UN ALGORITMO MEMÉTICO PARA LA MINIMIZACIÓN DEL MAKESPAN EN EL PROBLEMA DEL JOB SHOP SCHEDULING.

KARIN JULIETH AGUILAR IMITOLA

YULEINY TATIANA PÉREZ DIAZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FISICO-MECÁNICAS
ESCUELA DE ESTUDIOS INDUSTRIALES Y EMPRESARIALES
BUCARAMANGA
2013

UN ALGORITMO MEMÉTICO PARA LA MINIMIZACIÓN DEL MAKESPAN EN EL PROBLEMA DEL JOB SHOP SCHEDULING.

KARIN JULIETH AGUILAR IMITOLA

YULEINY TATIANA PÉREZ DIAZ

Proyecto de grado para optar al título de

Ingeniero Industrial

Director

Ph. D HENRY LAMOS DIAZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FISICO-MECÁNICAS
ESCUELA DE ESTUDIOS INDUSTRIALES Y EMPRESARIALES
BUCARAMANGA
2013

AGRADECIMIENTOS

A Dios por permitir que este proyecto sea realidad.

A nuestros padres por su apoyo incondicional durante toda la carrera.

Al PhD. Henry Lamos Diaz por su tiempo, dedicación y el conocimiento transmitido en el desarrollo de esta investigación.

Al Ingeniero Leonardo Jaimes por su paciencia y compromiso.

A Oscar Arias y los auxiliares de sala de cómputo por su colaboración.

DEDICATORIA

A Dios por todas sus bendiciones.

A mi mamá por ser el motor de mi vida y la razón de todos mis esfuerzos.

A mi papá porque su esfuerzo, apoyo y cariño me motivaron para alcanzar esta meta.

A mis hermanas y familiares por acompañarme en este proceso.

A Jorge Olarte por creer en mí, ser mi compañero incondicional y permitirme soñar juntos.

A las personas que contribuyeron para hacer posible este sueño: mis suegros, mis cuñadas, Doña Luz, Doña Nubia y a mis amigos.

Karin Aguilar Imitola

A Dios por ser mi guía y fortaleza en todo momento.

A mi madre por su constancia al brindarme su apoyo e inmenso amor, porque gracias a ella esto es una realidad.

A mi padre por ser comprensivo, la confianza que ha depositado en mí siempre y por ser un pilar importante de mi vida.

A mis hermanos y mi primita Sharick porque fueron razones importantes para lograr tan anhelada meta.

A familiares que siempre creyeron en mí e incentivaron la culminación de este proceso.

A Javier por estar junto a mí y permitirme compartir este logro con él.

Yuleiny Tatiana Pérez Diaz

CONTENIDO

LISTA DE CUADROS

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

RESUMEN

TÍTULO: "UN ALGORITMO MEMÉTICO PARA LA MINIMIZACIÓN DEL MAKESPAN EN EL PROBLEMA DEL JOB SHOP SCHEDULING"[1]

AUTORES:
AGUILAR IMITOLA, Karin Julieth
PÉREZ DIAZ, Yuleiny Tatiana[2]

PALABRAS CLAVES:
Job Shop, Algoritmo memético, Búsqueda local, Schedule, Metaheurísticas.

DESCRIPCIÓN

El problema del Job Shop Scheduling se aborda en la presente investigación con el objetivo de minimizar el makespan. Es considerado un problema de optimización combinatoria catalogado de tipo NP-hard debido a su complejidad computacional y que actualmente ocupa un importante lugar en el secuenciamiento de operaciones en la industria para lograr la minimización de un recurso significativo como el tiempo.

Para la solución de este problema se han implementado diversos métodos heurísticos y metaheurísticos que resultan ser muy eficientes, aunque no garantizan la obtención de resultados óptimos, arrojan soluciones aproximadas y en un tiempo computacional razonable. Un algoritmo memético es propuesto para minimizar el makespan, aprovechando las características del algoritmo genético mediante la aplicación del operador de cruce JOX, la representación basada en operaciones y un método de búsqueda local modificado, basado en la estrategia de Nowicki y Smutnicki.

El algoritmo fue ejecutado en 17 problemas del benchmark extraídos de la OR-Library para su validación en el lenguaje de programación Matlab® y comparado con un algoritmo genético simple (GA). Los resultados obtenidos muestran que el algoritmo memético propuesto es más eficiente al encontrar mejores soluciones que el GA. El buen rendimiento del MA se ve reflejado al encontrar las mejores soluciones conocidas de instancias de baja y media complejidad y soluciones aproximadas en instancias de alta complejidad.

[1] Proyecto de Grado
[2] Facultad de Ingenierías Fisicomecánicas. Escuela de Estudios Industriales y Empresariales. Director: Ph. D. Henry Lamos Díaz

ABSTRACT

TITLE: "A MEMETIC ALGORITHM FOR MINIMIZING MAKESPAN IN JOB SHOP SCHEDULING PROBLEM"[3]

AUTHORS:
AGUILAR IMITOLA, Karin Julieth
PÉREZ DIAZ, Yuleiny Tatiana[4]

KEYWORDS:
Job Shop, Memetic Algorithm, Local Search, Schedule, Metaheuristics.

DESCRIPTION

The Job Shop Scheduling problem addressed in this research with the objective of minimizing the makespan. Is considered cataloged combinatorial optimization problem NP -hard type due to their computational complexity and currently represents an important place in the industry in the sequencing of operations to achieve the minimization of a significant resource as time.

