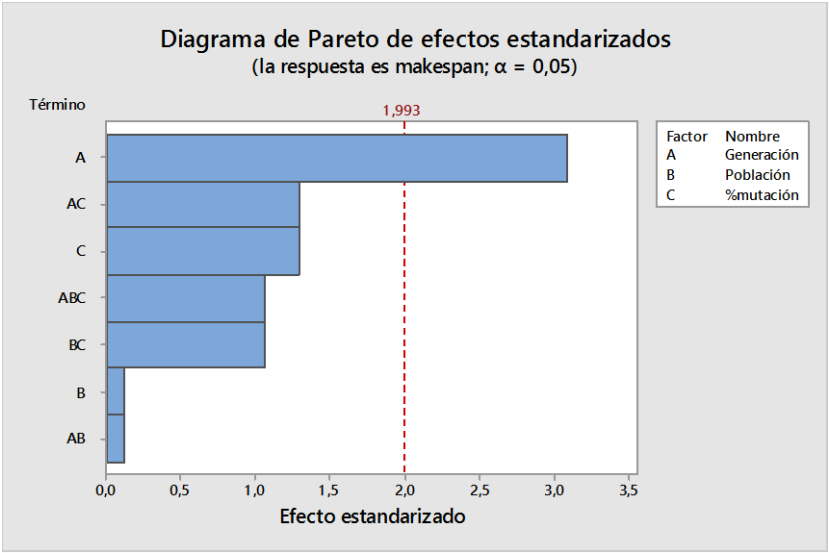
11.1.5. Análisis de varianza para la instancia LA04

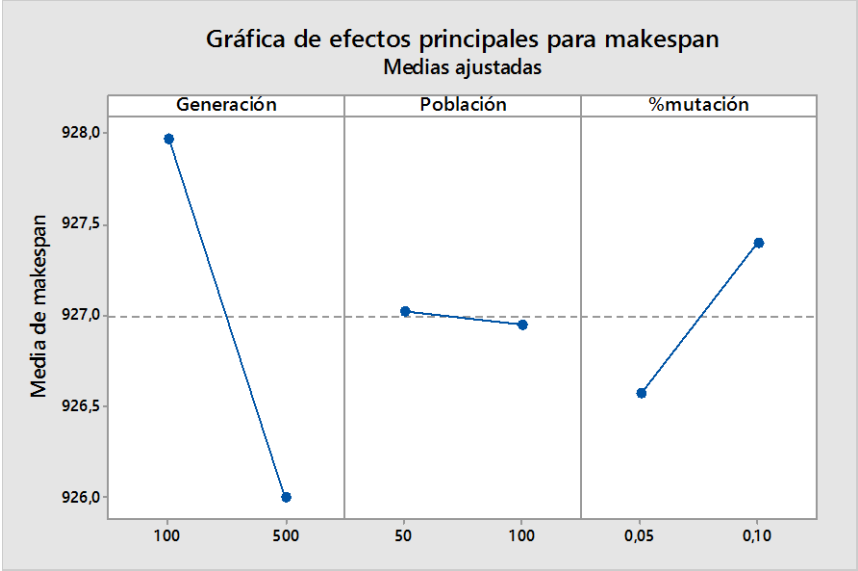




Para este caso los factores número de generaciones y tamaño de la población son los que evidencian efectos significativos sobre el valor del *makespan*, en el caso del número de generaciones el nivel alto disminuye la variable respuesta, mientras que en el tamaño de la población es el nivel bajo de este factor.

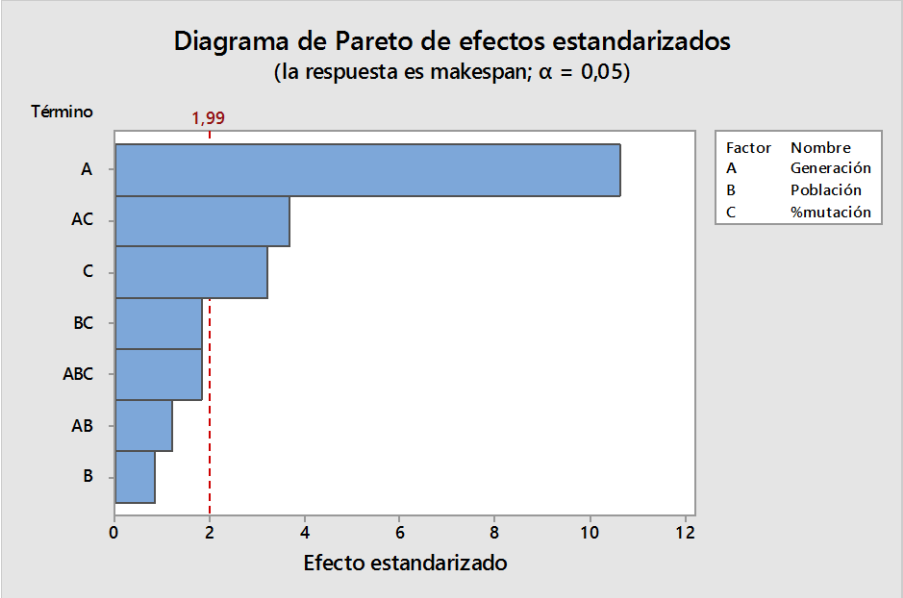
11.1.6. Análisis de varianza para la instancia LA06

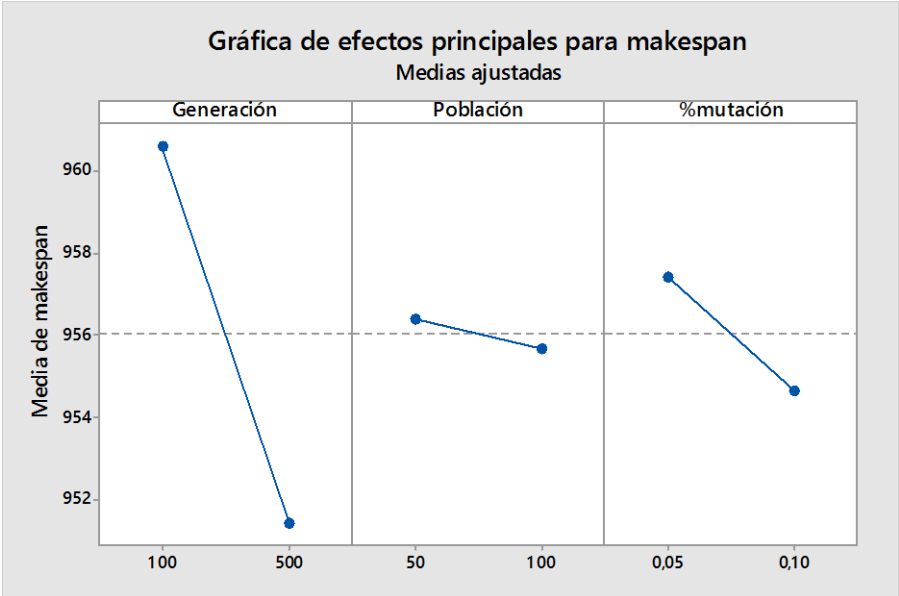




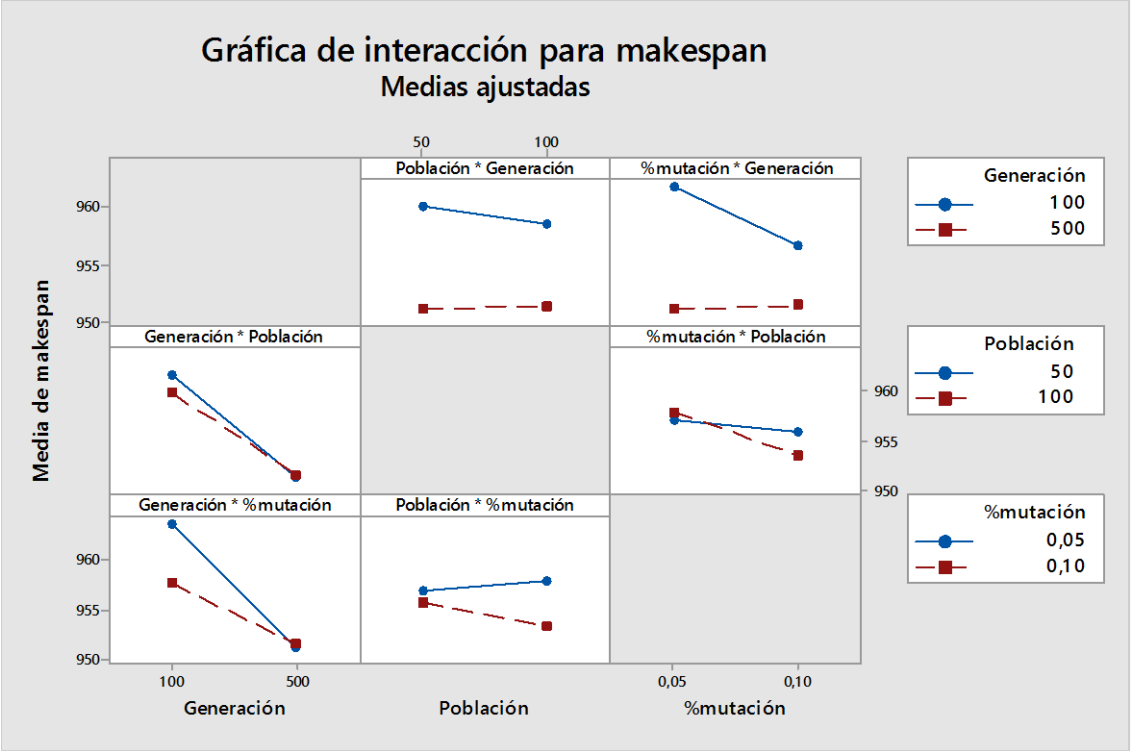
Para esta instancia el efecto significativo se evidencia en el factor número de generaciones, estando este en un nivel alto arroja menores valores del *makespan*. Los demás factores y las interacciones de estos no evidencian un efecto significativo.

11.1.7. Análisis de varianza para la instancia LA09



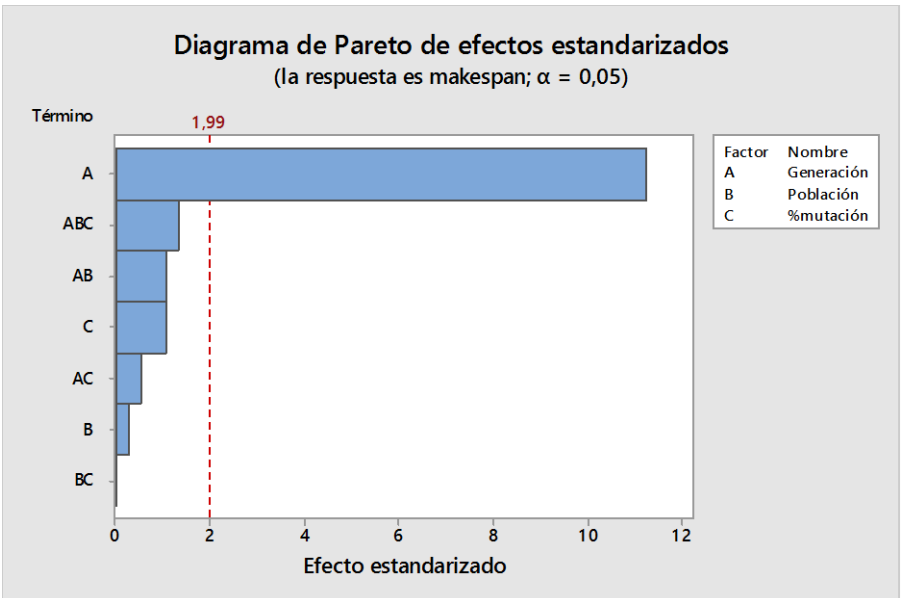


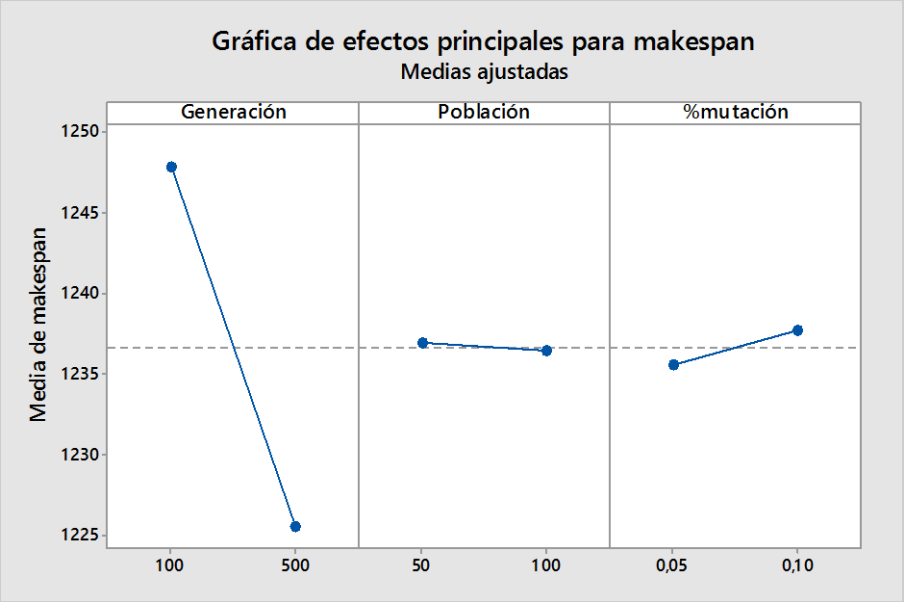
En esta instancia los factores, número de generaciones, porcentaje de mutación y la interacción de estos mismos evidencian efecto significativo sobre el *makespan*, en el nivel alto de cada uno se observaron las mejores respuestas.



En la gráfica de interacción de los factores se observa que a un nivel alto de generaciones la variación es mínima en el *makespan* al utilizar cualquiera de los dos niveles de mutación, sin embargo en el nivel alto de generaciones se disminuye el *makespan* con un porcentaje de mutación del 10%.

11.1.8. Análisis de varianza para la instancia LA11



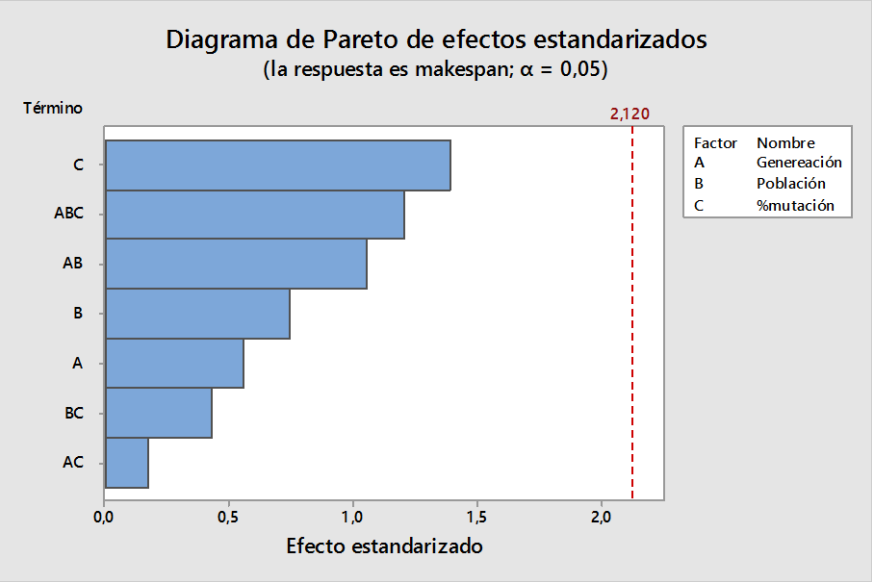


Para esta instancia el factor número de generaciones es el único que evidencia un efecto significativo en el *makespan*, en su nivel alto se observan los menores valores de este.