To solve this problem have implemented several heuristics and metaheuristics that happen to be very efficient, but do not guarantee optimal results , yield approximate solutions and a reasonable computational time. A memetic algorithm is proposed to minimize the makespan, taking advantage of the characteristics of the genetic algorithm by applying the crossover operator JOX, operation-based representation and modified local search method based on Nowicki and Smutnicki strategy.

The algorithm was executed on 17 benchmark problems taken from the OR -Library for validation in the programming language Matlab® and compared with a simple genetic algorithm (GA). The results show that the proposed memetic algorithm is more efficient to find better solutions than the GA. The good performance of AM is reflected by finding best known solutions of instances of low and medium complexity and approximate solutions in instances of high complexity.

[3] Proyecto de Grado
[4] Facultad de Ingenierías Fisicomecánicas. Escuela de Estudios Industriales y Empresariales. Director: Ph. D. Henry Lamos Díaz

INTRODUCCIÓN

La necesidades actuales del mercado llevan a las empresas a programar de forma eficaz y eficiente los recursos disponibles, esta es una tarea compleja incluso para sistemas de producción pequeños, por ello en las últimas décadas ha aumentado el uso de herramientas de optimización y el desarrollo de algoritmos y heurísticas que faciliten la programación generando resultados de calidad a bajo costo; de esta forma resulta relevante el problema de Scheduling que intenta determinar cómo asignar recursos limitados a tareas a lo largo del tiempo para optimizar uno o más objetivos.

El Job shop es un problema de Scheduling que modela diversas situaciones reales y permite mejorar la eficiencia de los procesos de manufactura, el JSP consiste en asignar n trabajos que deben ser procesados en m máquinas, cumpliendo una serie de restricciones para uno o más objetivos, generalmente es la minimización del tiempo de finalización de la última tarea llamado makespan.

En JSP el desarrollo de métodos exactos que encuentren soluciones óptimas en un tiempo razonable es difícil, por ello las investigaciones se han centrado en métodos heurísticos que puedan encontrar buenas soluciones a bajo costo computacional. Se han desarrollado diversas técnicas heurísticas que buscan solucionar el problema hallando buenas respuestas en un tiempo de ejecución corto, algunas de estas técnicas son de búsqueda local como la búsqueda tabú (TS), el recocido simulado (AS), Búsqueda de entorno variable (VNS) o reglas de prioridad, otros de búsqueda global como algoritmos genéticos, GRASP, Optimización por enjambre de partículas (PSO) u Optimización mediante colonia de Hormigas (ACO).

En este trabajo, se propone la construcción de un algoritmo memético que busca minimizar el makespan en un problema Job Shop Scheduling, la eficiencia del

algoritmo se mide mediante la validación de instancias del benchmarking, el lenguaje de programación usado en la implementación del algoritmo de solución es Matlab®.

Los aportes de este proyecto son de relevancia para el grupo de investigación OPALO y la Escuela de Estudios Industriales y Empresariales en la consolidación y el fortalecimiento de la línea de investigación de programación de operaciones.

Cumplimiento de objetivos	
Objetivos específicos	**Numerales relacionados**
Realizar una revisión bibliográfica sobre los métodos metaheurísticos con enfoque evolutivo para determinar la solución de problemas combinatorios.	4.1
Aplicar métodos heurísticos para obtener soluciones factibles al problema de minimización del makespan en JSP.	3.2.1 Anexo A y B
Construir el framework basado en un algoritmo memético para el problema de minimización del makespan en Job Shop.	5 Anexo D
Implementar y validar el algoritmo construido mediante instancias del Benchmarking.	6
Analizar los factores relevantes que influyen en la calidad de la respuesta arrojada por el algoritmo memético por medio de un diseño experimental.	7

1. GENERALIDADES DEL PROYECTO

1.1. PLANTEAMIENTO DEL PROBLEMA

La programación de operaciones en procesos de manufactura ha sido de gran interés para muchos investigadores en sistemas de producción de bajo volumen en que los productos se hacen a través de órdenes. En forma general el problema JSP (por sus siglas en inglés, Job Shop Scheduling Problem) se define como la programación de un conjunto de n trabajos que deben ser procesados en un conjunto de m máquinas tecnológicamente diferentes; cada una con la capacidad de procesar máximo un trabajo al tiempo. Los trabajos siguen diferentes rutas de procesamiento entre las máquinas y un trabajo no puede ser procesado en más de una máquina de manera simultánea. Cada operación ejecutada en una máquina tiene un tiempo de procesamiento asignado.

El JSP con el objetivo de minimizar el makespan cae en la categoría de problemas NP-hard. Esta complejidad existe aun en pequeñas instancias de tres trabajos y tres máquinas.

La alta complejidad del problema hace que sea difícil computacionalmente encontrar una solución óptima en un tiempo razonable, por lo tanto se prefiere la búsqueda de soluciones aproximadas en tiempo polinomial para casos de instancia de tamaño grande, en lugar de soluciones exactas a un alto costo.

Un enfoque promisorio desarrollado en los últimos años para la solución del JSP es por medio de algoritmos evolutivos híbridos. Los algoritmos evolutivos híbridos combinan las ventajas de diferentes metaheurísticas evolutivas con heurísticas de mejoramiento permitiendo mejorar la calidad de la solución.

En el presente proyecto se usa un algoritmo memético que consiste en construir un algoritmo genético con el que se hará la exploración del espacio de búsqueda y un algoritmo de búsqueda local que explotará las soluciones encontradas, disminuyendo el tiempo de búsqueda dentro de regiones que ya han sido exploradas o que no han arrojado buenas soluciones.

1.2. JUSTIFICACIÓN DEL PROYECTO

La función del Scheduling es asignar en el tiempo la ejecución de tareas a un número de recursos limitados con la finalidad de optimizar uno o más objetivos; por esta razón el problema de JSP es uno de los problemas de Scheduling que se encuentra con mayor frecuencia en la vida real en diversos ambientes de producción y servicios, ya que permite incrementar la eficiencia de los procesos. Se puede mencionar algunos ejemplos como la planificación del aterrizaje de un conjunto de aviones, el desarrollo de proyectos de ingeniería; en los sistemas de manufactura resulta relevante para poder determinar en qué tiempo y máquinas serán efectuadas las operaciones.

Debido a su importancia práctica como teórica existe abundante literatura sobre investigaciones realizadas en esta área. La revisión de la literatura sobre el JSP muestra distintos enfoques que han sido usados para su solución; existiendo una gran cantidad de trabajos de investigación en la línea de los algoritmos heurísticos que generan buenas soluciones al problema JSP. Sin embargo, puede darse el caso que la solución encontrada sea un mínimo local por esta razón surge la necesidad de buscar otras alternativas que mejoren la solución hallada. En las últimas décadas han surgido algoritmos basados en el comportamiento social de

individuos que han llevado a determinar soluciones bastante buenas como lo enuncia Fogel et al.[5], Blum y Roli [6] en sus investigaciones.

En el presente proyecto se pretende abarcar dos aspectos de relevancia para la consolidación de “escuela” en el grupo OPALO. El primer aspecto tratará sobre una buena recopilación de los métodos heurísticos usados para hallar solución del JSP respecto a la minimización del Makespan en sistemas Job Shop. El segundo aspecto es la implementación y validación de un algoritmo memético en Matlab, mediante instancias del Benchmarking.

Al ser un problema que responde a una necesidad real con aplicación práctica, resulta útil encontrar un algoritmo que permita una solución de calidad a bajo costo al problema de programación de recursos que se presente tanto a nivel industrial como en servicios.

1.3. OBJETIVOS

1.3.1. Objetivo General

Desarrollar un algoritmo memético para la solución del problema de minimización del makespan en sistemas de Job Shop.

1.3.2. Objetivos específicos

- Realizar una revisión bibliográfica sobre los métodos metaheurísticos con enfoque evolutivo para determinar la solución de problemas combinatorios.

[5] FOGEL, Lawrence, OWENS, Alvin y WALSH, Michael. Artificial Intelligence Through Simulated Evolution, John Wiley, UK, 1966.

[6] BLUM, Christian y ROLI, Andrea. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. En: ACM Computing Surveys. Vol 35, N° 3. (2003), p. 268-308.

- Aplicar métodos heurísticos para obtener soluciones factibles al problema de minimización del makespan en JSP.
- Construir el framework basado en un algoritmo memético para el problema de minimización del makespan en Job Shop.
- Implementar y validar el algoritmo construido mediante instancias del Benchmarking.
- Analizar los factores relevantes que influyen en la calidad de la respuesta arrojada por el algoritmo memético por medio de un diseño experimental.

1.4. METODOLOGÍA

Durante la ejecución de este proyecto de grado se llevarán a cabo las siguientes etapas:

1) **Revisión bibliográfica:** Seleccionar la bibliografía disponible sobre los métodos metaheurísticos con enfoque evolutivo en problemas combinatorios, específicamente el de Job Shop Scheduling a partir de bases de datos que contienen temas multidisciplinarios como ElSevier, Scopus, Science Direct, SpringerLink entre otras, y su evolución a través de la investigación realizada por diversos autores. Posteriormente se recopilará la información en un documento que proporcione las bases teóricas para abordar el problema del Job Shop Scheduling.

2) **Selección de métodos heurísticos para la construcción de soluciones factibles**: Mediante la elección de los métodos heurísticos estudiados, generar soluciones factibles que alimenten el algoritmo propuesto para la minimización del makespan en el problema de JSP.

3) **Estudio del algoritmo:** En esta etapa se llevará a cabo la comprensión de las bases y fundamentos teóricos del algoritmo memético, teniendo en cuenta el lenguaje empleado en el desarrollo del algoritmo genético y el método de búsqueda local para ser aplicados posteriormente en la solución de problemas en sistemas Job Shop.

4) **Construcción del framework para el JSP.**

5) **Implementación del algoritmo en Matlab®:** Inicialmente se conocerá las bases y el funcionamiento del lenguaje de programación Matlab®, aplicar los elementos que corresponden a los métodos heurísticos y metaheurísticos en este programa. A continuación se realizará el pseudocódigo tomando en cuenta los parámetros del algoritmo genético híbrido para finalizar la construcción del algoritmo memético.

6) **Validación del algoritmo:** Seleccionar algunos de los problemas de Benchmarking propuesto por diversos autores y anteriores proyectos de grado para la validación del algoritmo construido.

7) **Diseño experimental:** Consultar los diversos tipos de diseños de experimentos para elegir el adecuado en la elaboración del análisis de los factores que influyen en la calidad de la respuesta del algoritmo memético. Seguir las pautas de acuerdo al diseño, identificación de las variables, definir el nivel de confianza y de esta forma determinar el número de réplicas necesarias del mismo.