Se puede observar que el análisis de varianza para este grupo de instancias muestra que en el 100% de las instancias seleccionadas el factor número de generaciones tiene un efecto significativo en la variable *makespan* y para todos los casos con un nivel alto de este factor se obtiene un mejor valor del *makespan*, esto coincide con lo observado en las tablas de validación. Adicionalmente, el 37.5 % de las instancias evidencia tener efecto significativo en el factor tamaño de la población, en dos de los casos con un menor valor del *makespan* en el nivel alto de este factor y en uno de estos en el nivel bajo; este mismo porcentaje equivale a las instancias en las cuales el número de generaciones y el tamaño de la población tienen efectos significativos al mismo tiempo. Finalmente, solo una instancia, la cual representa el 12.5% muestra efectos significativos en interacciones, en este caso número de generaciones y porcentaje de mutación.

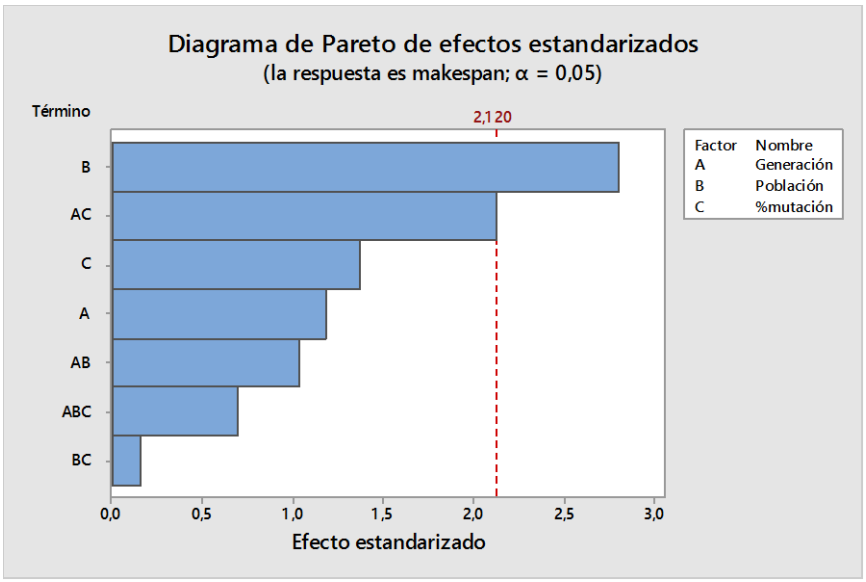
11.2. INSTANCIAS DEL GRUPO 2

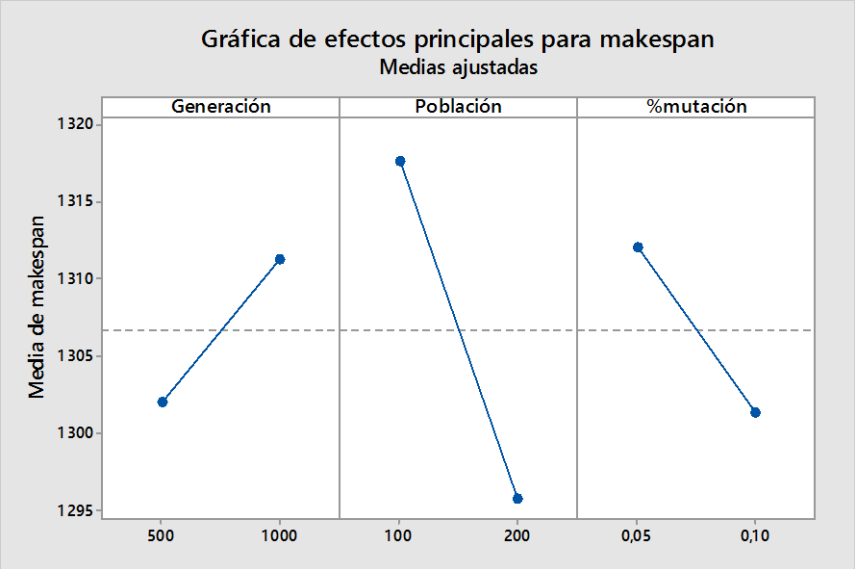
11.2.1. Análisis de varianza para la instancia FT10



Para esta instancia los factores e interacciones analizadas no evidencian un efecto significativo en el *makespan*.

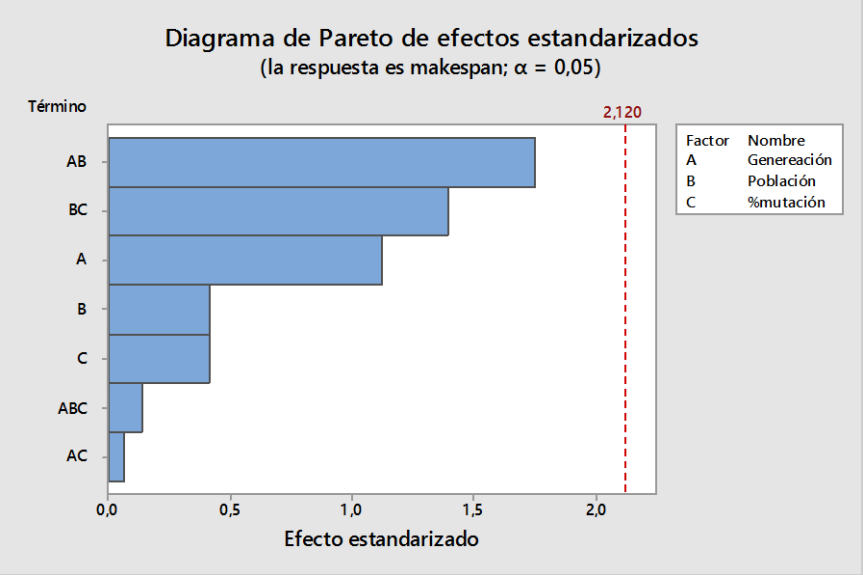
11.2.2. Análisis de varianza para la instancia FT20





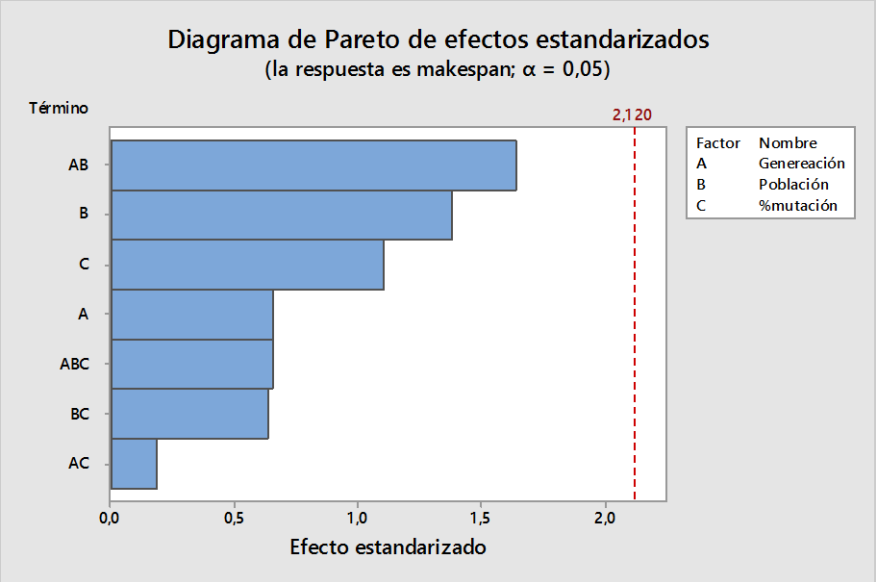
Para esta instancia se puede observar que el factor que evidencia un efecto significativo es el tamaño de la población, este en un nivel alto arroja mejores resultados. Los demás factores e interacciones no tienen un efecto significativo.

11.2.3. Análisis de varianza para la instancia LA16



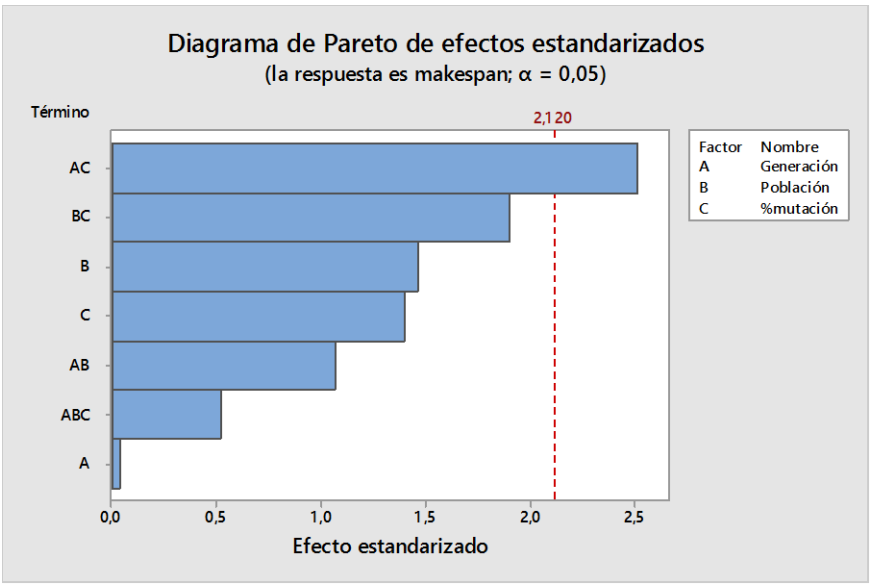
Para esta instancia ninguno de los factores considerados evidencia tener un efecto significativo sobre la variable *makespan*.

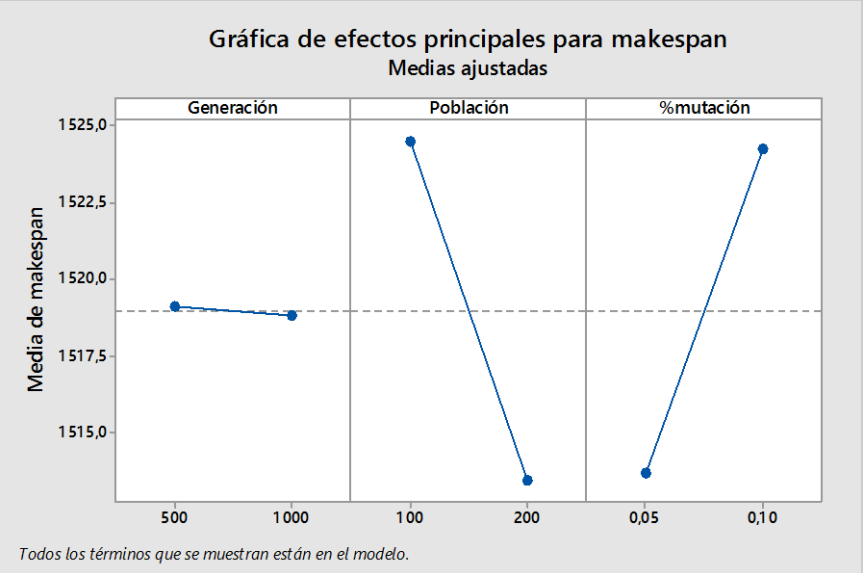
11.2.4. Análisis de varianza para la instancia LA21



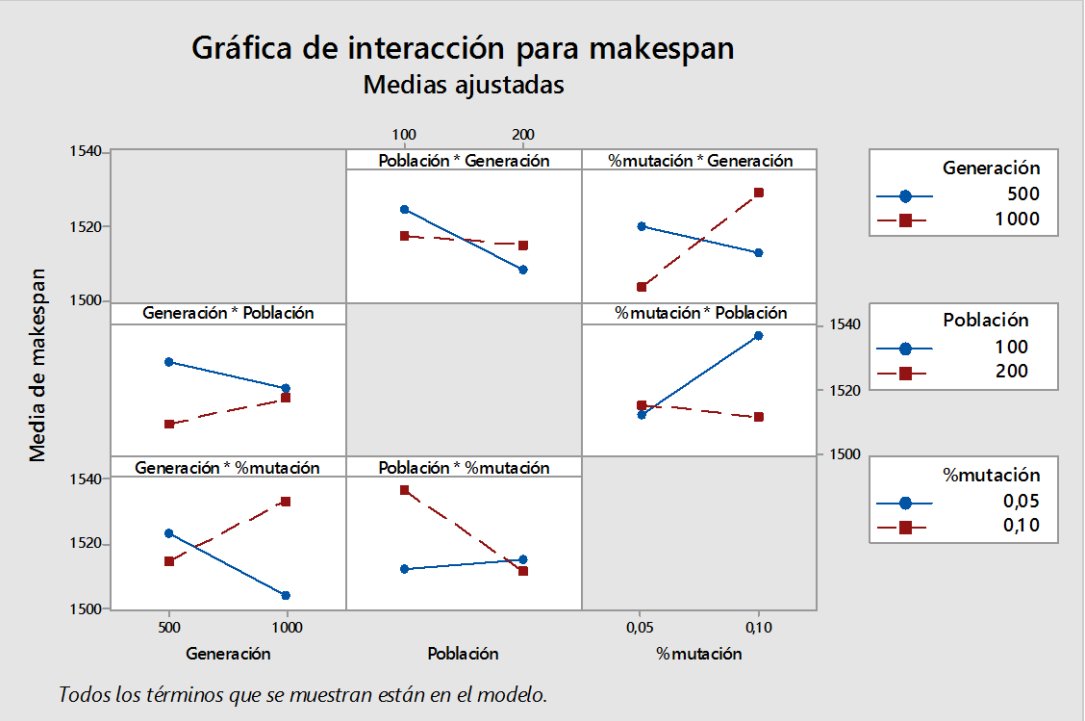
De igual forma para este caso los factores considerados no evidencian efecto significativo sobre la variable respuesta *makespan*.

11.2.5. Análisis de varianza para la instancia LA28



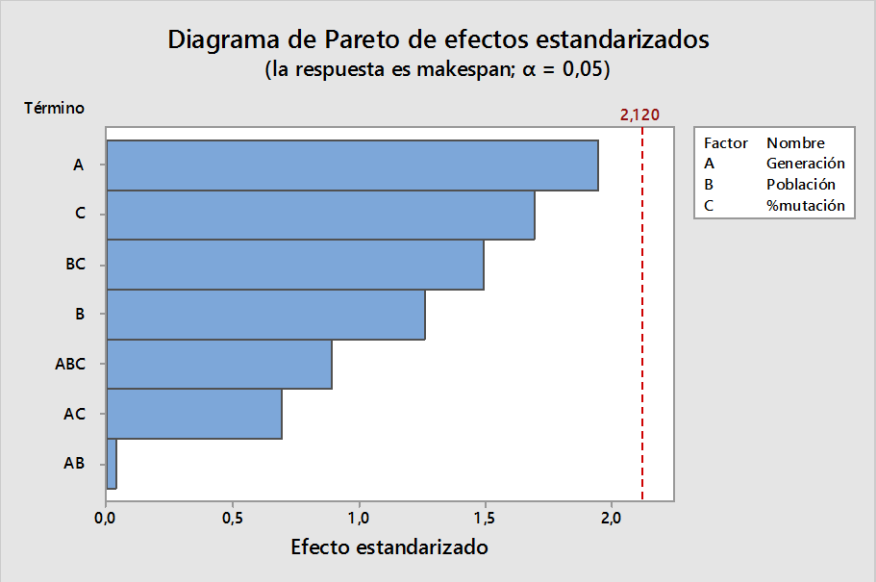


Para esta instancia el único efecto significativo se evidencia en la interacción número de generaciones*probabilidad de mutación.