2. EL PROBLEMA DE SCHEDULING

Los problemas de Scheduling son variados y dependen del entorno de producción donde se desarrollan. El primer problema que se estudió fue la optimización de un objetivo en una sola máquina, con la identificación de otros escenarios se han desarrollado metodologías que permiten acercarse más a los problemas complejos del mundo real. Los métodos exactos fueron los primeros en incursionar en esta área de la optimización combinatoria, aunque podían generar una solución óptima a medida que aumentaba la complejidad del problema se mostraban inviables, posteriormente los estudios se centraron en los métodos heurísticos y metaheurísticos. Las investigaciones actuales realizan combinaciones de varias de estas metodologías, esta mezcla se conoce como algoritmos híbridos.

2.1. PARÁMETROS DEL PROBLEMA

Generalmente un problema de Scheduling P puede ser formulado como un cuarteto, $P = (J, M, R, \mu)$.

Los elementos que pertenecen al conjunto J se denominan trabajos, hay n trabajos, y se denotan por $J_1, J_2, J_3, \ldots, J_n$. El trabajo J_i, $1 \leq i \leq n$, consiste de n_j operaciones $O_{1j,}\, O_{2j,} \ldots, O_{ij}$. Para cada operación se define un tiempo de procesamiento denotado por p_{ij} , donde $1 \leq j \leq n_j$ y $1 \leq i \leq$ n. Para cada trabajo J_i, se puede definir un tiempo de preparación r_i, una fecha limite d_i y una ponderación w_i, en la mayoría de los casos se trabaja con r_i=0.

Existen restricciones de precedencia entre los trabajos, que corresponde al orden en que estos deben ser ejecutados, pueden ser en forma de cadena, árbol, en serie, en paralelo o acíclicas.

Los trabajos son ejecutados por los elementos del conjunto M, llamado Máquinas, hay m máquinas y se denotan por M_1, $M_{2,}M_{3,}$..., M_m. R es un grupo de entidades adicionales necesarias para ejecutar los trabajos y μ corresponde a la función con la que se mide la calidad de las soluciones encontradas al problema estudiado.[7]

2.2. SCHEDULE

Un Programa *o Schedule* es una asignación de máquinas o recursos a trabajos en un tiempo determinado.

Tipos de Schedule

Dependiendo de las características, un Schedule asociado a un problema de scheduling puede ser:

- **Schedule factible:** un Schedule es factible si satisface todas las restricciones y condiciones específicas del problema.

- **Schedule óptimo**: un Schedule es óptimo si el valor del criterio de optimización para la programación es óptimo.

- **Schedule activo:** un Schedule factible es activo cuando ninguna operación puede finalizar temprano sin modificar el tiempo de otra, de este grupo hacen parte los Schedule non-delay (sin retraso).

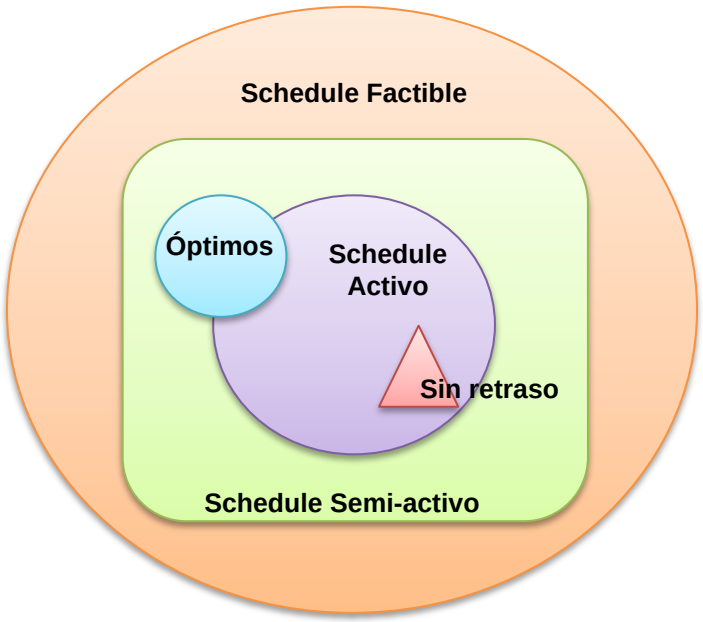
[7] GAWIEJNOWICZ, Stanislaw. Basics of the scheduling theory. En: _____. Time-dependent Scheduling. Springer Berlin Heidelberg, 2008. p. 35-46

- **Schedule Semi-activo:** un Schedule factible es semi-activo cuando ninguna operación puede ser completada con anticipación sin modificar la secuencia de los trabajos en las máquinas, estos Schedule pueden ser activos.

- **Schedule Non-delay (sin retraso):** es un Schedule factible en el que ninguna máquina se encuentra ociosa si existen trabajos esperando a ser ejecutados. Los Schedule sin retraso hacen parte de los Schedule activos, pero no necesariamente ocurre lo contrario.

La Figura 1 muestra la relación entre las clases de Schedule, se observa que las soluciones óptimas pueden pertenecer tanto a los Schedule activos como a los semi-activos, aunque no implica que estén dentro de los Schedule sin retraso.

Figura 1. Tipos de Schedule



2.3. OPTIMIZACIÓN COMBINATORIA

La optimización combinatoria es una disciplina de la matemática moderna aplicada y de la ciencia de la computación, que estudia problemas que se caracterizan por tener una cantidad finita de soluciones factibles.

La combinatoria es una herramienta flexible para la comprensión de problemas de optimización discreta, es decir, aquellos que utilizan exclusivamente variables discretas (el dominio de cada variable consiste en un conjunto finito de valores discretos). Se han diseñado muchos algoritmos para resolver problemas combinatorios en función del tamaño de la instancia del problema que se creen son difíciles de resolver. Las instancias permiten medir los resultados obtenidos en la aplicación de los diferentes métodos de solución y están compuestas por un conjunto de datos característicos del problema.

Se han estudiado avances recientes en casos especiales de solución polinomial en problemas de optimización combinatoria denominados “difíciles”. Tales casos se consideran especiales por su estructura de costos, geometría del problema, la estructura del grafo subyacente o mediante el análisis de algoritmos especiales[8].

En estos problemas el objetivo es encontrar el máximo (o el mínimo) de una determinada función sobre un conjunto finito de soluciones. Es importante denotar que dada la finitud de las posibles soluciones, las variables han de ser discretas, restringiendo su dominio a una serie finita de valores. Habitualmente, el espacio

[8] BURKARD, Rainer, Efficiently solvable special cases of hard combinatorial optimization problems, En: Mathematical Programming, 1997, Vol. 79, pp 55-69.

de búsqueda resulta ser muy elevado, haciendo impracticable la evaluación de todas sus soluciones para determinar el óptimo[9].

2.3.1. Problemas combinatorios

Muchos problemas de importancia tanto teórica como práctica se enfocan en la elección de la mejor configuración o el mejor conjunto de parámetros para lograr algún objetivo.

De acuerdo a Papadimitriou y Steigliz[10], un problema de optimización combinatoria $P = (S, f)$ es un problema de optimización que está formado por un conjunto finito de objetos S, llamado usualmente espacio de búsqueda y por una función objetivo $f: S \rightarrow R+$ que asigna un costo positivo a cada objeto $s^* \in S$. El objetivo es encontrar una solución que tenga el valor mínimo en la función, es decir el hallar el $s^* \in S | f(s^*) \leq f(s), \forall s \in S \; y \; s \neq s^*, s^*$ es llamado una solución óptima global de (S, f). Se puede resaltar que minimizar sobre una función objetivo f es lo mismo que maximizar utilizando $-f$, por lo que cualquier problema de optimización combinatoria puede ser descrito como de minimización.

En este tipo de problemas, las variables se agrupan en varios conjuntos que representan objetos que incluyen una estructura de datos compleja, como permutaciones, grafos, árboles, etc. Cada una de las variables del problema es

[9] MARTÍ, Rafael, Procedimientos Metaheurísticos en optimización combinatoria, [En línea] Universidad de Valencia, Departamento de Estadística e investigación operativa. (2003). p. 1-60 [Consultado 28 Ene. 2013].Disponible en: < http://www.uv.es/~rmarti/paper/cpapers.html>

[10] PAPADIMITRIOU,C.H y STEIGLITZ, K. Combinatorial Optimization: Algorithms and Complexity. Citado por BLUM, Christian y ROLI Andrea. Op. cit.

ubicada en ciertas posiciones generando una configuración. La combinatoria se encarga de estudiar dichas configuraciones. En los problemas combinatorios se trata de buscar cuál es la mejor configuración y para conocer cuál es mejor se construyen una o más funciones de valor sobre el espacio de las configuraciones; de acuerdo a la cantidad de funciones de valor construidas los problemas combinatorios pueden ser uniobjetivos (una función de valor) y multiobjetivos (más de una).[11]

Los problemas asociados al Scheduling (secuenciación) como el problema JSP se encuentran clasificados como un problema de naturaleza combinatoria, debido que se requiere encontrar una configuración para la programación de un conjunto de trabajos en un conjunto de máquinas (variables del problema), en el que cada trabajo contiene una serie de operaciones que deben ser procesadas en una secuencia definida y un tiempo de procesamiento establecido. En la mayoría de los casos en los cuales se ha tratado de solucionar el problema del Job Shop se desea encontrar la mejor configuración con el objetivo de minimizar el makespan.

En general, los problemas de optimización combinatoria (por sus siglas en inglés CO) son clasificados de acuerdo a su complejidad computacional, y esto ha llevado al desarrollo de muchos algoritmos para hallar la solución del problema. La CO no solo es útil para comprender la complejidad de los algoritmos sino también permite verificar si una propuesta de solución de un problema de optimización discreta es óptima[12].

[11] SANCHEZ, Miguel, Optimización combinatorial. En: Matemáticas del siglo XX: Una mirada en 101 artículos [En línea] Universidad de la Laguna, 2000, p. 115-120 [Consultado 11 Feb. 2013].Disponible en: < http://www.sinewton.org/numeros/numeros/43-44/Articulo22.pdf>

[12] LANGE, Kenneth. Applied Probability: Combinatorial Optimization. En: Springer Science Business Media. Cap. 5, (2010), p.103-122.

2.3.2. Complejidad computacional

Según Papadimitriou[13], la complejidad está determinada por las especificaciones de un modelo de computación (generalmente la máquina de Turing), el modo de computación y los recursos controlados (espacio o tiempo). La complejidad se determina por la dificultad del cálculo que está directamente relacionado con el uso de los recursos.

2.3.2.1. Clases de complejidad

La teoría de la complejidad divide el conjunto de problemas en clases según el límite superior del número de operaciones en el peor caso, las dos clases P y NP, relacionan sus siglas con la modo de computación de la máquina de Turing, la ***Clase P***, son los problemas que pueden ser resueltos por una máquina de Turing determinista en un tiempo polinomial y la ***Clase NP*** contiene aquellos problemas que se resuelven en tiempo polinomial por una máquina de Turing no determinista.