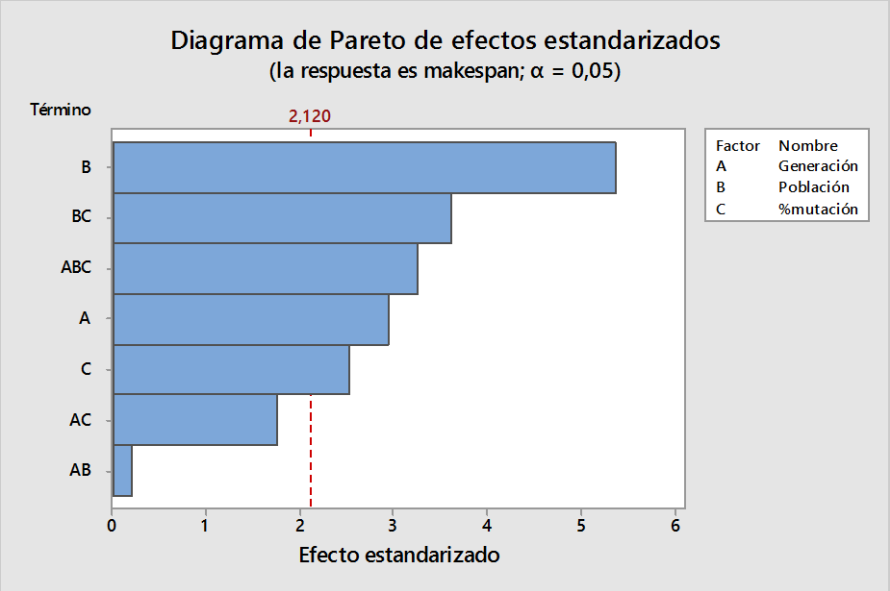
En la gráfica de interacción para *makespan* se observa que a un nivel alto de numero de generaciones y un nivel bajo de probabilidad de mutación se encuentran valores de *makespan* menores.

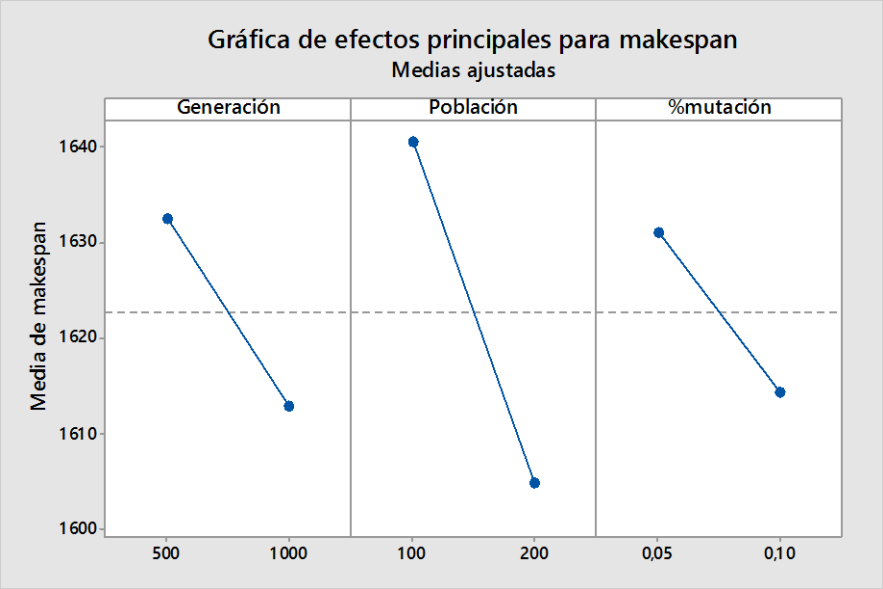
11.2.6. Análisis de varianza para la instancia LA31



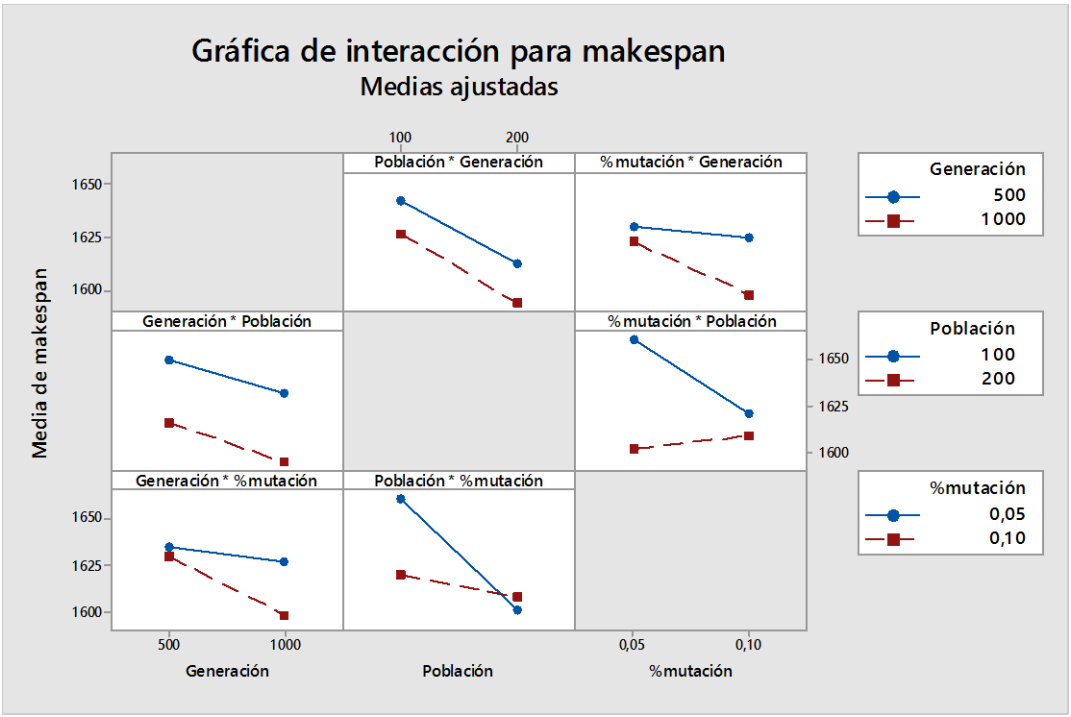
Para esta instancia se puede observar que ningún factor, ni interacción de estos tienen un efecto significativo sobre el *makespan*.

11.2.7. Análisis de varianza para la instancia LA37





En esta instancia se observa que los tres factores considerados, evidencian tener un efecto significativo sobre el *makespan*, estos tres en un nivel alto minimizan el valor del *makespan*; además la interacción tamaño de la población*porcentaje de mutación y la interacción de los tres factores también evidencian un efecto significativo.



En la gráfica de interacción para *makespan* se observa que los valores que minimizan el *makespan* es un tamaño de población de 200 con cualquiera de los

niveles de mutación o un tamaño de población de 100 con 10% como porcentaje de mutación.

Se puede observar que el análisis del diseño experimental de este grupo muestra que de las siete instancias analizadas el 57.14% no evidencia ningún efecto significativo en los factores seleccionados, es decir que con cualquier combinación la respuestas no variaran tanto y se podrían lograr respuestas similares en menores tiempos computacionales. Por otro lado el 28,57% tiene efectos significativos en el factor población, arrojando mejores resultados cuando se encuentra en el nivel alto. Adicionalmente, solo una instancia, la de mayor tamaño analizado, que corresponde al 14,26% muestra efecto significativo en los tres factores, cada uno de ellos en los niveles altos, y en las interacciones tamaño de la población*porcentaje de mutación y la interacción de los tres factores. En la tabla 20 se resumen los factores con efectos significativos para ambos grupos de instancias.

Tabla 20. Factores con efectos significativos para cada instancia

INSTANCIA	TAMAÑO	FACTORES CON EFECTO SIGNIFICATIVO
FT06	6X6	Generación
LA01	10X5	Generación
LA02	10X6	Generación - Población
LA03	10X7	Generación - Población - Interacción Generación*Población*Mutación
LA04	10X8	Generación - Población
LA06	15X5	Generación
LA09	15X5	Generación -Mutación - Interacción Generación*Mutación
LA11	20X5	Generación
FT10	10X10	No hay evidencia de efectos significativos
FT20	20X5	Población
LA16	10X10	No hay evidencia de efectos significativos
LA21	15X10	No hay evidencia de efectos significativos
LA28	20X10	Generación*Mutación
LA31	30X10	No hay evidencia de efectos significativos
LA37	15X15	Generación - Población - Mutación - Población*Mutación - Generación*Población*Mutación

12. RESULTADOS

En el capítulo once correspondiente a la validación se pudo constatar que el algoritmo diseñado es capaz de resolver correctamente el *Job Shop Scheduling problem*, es decir, obtener la programación de los trabajos teniendo en cuenta todas las restricciones que este problema involucra. En el capítulo siguiente se analizaron los factores considerados en el cual se identificaron efectos significativos de algunos factores que por medio de la observación no se detectaron en las tablas de validación y otros se constataron; utilizando el análisis obtenido por medio del diseño experimental se realizaron nuevas corridas del algoritmo ajustando los parámetros en busca de mejores resultados. Estos resultados se resumen en la tabla 21 donde se muestra, el mejor valor observado, la combinación de parámetros con la que se obtuvo, el tiempo computacional promedio de las corridas realizadas para cada instancia y la diferencia del algoritmo propuesto con el mejor valor conocido. El algoritmo se ejecutó en un computador HP ProDesk 600 G1 SSF con procesador Intel(R) Core(TM) i5-4570 CPU @ 320 GHz, memoria RAM de 8 GB y sistema operativo de 64 bits, en las instalaciones de la Escuela de Estudios Industriales y Empresariales UIS.

Tabla 21. **Resultados obtenidos del algoritmo propuesto.**

Instancia	Tamaño	Número de Generaciones	Tamaño de la población	Porcentaje de Mutación	MC	Algoritmo propuesto	Tiempo computacional [s]	%Dif
FT06	6X6	500	50	5	55	**55**	27,75	0,00
LA01	10X5	500	50	10	666	**666**	69,17	0,00
LA02	10X5	500	100	5	655	686	133,70	4,73
LA03	10X5	500	100	5	597	629	128,25	5,36
LA04	10X5	500	50	5	590	611	68,37	3,56
LA06	15X5	500	50	5	926	**926**	137,83	0,00
LA09	15X5	500	50	10	951	**951**	142,71	0,00
LA11	20X5	500	50	5	1222	**1222**	224,98	0,00
FT10	10X10	1000	100	5	930	1051	1053,83	13,01
FT20	20X5	1000	200	5	1165	1273	1702,91	9,27
LA16	10X10	500	100	5	945	985	361,01	4,23
LA21	15X10	500	200	5	1046	1252	1597,57	19,69
LA28	20X10	1000	100	5	1216	1474	2905,28	21,22
LA31	30X10	500	200	10	1784	1964	13047,69	10,09
LA37	15X15	1000	200	5	1397	1571	6650,17	12,46

Las tablas 22 y 23 que se muestran a continuación detallan la información acerca de los parámetros utilizados para las corridas que se realizaron.

Tabla 22. Selección de parámetros en el algoritmo propuesto.

PARÁMETROS	
A	Aleatorio en cada iteración
Tamaño de la sub-cadena para el operador cruce	Se selecciona aleatoriamente en cada iteración entre 1⁄2 , 1⁄3, 1⁄4
Selección	50% del tamaño de la población
Cruce	100% de los individuos seleccionados

Tabla 23. Selección de parámetros en el algoritmo propuesto.

Tamaño de la población	**Padres obtenidos con GRASP**	**Padres obtenidos aleatoriamente**	**Mejores individuos que pasan a la siguiente generación**
50	40	10	5
100	80	20	5
200	160	40	10

Finalmente, en la tabla 24 se compara el algoritmo propuesto frente a los resultados obtenidos por los proyectos antecedentes a este realizados en la Escuela de Estudios Industriales y Empresariales UIS (Grupo de investigación OPALO) y también frente a el algoritmo original de GRASP propuesto por BINATO et al[137].

[137] BINATO et al. Op cit.

Tabla 24. Comparación de resultados del algoritmo propuesto con otros algoritmos.

Instancia	Tamaño	Algoritmo propuesto	Algoritmo Memético AGULAR Y PEREZ	% Dif	Algoritmo Genético GOMEZ Y ORDUZ	%Dif	GRASP BINATO et. al	% Dif
FT06	6x6	55	55	0,0	55	0,0	55	0,0
LA01	10x5	666	666	0,0	666	0,0	666	0,0
LA02	10x5	686	655	4,7	-	-	655	4,7
LA03	10x5	629	597	5,4	-	-	604	4,1
LA04	10x5	611	590	3,6	-	-	590	3,6
LA06	15x5	926	926	0,0	926	0,0	926	0,0
LA09	15x5	951	951	0,0	951	0,0	951	0,0
LA11	20x5	1222	1222	0,0	-	-	1222	0,0
FT10	10x10	1051	937	12,2	978	7,5	938	12,0
FT20	20x5	1273	1182	7,7	1217	4,6	1169	8,9
LA16	10x10	985	946	4,1	-	-	946	4,1
LA21	15x10	1252	1081	15,8	1079	16,0	1091	14,8
LA28	20x10	1474	-	-	-	-	1293	14,0
LA31	30x10	1964	1784	10,1	-	-	1784	10,1
LA37	15x15	1571	-	-	-	-	1457	7,8

13. CONCLUSIONES

- El algoritmo diseñado es competitivo en las instancias pequeñas y medianas, dado que se logra alcanzar los mejores valores de makespan conocidos en el 62.5% de las instancias de este tamaño seleccionadas para la validación y el porcentaje restante la diferencia al mejor conocido no sobrepasa el 5%, todo esto en tiempos computaciones razonables, puesto que estos van desde tan solo segundos para la más pequeña a 3.7 minutos para aquella que toma más tiempo.

- Por otra parte, para las instancias de tamaño grande que fueron utilizadas, no se logra llegar al mejor valor conocido y se difiere de este valor considerablemente para la mayoría de estas. Sin embargo, es posible llegar a valores próximos en tiempos computacionales razonables; el valor más alto para este grupo de instancias fue de 3.6 horas, sin embargo, este puede reducirse notoriamente al utilizar menor número de generaciones, logrando respuestas similares dado ningún factor evidenció significancia, esto mismo aplicaría para las instancias de este grupo que no evidenciaron efectos significativos en los factores analizados.

- En el análisis del diseño experimental para las instancias del grupo uno se concluye que el efecto del factor número de generaciones es significativo en todas las instancias analizadas, al modificar este es posible llegar a mejores resultados; sin embargo, al realizar nuevamente corridas aumentando este valor, se incrementaba el tiempo computacional sin tener una mejora importante en el makespan, por ello los resultados finales se manejan con el nivel alto tomado para los diseños factoriales.

- En las instancias más grandes al no presentarse efectos significativos en más del 50% de las mismas, se considera que existen otros factores que influyen en la respuesta final. Dado que algunos parámetros son aleatorios, estos pueden ser una causal de dicho comportamiento, pues en este tipo de problemas se observa alto nivel de aleatoriedad.

- También se hace notorio que instancias que tienden a ser, o son, cuadradas tienen resultados más alejados al mejor conocido que aquellas cuya tendencia es rectangular, esto puede notarse en las tablas de resultados donde se especifica el tamaño de cada instancia, su resultado y su cercanía al BKS. En cuanto al comportamiento del código, cuyo tiempo clasificado por proceso se muestra en el anexo D, se puede concluir que tiene un mayor uso de recursos cuando se está realizando el diagrama de Gantt correspondiente al problema que se hace con el fin de encontrar el valor del makespan total y parcial; esto puede darse por los movimientos cíclicos que implican verificaciones en cada paso.

- Finalmente se puede decir que la combinación de las metaheurísticas como método de soluciones es una buena alternativa que ofrece versatilidad en el manejo de los problemas, es decir, que dependiendo el tamaño o complejidad del problema, se pueden ajustar parámetros para buscar mejorar los resultados y analizar los mismos para encontrar puntos potenciales de mejora de forma concreta.

14. RECOMENDACIONES

- Se destaca la importancia de que el Grupo de Investigación OPALO continúe realizando trabajos de este tipo, dado que permite que los estudiantes profundicen en temas vistos durante la carrera y como consecuencia de esto puedan apropiarse de los mismos; adicionalmente, permiten desarrollar habilidades investigativas, de programación en lenguajes como matlab y el manejo de un segundo idioma.

- Se recomienda revisar y posteriormente modificar etapas del algoritmo, como es la construcción de soluciones mediante un diagrama de Gantt, para mejorar el tiempo computacional y así mismo las soluciones obtenidas por este algoritmo.

- Se hace necesaria la disponibilidad de recursos computacionales más competentes que permitan analizar problemas de gran tamaño con mayor profundidad y así poder determinar si existen factores, dentro de los métodos de solución propuestos, que permitan un mejor desempeño en la función objetivo.

- Al estudiar el JSP en su esquema básico, este trabajo queda como preámbulo para variaciones que se realicen con el fin de encontrar soluciones, las cuales se ajusten en mayor medida a necesidades de sistemas productivos reales y que el grupo OPALO tenga la posibilidad de asesorar.

- Dado que el algoritmo presentado tiene alto grado de aleatoriedad, se recomienda para investigaciones futuras se enfoquen es este punto que puede ser clave en la calidad de la solución final, del mismo modo la revisión

y análisis de los operadores genéticos y de los otros parámetros utilizados puede ser un camino viable que ayude a mejorar dichas soluciones.

- Antes de realizar un trabajo de investigación, se hace necesario adquirir competencias propias de las herramientas a utilizar, esto ayudaría en gran medida a que futuros métodos de solución sean encontrados sin cometer errores en el uso de la herramienta que sesgarían los resultados

BIBLIOGRAFÍA

AGUILAR, Karin y PÉREZ, Yuleiny. Un algoritmo memético para la minimización del makespan en el problema del Job shop scheduling. Bucaramanga, 2012, 223p. Tesis (Ingeniería Industrial). Universidad Industrial de Santander. Facultad de ingeniería físicomecánicas. Escuela de estudios industriales y empresariales.

AHUJA, Ravindra K.; ORLIN, James B.; TIWARI, Ashish. A greedy genetic algorithm for the quadratic assignment problem. Computers & Operations Research, 2000, vol. 27, no 10, p. 917-934.

AIEX, Renata M., et al. GRASP with path relinking for three-index assignment. En: INFORMS Journal on Computing, 2005, vol. 17, no 2, p. 224-247.

AIEX, Renata M.; BINATO, Silvio; RESENDE, Mauricio GC. Parallel GRASP with path-relinking for job shop scheduling. En: Parallel Computing, 2003, vol. 29, no 4, p. 393-430.

ALVAREZ-VALDÉS, Ramón, et al. A tabu search algorithm for large-scale guillotine (un) constrained two-dimensional cutting problems. En: Computers & Operations Research, 2002, vol. 29, no 7, p. 925-947.

ALVAREZ-VALDÉS, Ramón; PARREÑO, Francisco; TAMARIT, José Manuel. Reactive GRASP for the strip-packing problem. En: Computers & Operations Research, 2008, vol. 35, no 4, p. 1065-1083.

AZIZI, Nader y ZOLFAGHARI, Saeed. Adaptive temperature control for simulated annealing: a comparative study. En: computers & operations research, 2004. Vol. 31, no. 14, p. 2439-2451.

BAHIENSE, Laura, et al. A mixed integer disjunctive model for transmission network expansion. En: Power Systems, IEEE Transactions, 2001, vol. 16, no 3, p. 560-565.

BARD, Jonathan F.; KONTORAVDIS, George; YU, Gang. A branch-and-cut procedure for the vehicle routing problem with time windows. En: Transportation Science, Mayo 2002, vol. 36, no 2, p. 250-269.

BERTSIMAS, Dimitris; SETHURAMAN, Jay. From fluid relaxations to practical algorithms for job shop scheduling: the makespan objective. En: Mathematical Programming, 2002, vol. 92, no 1, p. 61-102.

BIERWIRTH, Christian. A generalized permutation approach to job shop scheduling with genetic algorithms. Operations-Research-Spektrum, 1995, vol. 17, no 2-3, p. 87-92.

BIERWIRTH, Christian; MATTFELD, Dirk C.; KOPFER, Herbert. On permutation representations for scheduling problems. EnParallel Problem Solving from Nature—PPSN IV. Springer Berlin Heidelberg, 1996. p. 310-318.

BINATO, S., et al. A GRASP for job shop scheduling. En: Essays and surveys in metaheuristics. Springer US, 2002. p. 59-79.

BOUDIA, Mourad; LOULY, Mohamed Aly Ould; PRINS, Christian. A reactive GRASP and path relinking for a combined production–distribution problem. En: Computers & Operations Research, 2007, vol. 34, no 11, p. 3402-3419.

BŁAŻEWICZ, Jacek; DOMSCHKE, Wolfgang; PESCH, Erwin. The job shop scheduling problem: Conventional and new solution techniques. European journal of operational research, 1996, vol. 93, no 1, p. 1-33.

BLUM, Christian; ROLI, Andrea. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. En: ACM Computing Surveys (CSUR), 2003, vol. 35, no 3, p. 268-308.

BRESINA, John L. Heuristic-biased stochastic sampling. En: AAAI/IAAI, Vol. 1. 1996. p. 271-278. Citado por: BINATO, S., et al. A GRASP for job shop scheduling. En: Essays and surveys in metaheuristics. Springer US, 2002. p. 59-79.

BRUCKER, Peter; BRUCKER, P. Scheduling algorithms. Berlin: Springer, 2007

CHANDRASEKARAN, M., et al. Solving job shop scheduling problems using artificial immune system. En: The International Journal of Advanced Manufacturing Technology, 2006, vol. 31, no 5-6, p. 580-593.

CHASSAING, Maxime, et al. A GRASP× ELS approach for the job-shop with a web service paradigm packaging. En: Expert Systems with Applications, Febrero, 2014, vol. 41, no 2, p. 544-562.

CHENG, Runwei; GEN, Mitsuo; TSUJIMURA, Yasuhiro. A tutorialsurvey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. Computers & Industrial Engineering, 1999, vol. 36, no 2, p. 343-364.

CONWAY, R. W.; MAXWELL, W. L.; MILLER, L. W. Theory of scheduling. 1967. Theory of Scheduling, Palo Alto-London, 1967.

CUNQUERO, Rafael Martí. Algoritmos Heurísticos en Optimización Combinatoria.

DUARTE MUÑOZ, Abraham; PANTRIGO FERNÁNDEZ, J. J.; GALLEGO CARRILLO, M. Metaheurísticas. Madrid: Dykinson, 2007.

DUARTE, Abraham; MARTÍ, Rafael. Tabu search and GRASP for the maximum diversity problem. En: European Journal of Operational Research, 2007, vol. 178, no 1, p. 71-84.

FALKENAUER, Emanuel; BOUFFOUIX, S. A genetic algorithm for job shop. En Robotics and Automation, 1991. Proceedings, 1991 IEEE International Conference on. IEEE, 1991. p. 824-829.

FEO, Thomas A.; RESENDE, Mauricio GC; SMITH, Stuart H. A greedy randomized adaptive search procedure for maximum independent set. En: Operations Research, 1994, vol. 42, no 5, p. 860-878. Citado por: DUARTE, Op cit.

FEO, Thomas A.; RESENDE, Mauricio GC. Greedy randomized adaptive search procedures. En: Journal of global optimization, 1995, vol. 6, no 2, p. 109-133.

FERNANDES, Susana; LOURENÇO, Helena R. A GRASP and branch-and-bound metaheuristic for the job-shop scheduling. En: Evolutionary Computation in Combinatorial Optimization. Springer Berlin Heidelberg, 2007. p. 60-71.

FESTA, Paola; RESENDE, Mauricio GC. Hybrid GRASP heuristics. En: Foundations of Computational Intelligence Volume 3. Springer Berlin Heidelberg, 2009. p. 75-100.

GAO, Liang, et al. An efficient memetic algorithm for solving the job shop scheduling problem. Computers & Industrial Engineering, 2011, vol. 60, no 4, p. 699-705.

GARCÍA, José Francisco Chicano. Metaheurísticas e ingeniería del software. 2007. Tesis Doctoral. Universidad de Málaga.

GAWIEJNOWICZ, Stanislaw. Basics of the scheduling theory. En Time-Dependent Scheduling. Springer Berlin Heidelberg, 2008. p. 35-46.

GE, Hong-Wei, et al. An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling. En: Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 2008, vol. 38, no 2, p. 358-368.

GÓMEZ, Juan y ORDUZ, Edwin. Minimización del makespan en el problema de Job Shop Flexible con restricciones de transporte utilizando algoritmo genético. Bucaramanga, 2015, 126p. Tesis (Ingeniería Industrial). Universidad Industrial de Santander. Facultad de ingeniería físicomecánicas. Escuela de estudios industriales y empresariales.

GONÇALVES, Jose Fernando; DE MAGALHÃES MENDES, Jorge JoséGC. A hybrid genetic algorithm for the job shop scheduling problem. European journal of operational research, 2005, vol. 167, no 1, p. 77-95.

GUPTA, Skylab R.; SMITH, Jeffrey S. Algorithms for single machine total tardiness scheduling with sequence dependent setups. En: European Journal of Operational Research, 2006, vol. 175, no 2, p. 722-739.

HERRMANN, Jeffrey W. A history of production scheduling. En Handbook of Production Scheduling. Springer US, 2006. p. 1-22.

JAIN, Anant Singh; MEERAN, Sheik. A state-of-the-art review of job-shop scheduling techniques. Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998.

JAIN, Anant Singh; MEERAN, Sheik. Deterministic job-shop scheduling: Past, present and future. En: European journal of operational research, 1999, vol. 113, no 2, p. 390-434.

JIA, Wenyou; JIANG, Zhibin; LI, You. Scheduling to minimize the makespan in large-piece one-of-a-kind production with machine availability constraints. En: Expert Systems with Applications, 2015, vol. 42, no 23, p. 9174-9182.

JONG, Wen-Ren; LAI, Po-Jung. The navigation process of mould-manufacturing scheduling optimisation by applying genetic algorithm. En: International Journal of Computer Integrated Manufacturing, 2014, no ahead-of-print, p. 1-19.

LEI, Deming; WU, Zhiming. Crowding-measure-based multiobjective evolutionary algorithm for job shop scheduling. En: The International Journal of Advanced Manufacturing Technology, 2006, vol. 30, no 1-2, p. 112-117.

LI, Ye; CHEN, Yan. A genetic algorithm for job-shop scheduling. Journal of software, 2010, vol. 5, no 3, p. 269-274.

LIAN, Zhigang; JIAO, Bin. Y GU, Xingsheng. A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. En: applied mathematics and computation, 2006. Vol. 183, no. 2, p. 1008-1017.

LIU, Shi Qiang; ONG, Hoon Liong; NG, Kien Ming. A fast tabu search algorithm for the group shop scheduling problem. Advances in Engineering Software, 2005, vol. 36, no 8, p. 533-539.

MARINAKIS, Yannis; MARINAKI, Magdalene. A hybrid genetic–Particle Swarm Optimization Algorithm for the vehicle routing problem. En: Expert Systems with Applications, 2010, vol. 37, no 2, p. 1446-1455.

MARINAKIS, Yannis; MARINAKI, Magdalene. A hybrid multi-swarm particle swarm optimization algorithm for the probabilistic traveling salesman problem. En: Computers & Operations Research, 2010, vol. 37, no 3, p. 432-442.

MARINAKIS, Yannis; MARINAKI, Magdalene; DOUNIAS, Georgios. A hybrid particle swarm optimization algorithm for the vehicle routing problem. En: Engineering Applications of Artificial Intelligence, 2010, vol. 23, no 4, p. 463-472.

MARINAKIS, Yannis; MIGDALAS, Athanasios; PARDALOS, Panos M. Expanding neighborhood GRASP for the traveling salesman problem. En: Computational Optimization and Applications, 2005, vol. 32, no 3, p. 231-257.

MARINAKIS, Yannis; MARINAKI, Magdalene; DOUNIAS, Georgios. Honey bees mating optimization algorithm for the Euclidean traveling salesman problem. En: Information Sciences, 2011, vol. 181, no 20, p. 4684-4698.

MEISEL, José DaviD; PRADO, Liliana Katherine. Un algoritmo genético híbrido y un enfriamiento simulado para solucionar el problema de programación de pedidos job shop. Revista EIA, 2010, no 13, p. 31-51.

MENCÍA, Raúl, et al. Memetic algorithms for the job shop scheduling problem with operators. En: Applied Soft Computing, 2015, vol. 34, p. 94-105.

MENCÍA, Carlos, et al. Solving the job shop scheduling problem with operators by depth-first heuristic search enhanced with global pruning rules. En: AI Communications, 2015, vol. 28, no 2, p. 365-381.

MUTH, John F.; THOMPSON, Gerald Luther (ed.). Industrial scheduling. Prentice-Hall, 1963. Citado por: Ibid.

PAPADIMITRIOU, Christos H. Computational complexity. John Wiley and Sons Ltd., 2003.

PARK, Byung Joo; CHOI, Hyung Rim; KIM, Hyun Soo. A hybrid genetic algorithm for the job shop scheduling problems. Computers & industrial engineering, 2003, vol. 45, no 4, p. 597-613.

PINANA, Estefanıa, et al. GRASP and path relinking for the matrix bandwidth minimization. En: European Journal of Operational Research, 2004, vol. 153, no 1, p. 200-210.

PINEDO, Michael L. Scheduling: theory, algorithms, and systems. Springer Science & Business Media, 2012.

RESENDE, Mauricio GC; RIBEIRO, Celso C. A GRASP with path-relinking for private virtual circuit routing. En: Networks, 2003, vol. 41, no 2, p. 104-114.

RESENDE, Mauricio GC, et al. GRASP and path relinking for the max–min diversity problem. En: Computers & Operations Research, 2010, vol. 37, no 3, p. 498-508.

RESENDE, Mauricio GC; VELARDE, Jose Luis González. GRASP: Procedimientos de búsquedas miopes aleatorizados y adaptativos. En: Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 2003, vol. 7, no 19, p. 61-76.

RESENDE, Mauricio GC, et al. Handbook of Metaheuristics. Editado por Glover, F. e Kochenberger, G., Capítulo Greedy Randomized Adaptive Search Procedures, Ed. Kluwer Academic Publishers, pg, 2003, p. 219-249.

RIBEIRO, Celso; UCHOA, Eduardo; WERNECK, Renato F. A hybrid GRASP with perturbations for the Steiner problem in graphs. En: informs journal on computing, 2002. Vol. 14, no. 3, p. 228-246.