NP completos: un problema p es NP completo si p Є NP y todos los problemas de clase NP pueden ser reducidos a un problema p en un tiempo polinomial, esto implica que son problemas dentro de la clase NP difíciles de resolver.[14]

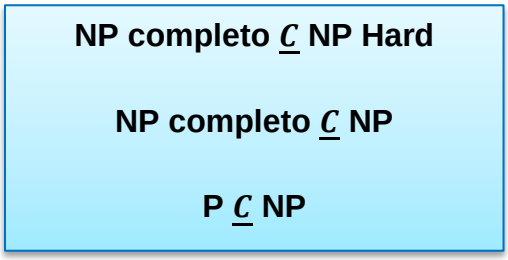
NP Hard (NP difícil o complejo) son problemas similares a los anteriores pero muy difíciles de resolver que requieren en su mayoría tiempo exponencial para ser solucionados.

---

[13] PAPADIMITRIOU, Christos. Computational Complexity. Addison-Wesley. 1994

[14] HAMALAINEN, Wilhemiina. Class NP, NP-complete, and NP-hard problems. [en linea]. (2006). [Consultado 12 Feb. 2013]. Disponible en < http://cs.joensuu.fi/pages/whamalai/daa/npsession.pdf >

Figura 2. Relaciones existentes entre las clases de complejidad



Fuente: Adaptado de HAMALAINEN, Wilhemiina. Ibid. p.7.

En general los problemas de Scheduling pertenecen a la clase NP, sólo pueden ser resueltos por algoritmos determinísticos con comportamiento exponencial ya que el tiempo necesario para solucionarlos crece exponencialmente conforme aumenta el tamaño de las entradas.[15]

2.4. CLASIFICACIÓN DEL PROBLEMA DE SCHEDULING

La clasificación de los problemas de Scheduling depende de diferentes factores. Basado en el momento de llegada del trabajo al proceso; el problema de Scheduling es estático si todos los trabajos llegan al mismo tiempo, pero si los trabajos entran al proceso de forma intermitente se denomina dinámico. Basados en las políticas de inventario, un problema puede ser abierto o cerrado, abierto si

[15] JAIN, Anant y MEERAN, Sheik. Deterministic job-shop scheduling: Past, present. En: European Journal of Operational Research. Vol 113 (1999) p. 390-434.

todos los productos son made-to-order y cerrado si todos los productos son made to stock.[16]

Existe otras clasificaciones para los problemas de Scheduling, determinístico si los tiempos de procesamiento de los trabajos y la disponibilidad de las máquinas se conocen previamente o probabilístico en caso contrario; basados en el ambiente de producción pueden ser de una o múltiples etapas, en los ambientes de una sola etapa cada trabajo pasa por una sola máquina, mientras que en los ambientes múltiples, cada trabajo consiste de diferentes operaciones que deben ser procesadas en diferentes máquinas, por último los problemas pueden catalogarse de acuerdo con el patrón de flujo de los recursos y los trabajos.

Principales problemas de Scheduling

Se pueden distinguir varios problemas de Scheduling basados en el patrón de flujo, los más estudiados son: el Flow Shop Scheduling Problem (FSSP), el Job Shop Scheduling Problem (JSSP) y el Open Shop Scheduling Problem (OSSP).

2.4.1. Job shop Scheduling Problem

El JSP consiste en un conjunto de trabajos J_i , $1 \leq i \leq n$, que deben ser procesadas en un conjunto limitado de M máquinas donde cada trabajo J_i tiene asociado un tiempo de operación y n_j operaciones las cuales deben programarse en un orden determinado, el propósito es encontrar la secuencia de trabajos para cada máquina que permita optimizar un objetivo de desempeño que puede ser el

[16] ZOBOLAS G; TARANTILIS, Christos y IOANNOU, George. Exact, Heuristic and Meta-heuristic Algorithms for Solving Shop Scheduling Problems. En: XHAFA, Fatos y ABRAHAM, Ajith. Metaheuristics for Scheduling in Industrial and Manufacturing Applications. Springer Berlin Heidelberg, 2008. p. 1- 40

makespan (tiempo total en el que todos los trabajos han sido ejecutados), la tardanza total o el tiempo de flujo, mientras satisface la siguientes restricciones y suposiciones:

- Cada máquina puede procesar solo un trabajo a la vez.
- Cada trabajo sólo puede ser procesado por una máquina al tiempo.
- La secuencia de visita de los trabajos a las máquinas está definida.
- Todos los trabajos deben ser procesados por cada máquina solo una vez y hay máximo m operaciones por cada trabajo.
- Las máquinas siempre están disponibles y nunca se interrumpen.
- El tiempo de procesamiento de todas las operaciones es conocido.

De acuerdo con la notación de Conway[17] el criterio de makespan puede indicarse por *n/m/J/*C_{max} y según Graham et al. puede indicarse por *J//*C_{max}.

Tomando en cuenta las restricciones establecidas y la función objetivo que se desee optimizar, el modelo matemático para el Job Shop Scheduling Problem se compone de restricciones lineales y una función objetivo lineal, adicionalmente presenta variables de decisión de naturaleza binaria.

El modelo para la minimización del makespan se representa a continuación:

Sea t_{ij}: tiempo de inicio de cada operación

J: Conjunto de n trabajos a ser procesados

M: Conjunto de m recursos o máquinas

[17] CONWAY, Richard W; MAXWELL, William L y MILLER, Louis W. Theory of scheduling. Addison-Wesley, 1967.

O_{ij}: Operación del trabajo J_i que debe ser procesado en la máquina M_j por un período ininterrumpido de tiempo τ_{ij}.

Minimizar $C_{max} = max(t_{ij} + \tau_{ij})$: $\forall J_i \in J, M_j \in M$

Sujeto a:

- Tiempos de inicio

  $t_{ij} \geq 0 \quad \{i,p\} \in J \quad \{j,h\} \in M$

- Restricción de precedencia

  $t_{ij} - t_{ih} \geq \tau_{ih}$ Si $O_{ih}\, precede\ a\ O_{ij}$

- Restricción disyuntiva

  $t_{pj} - t_{ij} + K(1 - y_{ipj}) \geq \tau_{ij} \qquad y_{ipj} = 1, si\ O_{ij}\, precede\ a\ O_{pj}$

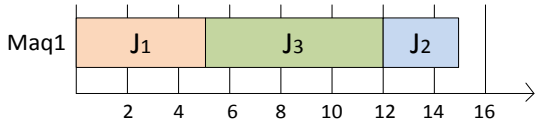
  $t_{ij} - t_{pj} + K(y_{ipj}) \geq \tau_{pj} \qquad y_{ipj} = 0,\ en\ otro\ caso$

Donde $K > \left(\sum_{i=1}^{n}\sum_{j=1}^{m}\tau_{ij} - \min(\tau_{ij})\right)$

2.4.1.1. Representación de la solución

La solución del JSP es usualmente presentada mediante el ***diagrama de Gantt***, este permite visualizar las operaciones a través de la línea de tiempo, es un gráfico bidimensional compuesto por bloques rectangulares, un eje vertical y horizontal que representa una línea de tiempo; cuando los bloques simbolizan trabajos u operaciones y el eje vertical corresponde a las máquinas se dice que es un diagrama de Gantt orientado a la máquina y cuando los bloques corresponden a máquinas y el eje vertical a trabajos u operaciones es un diagrama de Gantt orientado al trabajo. La Figura 3 muestra un diagrama de Gantt orientado a la máquina de tres trabajos programados en una máquina sobre la línea de tiempo.

Figura 3 Diagrama de Gantt para tres trabajos



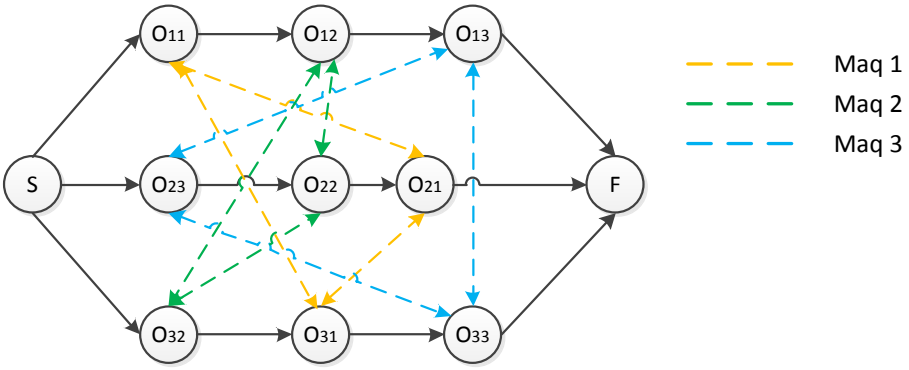
Roy y Sussmann[18] propusieron el ***grafo disyuntivo*** como representación del JSP, un conjunto de nodos N representa las operaciones a ser procesadas en un conjunto de máquinas M. Existen dos nodos ficticios, el nodo ficticio inicial S es la fuente o source y el nodo ficticio final F se llama sumidero o sink, el tiempo de procesamiento de cada operación está representado por el peso ponderado de cada nodo, O_{ij} es la operación j del trabajo i, A es el conjunto de arcos dirigidos o conjuntivos (se grafica como líneas completas) y representan las restricciones de precedencia de cada trabajo, las limitaciones de capacidad están representadas por el conjunto E de aristas no dirigidas o disyuntivas (se muestra como líneas punteadas), se vincula cada miembro de E con un par de arcos disyuntivos que comparten una máquina común, por lo tanto dos operaciones al ser procesadas por la misma máquina no pueden ejecutarse simultáneamente.[19]

La Figura 4 muestra un grafo disyuntivo para un problema de 3 máquinas y 3 trabajos, cada máquina está representada por arcos disyuntivos de color que conecta las operaciones que se relacionan con ella.

[18] ROY, Bernard y SUSSMANN. B. Op. cit.

[19] ZOBOLAS, G; TARANTILIS, Christos y IOANNOU, George. Op. cit. p. 6

Figura 4. Grafo Disyuntivo para un problema 3x3



Un Schedule factible corresponde a la selección de un arco disyuntivo al que se le asigna una dirección, la selección está completa cuando todos los arcos están dirigidos y el resultado es un gráfico acíclico.[20]

El makespan de un Schedule factible está determinado por la ruta más larga que siguen las operaciones desde el nodo S hasta el nodo sumidero F llamada ruta crítica, esta ruta consta de una operación inicial que arranca en un tiempo 0 y el valor del makespan lo determina el mayor tiempo de todas las rutas que llegan a F.

El problema de minimización del makespan se centra en encontrar el conjunto de arcos disyuntivos que minimizan la longitud de la ruta crítica.

[20] PINEDO, Michael L. Job Shops (Deterministic). En PINEDO, Michael L. Scheduling: Theory, Algorithms, and Systems. Springer New York, 2008. p. 179-216

2.4.1.2. Criterios de optimización para el JSP

Los problemas de scheduling se evalúan a través de criterios de optimización, a continuación se presentan las funciones con las que se puede evaluar el problema del JSP, y se explican algunas de las funciones comúnmente usadas.