SARMIENTO, Cindy Johanna. Método particle swarm optimization (PSO-Enjambre de partículas) aplicado al problema de múltiples objetivos del Job Shop Scheduling (JSP) o secuenciamiento de máquinas. Bucaramanga, 2012, 134p. Tesis (ingeniería industrial). Universidad industrial de Santander. Facultad de ingeniería físicomecánicas. Escuela de estudios industriales y empresariales.

SHA, D. Y.; LIN, Hsing-Hung. A multi-objective PSO for job-shop scheduling problems. En: Expert Systems with Applications, 2010, vol. 37, no 2, p. 1065-1070.

SIERRA, María R.; MENCÍA, Carlos; VARELA, Ramiro. New schedule generation schemes for the job-shop problem with operators. En: Journal of Intelligent Manufacturing, 2013, vol. 26, no 3, p. 511-525.

SIPSER, Michael. Introduction to the Theory of Computation. Cengage Learning, 2012.

SURESH, R. K.; MOHANASUNDARAM, K. M. Pareto archived simulated annealing for job shop scheduling with multiple objectives. En: The International Journal of Advanced Manufacturing Technology, 2006, vol. 29, no 1, p. 184-196.

TANG, Dunbing; DAI, Min. Energy-efficient approach to minimizing the energy consumption in an extended job-shop scheduling problem. En: Chinese Journal of Mechanical Engineering, 2015, vol. 28, no 5, p. 1048-1055.

TASGETIREN, M. Fatih; PAN, Quan-Ke; LIANG, Yun-Chia. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. En: Computers & Operations Research, 2009, vol. 36, no 6, p. 1900-1915.

WANG, Yuping, et al. A new hybrid genetic algorithm for job shop scheduling problem. Computers & Operations Research, 2012, vol. 39, no 10, p. 2291-2299.

WENQI, Huang; AIHUA, Yin. An improved shifting bottleneck procedure for the job shop scheduling problem. En: Computers & Operations Research, 2004, vol. 31, no 12, p. 2093-2110.

XIA, Wei-jun; WU, Zhi-ming. A hybrid particle swarm optimization approach for the job-shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 2006, vol. 29, no 3-4, p. 360-366.

ZHANG, et al. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. En: computers & operations research, 2007. Vol. 34, no. 11, p. 3229-3242.

ZHANG, et al. A very fast TS/SA algorithm for the job shop scheduling problem. En: computers & operations research, 2008. Vol. 35, no. 1, p. 282-294.

ZHOU, Hong; FENG, Yuncheng; HAN, Limin. The hybrid heuristic genetic algorithm for job shop scheduling. Computers & Industrial Engineering, 2001, vol. 40, no 3, p.191- 200.

ANEXOS

ANEXO A. CÓDIGO EN MATLAB

Función GRASP:

Crea la matriz de individuos que representan la mayor parte de la población inicial construyéndola a partir de la meta-heurística *Greedy Randomized Adaptive Search Procedure.* Esta matriz se compone de $(n * m) + 1$ columnas y filas determinadas por el tamaño deseado de individuos, la última columna mantiene el valor del *makespan* correspondiente a la representación de la solución que hay en la misma fila.

Entradas:

- *matriz:* Matriz del problema que se desea solucionar correspondiente a las instancias del *benchmark*.
- *tamano_poblacion_grasp:* número de individuos de la población inicial creados mediante la meta-heurística.

Salidas:

- *poblacion_grasp:* Matriz de soluciones generadas por la meta-heurística grasp.
- *m:* Número de máquinas del problema según la instancia seleccionada.
- *n:* Número de trabajos del problema según la instancia seleccionada.
- *matriz:* Matriz del problema correspondiente a las instancias del *benchmark.*

Las tres últimas salidas se hacen con el fin de guardar las variables para utilizarlas en funciones posteriores.

```
function [poblacion_grasp,m,n,matriz] =
```

```
GRASP(matriz,tamano_poblacion_grasp)
poblacion_grasp = [];
for p=1:tamano_poblacion_grasp
    % Parametros iniciales
    alpha = rand;
    tiempos = matriz(:,2:2:end);
    maquinas = matriz(:,1:2:end);
    n = size(tiempos,1);
    m = size(tiempos,2);
    sizeCromosoma = n*m;
    % Copia de los tiempos y maquinas para ir descartando las ya
escogidas
    tiemposCopia = tiempos;
    maquinasCopia = maquinas;
    % Primera lista de candidatos
    opCandidatas = tiemposCopia(:,1);
    maqCandidatas = maquinasCopia(:,1);
    % TamaÒo m·ximo del diagrama de Gantt
    suma = sum(tiempos);
    % IniciaciÛn del diagrama de Gantt para la programaciÛn de las
    % oepraciones
    gantt_maquinas = zeros(size(matriz,1),sum(suma));
    gantt_trabajos = zeros(size(matriz,1),sum(suma));
    % InicializaciÛn del cromosoma
    cromosoma = [];
    for ii=1:sizeCromosoma
        [c,d]=size(opCandidatas);
        % Lista de candidatos actualizada, evaluando Cmax parcial
        opCanNuevaActualizada=[];
        for k=1:c
            if opCandidatas(k) > 0 && maqCandidatas(k) >= 0
            % Encontrar la ocupaciÛn de las maquinas y los trabajos en
Gantt
            ocupacion_trabajo = find(gantt_trabajos(k,:));
            ocupacion_maquina =
find(gantt_maquinas(maqCandidatas(k)+1,:));
            % Vector de espacios disponibles en las maquinas
            espacios_disponibles = [];
            % Gantt para la evaluaciÛn parcial del Cmax: Se va
reiniciando
            % en cada iteraciÛn, tomando el valor del Gantt general
            gantt_maquinas_parcial = gantt_maquinas;
            gantt_trabajos_parcial = gantt_trabajos;
            % Encuentra si el trabajo y la m·quina ya han sido escogidos
                if isempty(ocupacion_trabajo) == 1
                    if isempty(ocupacion_maquina) == 1
                        columna = 1;
                    else
                        % Primer espacio entre la m·quina y la columna 1
                        espacio_1 = min(ocupacion_maquina)-1;
                        % Halla los espacios disponibles entre la
ocupaciÛn
                        consecutivos_maquinas = diff(ocupacion_maquina)
== 1;
```

```
                        no_consecutivos = find(consecutivos_maquinas ==
0);
                        y = size(no_consecutivos,2);
                        for j=1:y
                             espacio_maquinas =
ocupacion_maquina(no_consecutivos(j)+1)-
ocupacion_maquina(no_consecutivos(j));
                             espacios_disponibles = [espacios_disponibles
espacio_maquinas];
                        end
                        [x,y] = size(espacios_disponibles);
                        if espacios_disponibles == 0
                            columna = max(ocupacion_maquina)+1;
                        else
                           if opCandidatas(k) <= espacio_1
                               columna = 1;
                           else
                               % Vector que define el estado de los
                               % espacios dispobibles para la candidata
                               alcanza = espacios_disponibles >=
opCandidatas(k);
                               % Encuentra el primer espacio disponible
                               % en la m·quina
                               espacio = find(alcanza,1);
                               if isempty(espacio) == 1
                                   columna = max(ocupacion_maquina)+1;
                               else
                                   columna =
ocupacion_maquina(no_consecutivos(espacio))+1;
                               end
                           end
                       end
                   end
               else
                   if isempty(ocupacion_maquina) == 1
                      columna = max(ocupacion_trabajo)+1;
                   else
                      if max(ocupacion_trabajo) >=
max(ocupacion_maquina)
                          columna = max(ocupacion_trabajo)+1;
                      else
                          [f,g] = size(ocupacion_maquina);
                          % Valores posteriores a la precedencia
                          mayores = ocupacion_maquina(ocupacion_maquina
> max(ocupacion_trabajo));
                          % Valores anteriores a la precedencia
                          menores = ocupacion_maquina(ocupacion_maquina
<= max(ocupacion_trabajo));
                          % Primer espacio entre el mÌnimo valor de la
                          % ocupaciÛn de la m·qina despuÈs de la
                          % precedencia y el valor de precedencia
                          espacio_1 = (min(mayores)-
max(ocupacion_trabajo))-1;
                          consecutivos_mayores = diff(mayores) == 1;
```

```
                            no_consecutivos = find(consecutivos_mayores ==
0);
                            y = size(no_consecutivos,2);
                            for j=1:y
                                espacio_maquinas =
ocupacion_maquina(no_consecutivos(j)+1)-
ocupacion_maquina(no_consecutivos(j));
                                espacios_disponibles =
[espacios_disponibles espacio_maquinas];
                            end
                            [x,y] = size(espacios_disponibles);
                            if espacios_disponibles == 0
                                columna = max(ocupacion_maquina)+1;
                            else
                                if opCandidatas(k) <= espacio_1
                                    columna = max(ocupacion_trabajo)+1;
                                else
                                    alcanza = espacios_disponibles >=
opCandidatas(k);
                                    espacio = find(alcanza,1);
                                    if isempty(espacio) == 1
                                        columna =
max(ocupacion_maquina)+1;
                                    else
                                        columna =
ocupacion_maquina(no_consecutivos(espacio))+1;
                                    end
                                end
                            end
                        end
                    end
                end
                % DespuÈs de seleccionada la columna se implanta el valor
                % esogido en el Gantt parcial

gantt_maquinas_parcial(1+maqCandidatas(k),columna:columna+opCandidatas(k)
-1) = 1;
                gantt_trabajos_parcial
(k,columna:columna+opCandidatas(k)-1) = 1;
                [a,b] = size(gantt_maquinas_parcial);
                % Procedimiento para evaluar la m·xima ocupacion en todas
                % las m·quinas
                maxima_ocupacion = [];
                for j=1:a
                    ocupacion_parcial =
find(gantt_maquinas_parcial(j,:));
                    if isempty(ocupacion_parcial) == 1
                        maxima_ocupacion = [maxima_ocupacion 0];
                    else
                        maxima_ocupacion = [maxima_ocupacion
max(ocupacion_parcial)];
                    end
                end
            % Basados en la m·xima ocupaciÛn general el Cmax parcial de
```

```
                % cada candidata
                opCanNuevaActualizada = [opCanNuevaActualizada ;
max(maxima_ocupacion)];
            else
            opCanNuevaActualizada = [opCanNuevaActualizada ; 0];
            end
        end
        % Procedimiento de creaciÛn de la Lista Restringida de Candidatos
        mini = min(opCanNuevaActualizada(opCanNuevaActualizada>0));
        maxi = max(opCanNuevaActualizada);
        rango = maxi-mini;
        limsup = mini+(alpha*rango);
        [m1,n1] = size(opCanNuevaActualizada);
        RCL = [];
        pos = [];
        for i=1:m1
            x=opCanNuevaActualizada(i);
            if mini<= x & x <= limsup
                RCL = [RCL x];
                pos = [pos i];
            end
        end
        % Si el tamaÒo de l RCL es uno no se hace evaluaciÛn y se escoge
el
        % valor presente en ella
        if size(RCL) == 1
            posEscogidaCandi=pos;
            Maqescog = maqCandidatas(pos);
            Cmax=opCanNuevaActualizada(pos);
            valorEscogido=opCandidatas(pos);
        else
            % Si el tamaÒo de la RCL es dif de 1 se hace el proceso de
            % selecciÛn
            % FunciÛn de sesgo
            f_sesgo = 1./RCL;
            total_fsesgo = sum(f_sesgo);
            % FunciÛn relativa: Porcentaje de cada elemento
(Probabilidad)
            f_relativa = f_sesgo./total_fsesgo;
            [a,b]=size(f_relativa);
            % FunciÛn acumulada: Valor acumulado de cada elemento
            f_acumulada = [];
            for j=1:b
                if j==1
                    f_acumulada(j) = f_relativa(1);
                else
                    f_acumulada(j) = f_relativa(j)+f_acumulada(j-1);
                end
            end
            % SelecciÛn aleatoria de cada elemento de la RCL
            z=0;
            num_aleatorio = rand;
            for j=1:b
                if num_aleatorio <= f_acumulada(j)
```

```
                    escoger = f_acumulada(j);
                    z=j;
                    break
                end
            end
            posEscogidaCandi=pos(z);
            Maqescog = maqCandidatas(pos(z));
            Cmax=opCanNuevaActualizada(pos(z));
            valorEscogido=opCandidatas(pos(z));
        end
        % GeneraciÛn del Gantt definitivo en el caso de la RCL > 1
        ocupacion_trabajo = find(gantt_trabajos(posEscogidaCandi,:));
        ocupacion_maquina = find(gantt_maquinas(Maqescog+1,:));
        espacios_disponibles = [];
        if isempty(ocupacion_trabajo) == 1
            if isempty(ocupacion_maquina) == 1
                columna = 1;
            else
                espacio_1 = min(ocupacion_maquina)-1;
                consecutivos_maquinas = diff(ocupacion_maquina) == 1;
                no_consecutivos = find(consecutivos_maquinas == 0);
                y = size(no_consecutivos,2);
                for j=1:y
                     espacio_maquinas =
ocupacion_maquina(no_consecutivos(j)+1)-
ocupacion_maquina(no_consecutivos(j));
                     espacios_disponibles = [espacios_disponibles
espacio_maquinas];
                end
                [x,y] = size(espacios_disponibles);
                if espacios_disponibles == 0
                    columna = max(ocupacion_maquina)+1;
                else
                   if valorEscogido <= espacio_1
                       columna = 1;
                   else
                       alcanza = espacios_disponibles >= valorEscogido;
                       espacio = find(alcanza,1);
                       if isempty(espacio) == 1
                           columna = max(ocupacion_maquina)+1;
                       else
                           columna =
ocupacion_maquina(no_consecutivos(espacio))+1;
                       end
                   end
               end
            end
        else
            if isempty(ocupacion_maquina) == 1
               columna = max(ocupacion_trabajo)+1;
            else
               if max(ocupacion_trabajo) >= max(ocupacion_maquina)
                   columna = max(ocupacion_trabajo)+1;
               else
```

```
                    [f,g] = size(ocupacion_maquina);
                    mayores = ocupacion_maquina(ocupacion_maquina >
max(ocupacion_trabajo));
                    menores = ocupacion_maquina(ocupacion_maquina <=
max(ocupacion_trabajo));
                    espacio_1 = (min(mayores)-max(ocupacion_trabajo))-1;
                    consecutivos_mayores = diff(mayores) == 1;
                    no_consecutivos = find(consecutivos_mayores == 0);
                    y = size(no_consecutivos,2);
                    for j=1:y
                        espacio_maquinas =
ocupacion_maquina(no_consecutivos(j)+1)-
ocupacion_maquina(no_consecutivos(j));
                        espacios_disponibles = [espacios_disponibles
espacio_maquinas];
                    end
                    [x,y] = size(espacios_disponibles);
                    if espacios_disponibles == 0
                        columna = max(ocupacion_maquina)+1;
                    else
                        if valorEscogido <= espacio_1
                            columna = max(ocupacion_trabajo)+1;
                        else
                            alcanza = espacios_disponibles >=
valorEscogido;
                            espacio = find(alcanza,1);
                            if isempty(espacio) == 1
                                columna = max(ocupacion_maquina)+1;
                            else
                                columna =
ocupacion_maquina(no_consecutivos(espacio))+1;
                            end
                        end
                    end
                end
            end
        end
        gantt_maquinas(Maqescog+1,columna:columna+valorEscogido-1) = 1;
        gantt_trabajos(posEscogidaCandi,columna:columna+valorEscogido-1)
= 1;
        % Encuentra la fila de la cual se extrajo el valor escogido
        filaEscogida = tiemposCopia(posEscogidaCandi,:);
        % Encuentra la columna de la cual se extrajo el valor escogido
        col=find(filaEscogida==valorEscogido,1);
        % El tiempo esogido se hace cero para descartarlo de los
candidatos
        tiemposCopia(posEscogidaCandi,col)=0;
        % La máquina escogida se hace -1 para descartarla de las
candidatas
        maquinasCopia(posEscogidaCandi,col)=-1;
        % Se va llenando los cromosomas con las posiciones y m·quinas
        % escoguidas
        cromosoma = [cromosoma posEscogidaCandi];
        % Generación de los nuevos candidatos a partir del valor ya
```

```
        % escogido
        switch posEscogidaCandi
            case 1
                if(col==m)
                    OpCanNueva = tiemposCopia(1,end);
                    OpCanNueva = vertcat(OpCanNueva,opCandidatas(2:end));
                else
                    OpCanNueva = tiemposCopia(1,col+1);
                    OpCanNueva = vertcat(OpCanNueva,opCandidatas(2:end));
                end
            case n
                if(col==m)
                    OpCanNueva = opCandidatas(1:end-1);
                    OpCanNueva = vertcat(OpCanNueva,tiemposCopia(n,end));
                else
                    OpCanNueva = opCandidatas(1:n-1);
                    OpCanNueva =
vertcat(OpCanNueva,tiemposCopia(n,col+1));
                end
            otherwise
                if(col==m)
                    OpCanNueva = opCandidatas(1:posEscogidaCandi-1);
                    OpCanNueva = vertcat(OpCanNueva,
tiemposCopia(posEscogidaCandi,end));
                    OpCanNueva =
vertcat(OpCanNueva,opCandidatas(posEscogidaCandi+1:end));
                else
                    OpCanNueva = opCandidatas(1:posEscogidaCandi-1);
                    OpCanNueva = vertcat(OpCanNueva,
tiemposCopia(posEscogidaCandi,col+1));
                    OpCanNueva =
vertcat(OpCanNueva,opCandidatas(posEscogidaCandi+1:end));
                end
        end
        % Generación de las nuevas máquinas candidatas a partir de la ya
        % escogida
        switch posEscogidaCandi
            case 1
                if(col==m)
                    MaqCanNueva = maquinasCopia(1,end);
                    MaqCanNueva =
vertcat(MaqCanNueva,maqCandidatas(2:end));
                else
                    MaqCanNueva = maquinas(1,col+1);
                    MaqCanNueva =
vertcat(MaqCanNueva,maqCandidatas(2:end));
                end
            case n
                if(col==m)
                    MaqCanNueva = maqCandidatas(1:end-1);
                    MaqCanNueva =
vertcat(MaqCanNueva,maquinasCopia(n,end));
                else
                    MaqCanNueva = maqCandidatas(1:n-1);
```

```
                    MaqCanNueva =
vertcat(MaqCanNueva,maquinasCopia(n,col+1));
                end
            otherwise
                if(col==m)
                    MaqCanNueva = maqCandidatas(1:posEscogidaCandi-1);
                    MaqCanNueva = vertcat(MaqCanNueva,
maquinasCopia(posEscogidaCandi,end));
                    MaqCanNueva = vertcat(MaqCanNueva,
maqCandidatas(posEscogidaCandi+1:end));
                else
                    MaqCanNueva = maqCandidatas(1:posEscogidaCandi-1);
                    MaqCanNueva =
vertcat(MaqCanNueva,maquinasCopia(posEscogidaCandi,col+1));
                    MaqCanNueva =
vertcat(MaqCanNueva,maqCandidatas(posEscogidaCandi+1:end));
                end
        end
        % Se igualan las variables para repetir el procedimiento
        opCandidatas=OpCanNueva;
        maqCandidatas=MaqCanNueva;
    end
    poblacion_grasp = [poblacion_grasp ; cromosoma Cmax];
end
end
```

Función POBLACION_ALEATORIA:

Función que crea parte de la población inicial, construyendola de manera aleatoria. Al igual que la matriz anterior tiene $(n * m) + 1$ columnas y las filas están determinadas por el número de individuos que se desea crear de esta manera.

Entradas:

- *tamano_poblacion_aleatoria:* número de individuos de la población inicial creada de manera aleatoria.
- *m:* número de máquinas del problema correspondiente a la instancia del *benchmark.*
- *n:* número de trabajos del problema correspondiente a la instancia del *benchmark.*

- *matriz: :* Matriz del problema que se desea solucionar correspondiente a las instancias del *benchmark*.

Salidas:

- *poblacion_aleatorios:* Matriz de soluciones generadas de manera aleatoria.

```
function [poblacion_aleatorios] =
POBLACION_ALEATORIA(tamano_poblacion_aleatoria,m,n,matriz)
%% Generación de padres aleatorios
 % El cromosoma general contiene m números 1, m números 2,..., m números
n
 % en orden
tiempos = matriz(:,2:2:end);
maquinas = matriz(:,1:2:end);
suma = sum(tiempos);
cromosoma_general = [];
cromosoma_parcial = [];
for j=1:n
    cromosoma_parcial(1,1:m) = j;
    cromosoma_general = [cromosoma_general cromosoma_parcial];
end
  % Se genera un vector aleatorio que contenga las posiciones del
cromosoma
poblacion_aleatorios = [];
while size(poblacion_aleatorios,1) < tamano_poblacion_aleatoria
    posiciones = randperm(n*m);
    padre_aleatorio = [];
    for j = 1:(n*m)
        padre_aleatorio(j) = cromosoma_general(posiciones(j));
    end
    cromosoma = padre_aleatorio;
    cromosoma_copia = [];
    gantt_maquinas = zeros(size(matriz,1),sum(suma));
    gantt_trabajos = zeros(size(matriz,1),sum(suma));
    b = size(cromosoma,2);
    tiempos_copia = tiempos;
    maquinas_copia = maquinas;
    for i=1:b
        trabajo = cromosoma(i);
        fila = tiempos_copia(trabajo,:);
        operacion = find(fila,1);
        tiempo = tiempos(trabajo,operacion);
        tiempos_copia(trabajo,operacion)=0;
        maquina = maquinas(trabajo,operacion);
        maquinas_copia (trabajo,operacion)=-1;
        ocupacion_trabajo = find(gantt_trabajos(trabajo,:));
        ocupacion_maquina = find(gantt_maquinas(maquina+1,:));
        espacios_disponibles = [];
       if isempty(ocupacion_trabajo) == 1
```

```
            if isempty(ocupacion_maquina) == 1
                columna = 1;
            else
                espacio_1 = min(ocupacion_maquina)-1;
                consecutivos_maquinas = diff(ocupacion_maquina) == 1;
                no_consecutivos = find(consecutivos_maquinas == 0);
                y = size(no_consecutivos,2);
                for j=1:y
                    espacio_maquinas =
ocupacion_maquina(no_consecutivos(j)+1)-
ocupacion_maquina(no_consecutivos(j));
                    espacios_disponibles = [espacios_disponibles
espacio_maquinas];
                end
                if espacios_disponibles == 0
                    columna = max(ocupacion_maquina)+1;
                else
                    if tiempo <= espacio_1
                        columna = 1;
                    else
                        alcanza = espacios_disponibles >= tiempo;
                        espacio = find(alcanza,1);
                        if isempty(espacio) == 1
                            columna = max(ocupacion_maquina)+1;
                        else
                            columna =
ocupacion_maquina(no_consecutivos(espacio))+1;
                        end
                    end
                end
            end
        else
            if isempty(ocupacion_maquina) == 1
                columna = max(ocupacion_trabajo)+1;
            else
                if max(ocupacion_trabajo) >= max(ocupacion_maquina)
                    columna = max(ocupacion_trabajo)+1;
                else
                    [f,g] = size(ocupacion_maquina);
                    mayores = ocupacion_maquina(ocupacion_maquina >
max(ocupacion_trabajo));
                    menores = ocupacion_maquina(ocupacion_maquina <=
max(ocupacion_trabajo));
                    espacio_1 = (min(mayores)-max(ocupacion_trabajo))-1;
                    consecutivos_mayores = diff(mayores) == 1;
                    no_consecutivos = find(consecutivos_mayores == 0);
                    y = size(no_consecutivos,2);
                    for j=1:y
                        espacio_maquinas =
ocupacion_maquina(no_consecutivos(j)+1)-
ocupacion_maquina(no_consecutivos(j));
                        espacios_disponibles = [espacios_disponibles
espacio_maquinas];
                    end
```

```
                    x = size(espacios_disponibles,2);
                    if espacios_disponibles == 0
                        columna = max(ocupacion_maquina)+1;
                    else
                        if tiempo <= espacio_1
                            columna = max(ocupacion_trabajo)+1;
                        else
                            alcanza = espacios_disponibles >= tiempo;
                            espacio = find(alcanza,1);
                            if isempty(espacio) == 1
                                columna = max(ocupacion_maquina)+1;
                            else
                                columna =
ocupacion_maquina(no_consecutivos(espacio))+1;
                            end
                        end
                    end
                end
            end
        end
         cromosoma_copia = [cromosoma_copia cromosoma(i)];
         gantt_maquinas(maquina+1,columna:columna+tiempo-1) = 1;
         gantt_trabajos(trabajo,columna:columna+tiempo-1) = 1;
    end
    maxima_ocupacion = [];
    for j=1:m
        ocupacion_parcial = find(gantt_maquinas(j,:));
        maxima_ocupacion = [maxima_ocupacion max(ocupacion_parcial)];
    end
    Cmax = max(maxima_ocupacion);
    poblacion_aleatorios = [poblacion_aleatorios ; padre_aleatorio Cmax];
end
end
```

Función POBLACION_INICIAL:

Esta función une las dos formas de crear la población inicial.

Entradas:

- *tamano_mejores:* tamaño la de matriz de mejores candidatos.
- *tamano_restantes:* tamaño de la matriz de los candidatos restantes.
- *poblacion_grasp:* población generada por medio de la meta-heurística GRASP.
- *población_aleat*orios: población generada de manera aleatoria.

Salidas:

- *poblacion_inicial:* matriz de población inicial teniendo en cuenta la unión de las dos poblaciones principales.
- *mejores:* matriz de mejores individuos de la población.
- *restantes:* matriz de individuos que relaciona las soluciones de menor desempeño en la función objetivo.
- *mejor_individuo:* Mejor individuo de la población, es decir, el primero de la matriz de mejores individuos.
- *tamano_mejores:* tamaño de matriz de mejores individuos.
- *tamano_restantes:* tamaño de la matriz de soluciones restantes.

```
function [poblacion_inicial, mejores, restantes,
mejor_individuo,tamano_mejores,tamano_restantes] =
POBLACION_INICIAL(tamano_mejores,tamano_restantes, poblacion_grasp,
poblacion_aleatorios)
    %% PoblaciÛn inicial
    poblacion_inicial = [poblacion_grasp ; poblacion_aleatorios];
    %% Mejores individuos de la poblaciÛn inicial
    [Y I] = sort (poblacion_inicial(:,end),1,'ascend');
    mejores = [];
    for i=1:tamano_mejores
        individuo_mejores = poblacion_inicial(I(i),:);
        mejores = [mejores ; individuo_mejores];
    end
    restantes = [];
    for i=tamano_mejores+1:tamano_mejores+tamano_restantes
        individuo_restante = poblacion_inicial(I(i),:);
        restantes = [restantes ; individuo_restante];
    end
    mejor_individuo= mejores(1,:);
end
```

Función SELECCIÓN:

La función de selección determina los individuos de la población inicial y posteriores generaciones que se van a cruzar, estableciendo como fijos los mejores individuos de la generación y seleccionando aleatoriamente de la matriz de individuos con menor desempeño en la función objetivo.

Entradas:

- *tamano_padres:* tamaño de la matriz de padres que se van a cruzar entre sí

- *tamano_mejores:* tamaño de matriz de mejores individuos.
- *tamano_restantes:* tamaño de la matriz de soluciones restantes.
- *restantes:* matriz de individuos que relaciona las soluciones de menor desempeño en la función objetivo.
- *mejores:* matriz de mejores individuos de la población.

Salidas:

- *matriz_padres:* Matriz de padres a cruzar.
- *tamano_padres:* tamaño de la matriz de padres que se van a cruzar entre sí.

```
function [matriz_padres,tamano_padres] = SELECCION
(tamano_padres,tamano_mejores,tamano_restantes,restantes,mejores)
 %% SelecciÛn aleatoria de padres para Cruce
    % GeneraciÛn de vector aleatorio para definir posisciones a ser
escogidas
    posiciones = randperm(tamano_restantes);
    % InicializaciÛn matriz de padres
    matriz_padres = [];
    for u=1:tamano_padres-tamano_mejores
        % Padre escogido de la poblaciÛn inicial para ir a la matriz de
padres
        padre = restantes(posiciones(u),:);
        matriz_padres = [matriz_padres ; padre];
    end
    matriz_padres = [matriz_padres ; mejores];
end
```

Función CRUCE:

La función de cruce tiene como fin escoger parejas de padres aleatoriamente de la matriz de padres, cruzarlos y obtener dos descendencias de cada cruce, todo esto mediante el operador de cruce seleccionado.

Entradas:

- *matriz_padres:* Matriz de padres a cruzar.
- *tamano_padres:* tamaño de la matriz de padres que se van a cruzar entre sí.

Salidas:

- *matriz_hijos:* Matriz de descendencias obtenidas del proceso de cruce.

```
function [matriz_hijos] = CRUCE(matriz_padres,tamano_padres)
 %% Cruce de los padres seleccionados
    % Genereador de posiciones aleatorias para cruce, se toma hasta el
total de
    % padres
    posiciones1 = randperm(tamano_padres);
    %Para garantizar una generaciÛn igual a la pobaciÛn inicial se hacen
dos
    %aleatorios
    posiciones2 = randperm(tamano_padres);
    % Total de posiciones que me indican cuales padres se han de cruzar
entre
    % si
    posiciones = horzcat(posiciones1,posiciones2);
    q = size(posiciones,2);
    % InicializaciÛn de matriz de hijos
    matriz_hijos = [];
    % i va de dos en dos para generar parejas
    for i=1:2:q-1
        padre_1 = matriz_padres(posiciones(i),1:end-1);
        padre_2 = matriz_padres(posiciones(i+1),1:end-1);
        %% GeneraciÛn de indices padre 1
        indices1 = [];
        b = size(padre_1,2);
        padre1_copia = [];
        % El primer indice siempre es 1
        indices1(1,1) = 1;
        for j=2:b
            % Vector que va llenando uno a uno los elementos del padre
            padre1_copia = [padre1_copia padre_1(j-1)];
            % Encuentra en el padre copia el elemento i del padre 1
            indice = find(padre1_copia == padre_1(j));
            if isempty(indice)
                gen = 1;
            else
                % N˙mero de veces que el indice est· en el padre copia
                cantidad = find(indice);
                ocurrencia_maxima = max(cantidad);
                gen = ocurrencia_maxima+1;
            end
            indices1 = [indices1 gen];
        end
        %% GeneraciÛn de indices padre 2
        indices2 = [];
        padre2_copia = [];
        % El primer indice siempre es 1
        indices2(1,1) = 1;
        for l=2:b
            % Vector que va llenando uno a uno los elementos del padre
            padre2_copia = [padre2_copia padre_2(l-1)];
            % Encuentra en el padre copia el elemento i del padre 2
```

```
            indice = find(padre2_copia == padre_2(l));
            if isempty(indice)
                gen = 1;
            else
                % N˙mero de veces que el indice est· en el padre copia
                cantidad = find(indice);
                ocurrencia_maxima = max(cantidad);
                gen = ocurrencia_maxima+1;
            end
            indices2 = [indices2 gen];
        end
        % Matriz de padres e indices
        mP1 = [padre_1 ; indices1];
        mP2 = [padre_2 ; indices2];
        %% Procedimiento de cruce
        % Definir el tamaÒo de la sub-cadena
        aleatorio = rand;
        tamano = [2 3 4];
        E = randi(3);
        if E == 3
            tamanoE = 1/tamano(E);
        else
            tamanoE = 1/(tamano(E)+aleatorio);
        end
        sizeChar = round(b*tamanoE);
        % Definir la posiciÛn en donde inicia la sub-cadena
        posicion = randi(b-sizeChar+1,1);
        %% Hijo 1
        charP1 = [];
        % Extraer la sub-cadena del padre 1
        for w = 1:sizeChar
            x = mP1(:,posicion+w-1);
            charP1 = [charP1 x];
        end
        copiaMP2 = mP2;
        % Hacer cero los elementos de la sub-cadena del padre 1 en el
padre 2
        for v=1:sizeChar
        x = mP1(:,(posicion-1)+v);
            for j=1:b
                if(mP2(:,j)==x)
                    copiaMP2(:,j) =0;
                end
            end
        end
        % Implantar la sub-cadena del padre 1 en el padre 2
        switch posicion
            case 1
                pseudohijo1 = [charP1 copiaMP2];
            otherwise
                parte1 = copiaMP2(:,1:posicion-1);
                parte2 = copiaMP2(:,posicion:end);
                pseudohijo1 = [parte1 charP1 parte2];
        end
```

```
        Phijo1 = pseudohijo1(1,:);
        % Eliminan los ceros del hijo 1
        hijo1 = Phijo1(Phijo1~=0);
        %% Hijo 2
        charP2 = [];
        % Extraer la sub-cadena del padre 2
        for ww = 1:sizeChar
            x = mP2(:,posicion+ww-1);
            charP2 = [charP2 x];
        end
        copiaMP1 = mP1;
        % Hacer ceros los elementos del la sub-cadena del padre 2 en el
padre 1
        for vv=1:sizeChar
        x = mP2(:,(posicion-1)+vv);
            for j=1:b
                if(mP1(:,j)==x)
                    copiaMP1(:,j) =0;
                end
            end
        end
        % Implantar la sub-cadena del padre 2 en el padre 1
        switch posicion
            case 1
                pseudohijo2 = [charP2 copiaMP1];
            otherwise
                parte1 = copiaMP1(:,1:posicion-1);
                parte2 = copiaMP1(:,posicion:end);
                pseudohijo2 = [parte1 charP2 parte2];
        end
        Phijo2 = pseudohijo2(1,:);
        % Eliminar ceros del hijo 2
        hijo2 = Phijo2(Phijo2~=0);
        % Cada hijo tiene un n˙mero aleatorio que representa la
probabilidad de
        % ser mutado
        matriz_hijos = [matriz_hijos ; hijo1 rand ; hijo2 rand ];
    end
end
```

Función MUTACIÓN:

Esta función, correspondiente al proceso de mutación, mediante una probabilidad establecida, selecciona los individuos a mutar de acuerdo al operador seleccionado.

Entradas:

- *matriz:* Matriz del problema que se desea solucionar correspondiente a las instancias del *benchmark*.

- *matriz_hijos:* Matriz de descendencias obtenidas del proceso de cruce.
- *prob_mut:* Probabilidad de que un individuo sea mutado.

Salidas:

- *Generacion:* Matriz de hijos después del proceso de mutación.

```
function [Generacion] = MUTACION(matriz, matriz_hijos, prob_mut)
    %% Proceso de mutaciÛn
    [p,q] = size(matriz_hijos);
    Generacion = [];
    tiempos = matriz(:,2:2:end);%%
    maquinas = matriz(:,1:2:end);%%
    suma = sum(tiempos);%%%
    [n,m] = size(tiempos);%%
    for i=1:p
        % Se pregunta si la probabilidad del hijo es menor o igual a la
probabilidad de mutaciÛn
        if matriz_hijos(i,q) <= prob_mut
            hijo = matriz_hijos(i,1:q-1);
            b = size(hijo,2);
            % Determinar las posiciones a intercambiar
            posicion1 = ceil(b*rand());
            posicion2 = ceil(b*rand());
            % En caso de tener el mismo valor se genera otro valor hasta
que sea
            % diferente
            while hijo(posicion2) == hijo(posicion1)
                posicion2 = ceil(b*rand());
            end
            % Se establecen los valores de las posiciones en el cromosoma
            valorpos1 = hijo(posicion1);
            valorpos2 = hijo(posicion2);
            % Se reemplazan los valores en las posiciones seleccionadas
            hijo(posicion1) = valorpos2;
            hijo(posicion2) = valorpos1;
            hijo_mutado = hijo;
            cromosoma = hijo_mutado;
            cromosoma_copia = [];
            gantt_maquinas = zeros(size(matriz,1),sum(suma));
            gantt_trabajos = zeros(size(matriz,1),sum(suma));
            b = size(cromosoma,2);
            tiempos_copia = tiempos;
            maquinas_copia = maquinas;
            for i=1:b
                trabajo = cromosoma(i);
                fila = tiempos_copia(trabajo,:);
                operacion = find(fila,1);
                tiempo = tiempos(trabajo,operacion);
                tiempos_copia(trabajo,operacion)=0;
```

```
                maquina = maquinas(trabajo,operacion);
                maquinas_copia (trabajo,operacion)=-1;
                ocupacion_trabajo = find(gantt_trabajos(trabajo,:));
                ocupacion_maquina = find(gantt_maquinas(maquina+1,:));
                espacios_disponibles = [];
               if isempty(ocupacion_trabajo) == 1
                   if isempty(ocupacion_maquina) == 1
                       columna = 1;
                   else
                       espacio_1 = min(ocupacion_maquina)-1;
                       consecutivos_maquinas = diff(ocupacion_maquina) ==
1;
                       no_consecutivos = find(consecutivos_maquinas ==
0);
                       y = size(no_consecutivos,2);
                       for j=1:y
                           espacio_maquinas =
ocupacion_maquina(no_consecutivos(j)+1)-
ocupacion_maquina(no_consecutivos(j));
                           espacios_disponibles = [espacios_disponibles
espacio_maquinas];
                       end
                       if espacios_disponibles == 0
                           columna = max(ocupacion_maquina)+1;
                       else
                           if tiempo <= espacio_1
                               columna = 1;
                           else
                               alcanza = espacios_disponibles >= tiempo;
                               espacio = find(alcanza,1);
                               if isempty(espacio) == 1
                                   columna = max(ocupacion_maquina)+1;
                               else
                                   columna =
ocupacion_maquina(no_consecutivos(espacio))+1;
                               end
                           end
                       end
                   end
               else
                   if isempty(ocupacion_maquina) == 1
                       columna = max(ocupacion_trabajo)+1;
                   else
                       if max(ocupacion_trabajo) >=
max(ocupacion_maquina)
                           columna = max(ocupacion_trabajo)+1;
                       else
                           [f,g] = size(ocupacion_maquina);
                           mayores = ocupacion_maquina(ocupacion_maquina
> max(ocupacion_trabajo));
                           menores = ocupacion_maquina(ocupacion_maquina
<= max(ocupacion_trabajo));
                           espacio_1 = (min(mayores)-
max(ocupacion_trabajo))-1;
```

```
                            consecutivos_mayores = diff(mayores) == 1;
                            no_consecutivos = find(consecutivos_mayores ==
0);
                            y = size(no_consecutivos,2);
                            for j=1:y
                                espacio_maquinas =
ocupacion_maquina(no_consecutivos(j)+1)-
ocupacion_maquina(no_consecutivos(j));
                                espacios_disponibles =
[espacios_disponibles espacio_maquinas];
                            end
                            x = size(espacios_disponibles,2);
                            if espacios_disponibles == 0
                                columna = max(ocupacion_maquina)+1;
                            else
                                if tiempo <= espacio_1
                                    columna = max(ocupacion_trabajo)+1;
                                else
                                    alcanza = espacios_disponibles >=
tiempo;
                                    espacio = find(alcanza,1);
                                    if isempty(espacio) == 1
                                        columna =
max(ocupacion_maquina)+1;
                                    else
                                        columna =
ocupacion_maquina(no_consecutivos(espacio))+1;
                                    end
                                end
                            end
                        end
                    end
                end
                 cromosoma_copia = [cromosoma_copia cromosoma(i)];
                 gantt_maquinas(maquina+1,columna:columna+tiempo-1) = 1;
                 gantt_trabajos(trabajo,columna:columna+tiempo-1) = 1;
            end
            maxima_ocupacion = [];
            for j=1:m
                ocupacion_parcial = find(gantt_maquinas(j,:));
                maxima_ocupacion = [maxima_ocupacion
max(ocupacion_parcial)];
            end
            Cmax = max(maxima_ocupacion);
            Generacion = [Generacion ; hijo_mutado Cmax];
        else
            hijo_no_mutado = matriz_hijos(i,1:end-1);
            cromosoma = hijo_no_mutado;
            cromosoma_copia = [];
            gantt_maquinas = zeros(size(matriz,1),sum(suma));
            gantt_trabajos = zeros(size(matriz,1),sum(suma));
            b = size(cromosoma,2);
            tiempos_copia = tiempos;
            maquinas_copia = maquinas;
```

```
            for i=1:b
                trabajo = cromosoma(i);
                fila = tiempos_copia(trabajo,:);
                operacion = find(fila,1);
                tiempo = tiempos(trabajo,operacion);
                tiempos_copia(trabajo,operacion)=0;
                maquina = maquinas(trabajo,operacion);
                maquinas_copia (trabajo,operacion)=-1;
                ocupacion_trabajo = find(gantt_trabajos(trabajo,:));
                ocupacion_maquina = find(gantt_maquinas(maquina+1,:));
                espacios_disponibles = [];
               if isempty(ocupacion_trabajo) == 1
                   if isempty(ocupacion_maquina) == 1
                       columna = 1;
                   else
                       espacio_1 = min(ocupacion_maquina)-1;
                       consecutivos_maquinas = diff(ocupacion_maquina) ==
1;
                       no_consecutivos = find(consecutivos_maquinas ==
0);
                       y = size(no_consecutivos,2);
                       for j=1:y
                           espacio_maquinas =
ocupacion_maquina(no_consecutivos(j)+1)-
ocupacion_maquina(no_consecutivos(j));
                           espacios_disponibles = [espacios_disponibles
espacio_maquinas];
                       end
                       if espacios_disponibles == 0
                           columna = max(ocupacion_maquina)+1;
                       else
                           if tiempo <= espacio_1
                               columna = 1;
                           else
                               alcanza = espacios_disponibles >= tiempo;
                               espacio = find(alcanza,1);
                               if isempty(espacio) == 1
                                   columna = max(ocupacion_maquina)+1;
                               else
                                   columna =
ocupacion_maquina(no_consecutivos(espacio))+1;
                               end
                           end
                       end
                   end
               else
                   if isempty(ocupacion_maquina) == 1
                       columna = max(ocupacion_trabajo)+1;
                   else
                       if max(ocupacion_trabajo) >=
max(ocupacion_maquina)
                           columna = max(ocupacion_trabajo)+1;
                       else
                           [f,g] = size(ocupacion_maquina);
```

```
                            mayores = ocupacion_maquina(ocupacion_maquina
> max(ocupacion_trabajo));
                            menores = ocupacion_maquina(ocupacion_maquina
<= max(ocupacion_trabajo));
                            espacio_1 = (min(mayores)-
max(ocupacion_trabajo))-1;
                            consecutivos_mayores = diff(mayores) == 1;
                            no_consecutivos = find(consecutivos_mayores ==
0);
                            y = size(no_consecutivos,2);
                            for j=1:y
                                espacio_maquinas =
ocupacion_maquina(no_consecutivos(j)+1)-
ocupacion_maquina(no_consecutivos(j));
                                espacios_disponibles =
[espacios_disponibles espacio_maquinas];
                            end
                            x = size(espacios_disponibles,2);
                            if espacios_disponibles == 0
                                columna = max(ocupacion_maquina)+1;
                            else
                                if tiempo <= espacio_1
                                    columna = max(ocupacion_trabajo)+1;
                                else
                                    alcanza = espacios_disponibles >=
tiempo;
                                    espacio = find(alcanza,1);
                                    if isempty(espacio) == 1
                                        columna =
max(ocupacion_maquina)+1;
                                    else
                                        columna =
ocupacion_maquina(no_consecutivos(espacio))+1;
                                    end
                                end
                            end
                        end
                    end
                end
                cromosoma_copia = [cromosoma_copia cromosoma(i)];
                gantt_maquinas(maquina+1,columna:columna+tiempo-1) = 1;
                gantt_trabajos(trabajo,columna:columna+tiempo-1) = 1;
            end
            maxima_ocupacion = [];
            for j=1:m
                ocupacion_parcial = find(gantt_maquinas(j,:));
                maxima_ocupacion = [maxima_ocupacion
max(ocupacion_parcial)];
            end
            Cmax = max(maxima_ocupacion);
            Generacion = [Generacion ; hijo_no_mutado Cmax];
        end
    end
end
```

Función MEJOR:

Esta función ordena los individuos de la generación de acuerdo a su *makespan* y selecciona una cantidad establecida de individuos para conservar en las siguientes generaciones.

Entradas:

- *Generación:* Matriz de hijos después del proceso de mutación.
- *mejor_individuo:* Mejor individuo de la población, es decir, el primero de la matriz de mejores individuos.
- *tamano_mejores:* tamaño de matriz de mejores individuos.
- *tamano_restantes:* tamaño de la matriz de soluciones restantes.

Salidas:

- *mejores:* Matriz de mejores individuos de la generación.
- *restantes*: Matriz de soluciones de menor desempeño en la función objetivo.
- *mejor_individuo:* Mejor individuo de la todas las generaciones.

```
function [mejores,restantes,mejor_individuo] =
MEJOR(Generacion,mejor_individuo,tamano_mejores,tamano_restantes)
 [Y I] = sort (Generacion(:,end),1,'ascend');
    mejores = [];
    for i=1:tamano_mejores
        individuo_mejores = Generacion(I(i),:);
        mejores = [mejores ; individuo_mejores];
    end
    restantes = [];
    for i=tamano_mejores+1:tamano_mejores+tamano_restantes
        individuo_restante = Generacion(I(i),:);
        restantes = [restantes ; individuo_restante];
    end
    if mejores(1,end) <= mejor_individuo(end)
        mejor_individuo = mejores(1,:);
    else
        mejor_individuo = mejor_individuo;
    end
end
```

Función ALGORITMO_GENETICO:

La función algoritmo genético realiza el proceso evolutivo y se divide en sub-funciones, las cuales contienen los operadores genéticos correspondientes al proceso.

Entradas:

- *numero_generaciones:* Criterio de parada que determina el número máximo de generaciones a crear en el proceso.
- *tamano_mejores:* tamaño de la matriz de los mejores individuos.
- *tamano_restantes:* tamaño de la matriz de las soluciones restantes.
- *mejores:* matriz de mejores individuos.
- *restantes:* matriz de individuos que relaciona las soluciones de menor desempeño en la función objetivo.
- *matriz:* Matriz del problema que se desea solucionar correspondiente a las instancias del *benchmark*.
- *mejor_individuo:* Mejor individuo de todo el proceso.
- *prob_mut:* Probabilidad de que un individuo sea mutado.

Salidas:

- *mejor_individuo:* Mejor individuo de todo el proceso.

```
function [mejor_individuo] =
ALGORITMO_GENETICO(numero_generaciones,tamano_mejores,tamano_restantes,me
jores,restantes,matriz,mejor_individuo,tamano_padres,prob_mut)
    matriz_generaciones = [];
    while size(matriz_generaciones) <
numero_generaciones*(tamano_mejores+tamano_restantes)
        [matriz_padres,tamano_padres] =
SELECCION(tamano_padres,tamano_mejores,tamano_restantes,restantes,mejores
);
        [matriz_hijos] = CRUCE (matriz_padres,tamano_padres);
        [Generacion] = MUTACION(matriz,matriz_hijos,prob_mut);
        [mejores,restantes,mejor_individuo] =
MEJOR(Generacion,mejor_individuo,tamano_mejores,tamano_restantes);
       matriz_generaciones = [matriz_generaciones ; Generacion];
    end
end
```

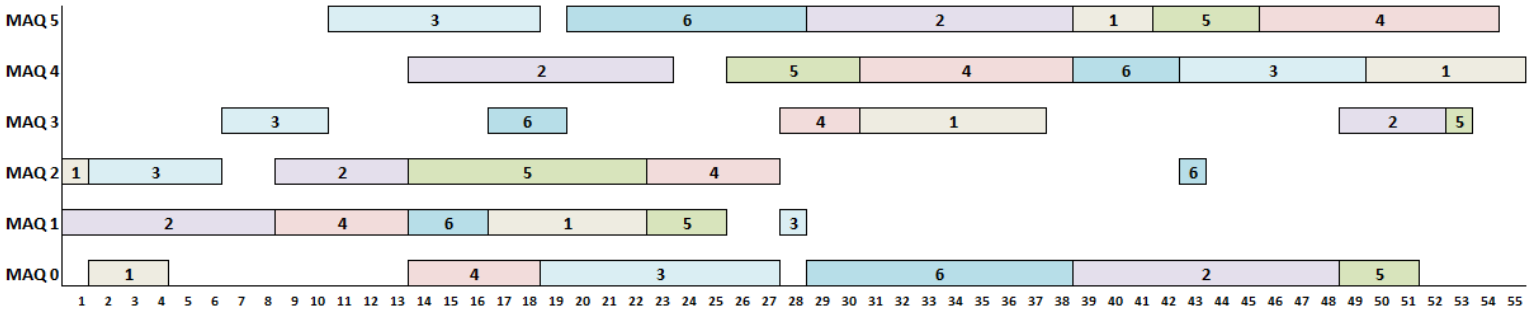
Script GRASP-AG

El script GRASP-AG resume el algoritmo propuesto, ya que incluye todas las funciones anteriormente mencionadas, por lo tanto permite cargar cada una de las instancias y realizar las corridas para cada una de ellas variando los parámetros necesarios.

Se muestra el ejemplo para la instancia ft06, para una población de 100, en donde 80 individuos se obtienen por medio de GRASP, 20 aleatorios, el tamaño mejores es de 5, tamaño restantes 95, numero de generaciones 100 y probabilidad de mutación 10%.

```
clear
clc
load 'ft06.mat'
tic
[poblacion_grasp,m,n,matriz] = GRASP (ft06,80);
[poblacion_aleatorios] = POBLACION_ALEATORIA(20,m,n,matriz);
[poblacion_inicial, mejores, restantes,
mejor_individuo,tamano_mejores,tamano_restantes,tamano_padres] =
POBLACION_INICIAL(5,95,poblacion_grasp, poblacion_aleatorios);
[mejor_individuo] =
ALGORITMO_GENETICO(100,tamano_mejores,tamano_restantes,mejores,restantes,
matriz,mejor_individuo,tamano_padres,0.10)
toc
```

ANEXO B. Representación de la solución para la instancia FT06.



ANEXO C. ANOVAS

En este anexo se encuentran las salidas de Minitab del análisis de varianza para cada una de las instancias.

ANOVA FT06

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	17,3875	2,4839	3,97	0,001
Lineal	3	13,6375	4,5458	7,26	0,000
Generación	1	10,5125	10,5125	16,78	0,000
Población	1	2,1125	2,1125	3,37	0,070
%mutación	1	1,0125	1,0125	1,62	0,208
Interacciones de 2 términos	3	3,4375	1,1458	1,83	0,149
Generación*Población	1	2,1125	2,1125	3,37	0,070
Generación*%mutación	1	1,0125	1,0125	1,62	0,208
Población*%mutación	1	0,3125	0,3125	0,50	0,482
Interacciones de 3 términos	1	0,3125	0,3125	0,50	0,482
Generación*Población*%mutación	1	0,3125	0,3125	0,50	0,482
Error	72	45,1000	0,6264		
Total	79	62,4875			

ANOVA LA01

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	5661,60	808,80	14,91	0,000
Lineal	3	5291,25	1763,75	32,51	0,000
Generación	1	5281,25	5281,25	97,35	0,000
Población	1	5,00	5,00	0,09	0,762
%mutación	1	5,00	5,00	0,09	0,762
Interacciones de 2 términos	3	207,90	69,30	1,28	0,289
Generación*Población	1	14,45	14,45	0,27	0,607
Generación*%mutación	1	1,25	1,25	0,02	0,880
Población*%mutación	1	192,20	192,20	3,54	0,064
Interacciones de 3 términos	1	162,45	162,45	2,99	0,088
Generación*Población*%mutación	1	162,45	162,45	2,99	0,088
Error	72	3906,20	54,25		
Total	79	9567,80			

ANOVA LA02

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	4510,8	644,40	6,10	0,000
Lineal	3	4066,3	1355,42	12,83	0,000
Generación	1	2599,2	2599,20	24,60	0,000
Población	1	1361,2	1361,25	12,88	0,001
%Mutación	1	105,8	105,80	1,00	0,320
Interacciones de 2 términos	3	443,3	147,77	1,40	0,250
Generación*Población	1	0,5	0,45	0,00	0,948
Generación*%Mutación	1	204,8	204,80	1,94	0,168
Población*%Mutación	1	238,0	238,05	2,25	0,138
Interacciones de 3 términos	1	1,3	1,25	0,01	0,914
Generación*Población*%Mutación	1	1,3	1,25	0,01	0,914
Error	72	7608,0	105,67		
Total	79	12118,8			

ANOVA LA03

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	2771,39	395,91	6,65	0,000
Lineal	3	2320,24	773,41	13,00	0,000
Generación	1	1557,61	1557,61	26,18	0,000
Población	1	762,61	762,61	12,82	0,001
%mutación	1	0,01	0,01	0,00	0,988
Interacciones de 2 términos	3	209,64	69,88	1,17	0,326
Generación*Población	1	23,11	23,11	0,39	0,535
Generación*%mutación	1	32,51	32,51	0,55	0,462
Población*%mutación	1	154,01	154,01	2,59	0,112
Interacciones de 3 términos	1	241,51	241,51	4,06	0,048
Generación*Población*%mutación	1	241,51	241,51	4,06	0,048
Error	72	4284,50	59,51		
Total	79	7055,89			

ANOVA LA04

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	2898,79	414,11	7,60	0,000
Lineal	3	2872,84	957,61	17,58	0,000
Generación	1	2610,61	2610,61	47,94	0,000
Población	1	255,61	255,61	4,69	0,034
%mutación	1	6,61	6,61	0,12	0,729
Interacciones de 2 términos	3	10,64	3,55	0,07	0,978
Generación*Población	1	5,51	5,51	0,10	0,751
Generación*%mutación	1	1,51	1,51	0,03	0,868
Población*%mutación	1	3,61	3,61	0,07	0,797
Interacciones de 3 términos	1	15,31	15,31	0,28	0,598
Generación*Población*%mutación	1	15,31	15,31	0,28	0,598
Error	72	3921,10	54,46		
Total	79	6819,89			

ANOVA LA06

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	123,687	17,6696	2,16	0,048
Lineal	3	91,737	30,5792	3,74	0,015
Generación	1	78,012	78,0125	9,53	0,003
Población	1	0,113	0,1125	0,01	0,907
%mutación	1	13,612	13,6125	1,66	0,201
Interacciones de 2 términos	3	22,837	7,6125	0,93	0,431
Generación*Población	1	0,112	0,1125	0,01	0,907
Generación*%mutación	1	13,612	13,6125	1,66	0,201
Población*%mutación	1	9,112	9,1125	1,11	0,295
Interacciones de 3 términos	1	9,113	9,1125	1,11	0,295
Generación*Población*%mutación	1	9,113	9,1125	1,11	0,295
Error	72	589,300	8,1847		
Total	79	712,988			

ANOVA LA09

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	2169,99	310,00	20,76	0,000
Lineal	3	1848,14	616,05	41,26	0,000
Generación	1	1683,61	1683,61	112,77	0,000
Población	1	10,51	10,51	0,70	0,404
%mutación	1	154,01	154,01	10,32	0,002
Interacciones de 2 términos	3	272,24	90,75	6,08	0,001
Generación*Población	1	21,01	21,01	1,41	0,239
Generación*%mutación	1	201,61	201,61	13,50	0,000
Población*%mutación	1	49,61	49,61	3,32	0,072
Interacciones de 3 términos	1	49,61	49,61	3,32	0,072
Generación*Población*%mutación	1	49,61	49,61	3,32	0,072
Error	72	1074,90	14,93		
Total	79	3244,89			

ANOVA LA11

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	10268,5	1466,93	18,65	0,000
Lineal	3	10019,3	3339,78	42,45	0,000
Generación	1	9923,5	9923,51	126,13	0,000
Población	1	5,5	5,51	0,07	0,792
%mutación	1	90,3	90,31	1,15	0,288
Interacciones de 2 términos	3	111,3	37,11	0,47	0,703
Generación*Población	1	90,3	90,31	1,15	0,288
Generación*%mutación	1	21,0	21,01	0,27	0,607
Población*%mutación	1	0,0	0,01	0,00	0,990
Interacciones de 3 términos	1	137,8	137,81	1,75	0,190
Generación*Población*%mutación	1	137,8	137,81	1,75	0,190
Error	72	5664,7	78,68		
Total	79	15933,2			

ANOVA LA16

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	714,62	102,089	0,94	0,502
Lineal	3	172,12	57,375	0,53	0,668
Genereación	1	135,37	135,375	1,25	0,280
Población	1	18,37	18,375	0,17	0,686
%mutación	1	18,37	18,375	0,17	0,686
Interacciones de 2 términos	3	540,46	180,153	1,66	0,215
Genereación*Población	1	330,04	330,042	3,05	0,100
Genereación*%mutación	1	0,38	0,375	0,00	0,954
Población*%mutación	1	210,04	210,042	1,94	0,183
Interacciones de 3 términos	1	2,04	2,042	0,02	0,893
Genereación*Población*%mutación	1	2,04	2,042	0,02	0,893
Error	16	1733,33	108,333		
Total	23	2447,96			

ANOVA LA21

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	2158,50	308,36	1,01	0,459
Lineal	3	1079,00	359,67	1,18	0,348
Genereación	1	130,67	130,67	0,43	0,522
Población	1	580,17	580,17	1,90	0,187
%mutación	1	368,17	368,17	1,21	0,288
Interacciones de 2 términos	3	948,83	316,28	1,04	0,403
Genereación*Población	1	816,67	816,67	2,68	0,121
Genereación*%mutación	1	10,67	10,67	0,04	0,854
Población*%mutación	1	121,50	121,50	0,40	0,537
Interacciones de 3 términos	1	130,67	130,67	0,43	0,522
Genereación*Población*%mutación	1	130,67	130,67	0,43	0,522
Error	16	4876,00	304,75		
Total	23	7034,50			

ANOVA LA28

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	5325,6	760,80	2,20	0,091
Lineal	3	1409,5	469,82	1,36	0,291
Generación	1	0,4	0,37	0,00	0,974
Población	1	737,0	737,04	2,13	0,164
%mutación	1	672,0	672,04	1,94	0,183
Interacciones de 2 términos	3	3824,1	1274,71	3,68	0,034
Generación*Población	1	392,0	392,04	1,13	0,303
Generación*%mutación	1	2185,0	2185,04	6,31	0,023
Población*%mutación	1	1247,0	1247,04	3,60	0,076
Interacciones de 3 términos	1	92,0	92,04	0,27	0,613
Generación*Población*%mutación	1	92,0	92,04	0,27	0,613
Error	16	5539,3	346,21		
Total	23	10865,0			

ANOVA LA31

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	5905,3	843,62	1,68	0,185
Lineal	3	4143,2	1381,06	2,75	0,077
Generación	1	1908,2	1908,17	3,80	0,069
Población	1	793,5	793,50	1,58	0,227
%mutación	1	1441,5	1441,50	2,87	0,110
Interacciones de 2 términos	3	1362,0	454,00	0,90	0,461
Generación*Población	1	0,7	0,67	0,00	0,971
Generación*%mutación	1	240,7	240,67	0,48	0,499
Población*%mutación	1	1120,7	1120,67	2,23	0,155
Interacciones de 3 términos	1	400,2	400,17	0,80	0,385
Generación*Población*%mutación	1	400,2	400,17	0,80	0,385
Error	16	8040,0	502,50		
Total	23	13945,3			

ANOVA LA37

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	18873,2	2696,17	10,08	0,000
Lineal	3	11725,0	3908,33	14,61	0,000
Generación	1	2320,7	2320,67	8,68	0,009
Población	1	7704,2	7704,17	28,81	0,000
%mutación	1	1700,2	1700,17	6,36	0,023
Interacciones de 2 términos	3	4331,5	1443,83	5,40	0,009
Generación*Población	1	10,7	10,67	0,04	0,844
Generación*%mutación	1	816,7	816,67	3,05	0,100
Población*%mutación	1	3504,2	3504,17	13,10	0,002
Interacciones de 3 términos	1	2816,7	2816,67	10,53	0,005
Generación*Población*%mutación	1	2816,7	2816,67	10,53	0,005
Error	16	4279,3	267,46		
Total	23	23152,5			

ANOVA FT10

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	1375,63	196,518	0,79	0,604
Lineal	3	689,46	229,819	0,93	0,450
Genereación	1	77,04	77,042	0,31	0,585
Población	1	135,38	135,375	0,55	0,471
%mutación	1	477,04	477,042	1,92	0,184
Interacciones de 2 términos	3	325,79	108,597	0,44	0,729
Genereación*Población	1	273,37	273,375	1,10	0,309
Genereación*%mutación	1	7,04	7,042	0,03	0,868
Población*%mutación	1	45,37	45,375	0,18	0,674
Interacciones de 3 términos	1	360,38	360,375	1,45	0,245
Genereación*Población*%mutación	1	360,38	360,375	1,45	0,245
Error	16	3965,33	247,833		
Total	23	5340,96			

ANOVA FT20

Análisis de Varianza

Fuente	GL	SC Ajust.	MC Ajust.	Valor F	Valor p
Modelo	7	6316,3	902,33	2,46	0,065
Lineal	3	4088,8	1362,93	3,71	0,034
Generación	1	513,4	513,38	1,40	0,255
Población	1	2882,0	2882,04	7,84	0,013
%mutación	1	693,4	693,38	1,89	0,189
Interacciones de 2 términos	3	2051,5	683,82	1,86	0,177
Generación*Población	1	392,0	392,04	1,07	0,317
Generación*%mutación	1	1650,0	1650,04	4,49	0,050
Población*%mutación	1	9,4	9,37	0,03	0,875
Interacciones de 3 términos	1	176,0	176,04	0,48	0,499
Generación*Población*%mutación	1	176,0	176,04	0,48	0,499
Error	16	5880,7	367,54		
Total	23	12197,0			

ANEXO D. Tiempo por proceso del algoritmo

En la imagen se puede observar que las líneas 770 y 771 son las que mayor poporción del tiempo consumen, las cuales corresponden a la construcción del Gantt para evaluar el makespan.