Definiendo C_1, C_2, …, C_n como el tiempo de terminación de todos los trabajos en un Schedule, las medidas de desempeño son: el tiempo máximo de terminación (C*max*), el retraso máximo (L*max*), la tardanza máxima (T*max*), el costo máximo (*fmax*), el tiempo total de terminación ($\sum C_i$), el tiempo total de terminación ponderado ($\sum w_i C_i$), la carga total de la máquina ($\sum C_{max}^{(k)}$), el número de trabajos tardíos ($\sum U_i$) y el costo total ($\sum f_i$).[21] Se definen asi:

- $Cmax=max\{Ci\}$
- $Lmax=\max_{1\le i\le n}\{L_i\}=\max_{1\le i\le n}\{C_i-d_i\}$
- $Tmax=\max_{1\le i\le n}\{T_i\}=\max_{1\le i\le n}\{\max\{0,L_i\}\}$
- $Fmax=\max_{1\le i\le n}\{f_i(C_i)\}$ *donde f1, f2,..., fn indican las funciones de costo.*
- $\sum Ci=\sum_{i=1}^{n} C_i$
- $\sum wiCi=\sum_{i=1}^{n} w_i C_i$
- $\sum C_{max}^{(k)}=\sum_{i=1}^{m} C_{\max}^{(k)}$ *donde* $C_{max}^{(k)}$ *indica el tiempo máximo de terminación de todos los trabajos asignados a la máquina Mk,* $1\le k\le m$
- $\sum Ui=\sum_{i=1}^{n} U_i$ *donde* $Ui=0$ *si* $Li\le 0$ *y* $Ui=1$ *si* $Li>0$

[21] GAWIEJNOWICZ, Stanislaw. Op cit. p. 43

- $\sum fi = \sum_{i=1}^{n} f_i(C_i)$ *donde f1, f2,…, fn indican las fuciones de costo*.

Retraso Máximo (Maximun Lateness)

Para un trabajo i el retraso máximo Lmax está dado como la diferencia entre *Ci* (el tiempo de finalización del trabajo i) y la fecha de vencimiento *di*, cuantifica lo tarde o temprano que se ha terminado un trabajo para su entrega. Si el valor de Lmax es mayor que 0 se llama Tardanza (*Ti*), es decir que el trabajo terminó después de la fecha de entrega; pero si el valor Lmax es negativo se denomina prontitud y significa que el trabajo se entregó antes de la fecha de vencimiento[22]

$$L\max = \max_{1<i<n} \{Ci - di\}$$

Si hay fechas de vencimiento para cada puesto de trabajo, el número de trabajos retrasados es una medida de desempeño propicia.

Tardanza ponderada (Weighted Tardiness)

La tardanza de un trabajo es el tiempo en el que se excedió la fecha de entrega proyectada. Con un tiempo Ci y di asociado a cada trabajo i, se incurre en *wi* (penalización por unidad de tiempo si el trabajo i finaliza después de di), la tardanza total ponderada *Tmax* está dada por[23]

$$Tmax = \sum_{i=1}^{n} wiTi$$

La tardanza de un trabajo es Ti= max (0, Ci-di).

[22] ZOBOLAS, G; TARANTILIS, Christos y IOANNOU, George. Op. cit. p.4

[23] BÜLBÜL, Kerem. A hybrid shifting bottleneck-tabu search heuristic for the job shop total weighted tardiness problem. En: Computers & Operations Research. Vol 38, 2011. p. 967-983.

Basado en este cálculo se puede determinar la tardanza promedio ponderada, definida como

$$\bar{T}max = \frac{1}{n}\sum_{i=1}^{n} wiTi$$

Tiempo de flujo total (Total Flow Time)

El tiempo de flujo está dado como el tiempo desde que llega un trabajo hasta la finalización de su última operación, es decir, el tiempo que permanece en el sistema. Es la diferencia entre el tiempo de finalización Ci y el instante de llegada Ri; también se puede calcular como el tiempo de espera qi más el tiempo de procesamiento p_i. Por tanto el tiempo de flujo para un trabajo i es representado como [24]

$$F_i = C_i - R_i \;\; o \;\; F_i = p_i + q_i$$

El tiempo de flujo total está dado por la sumatoria de cada F_i

$$F = \sum_{i=1}^{n} Fi$$

[24] JENSEN, Mikkel T. Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures. En: Applied Soft Computing, Vol. 1 (2001) p. 35-52

3. METODOLOGÍAS DE SOLUCIÓN PARA PROBLEMAS DE SCHEDULING

Existen dos enfoques para resolver un problema de Scheduling, el primero es el que usa métodos exactos o completos, el otro enfoque se divide en métodos heurísticos y metaheurísticos.

3.1. ALGORITMOS EXACTOS O COMPLETOS

Estos algoritmos encuentran soluciones óptimas en un tiempo determinado, sin embargo para la mayoría de los problemas de aplicación real de Scheduling requieren un tiempo computacional extenso; existen diferentes algoritmos exactos, los más comunes en solución de problemas de Scheduling son el algoritmo Branch and Bound y métodos de descomposición.

El algoritmo ***Branch and Bound*** es el método exacto más eficiente para la solución del JSP. Su principio fundamental es la representación en forma de árbol de todas las posibles soluciones factibles de tal manera que por medio de procedimientos las características y propiedades no compatibles con la solución óptima son detectadas rápidamente y removidas del árbol. Aunque este algoritmo genera una solución óptima, tiende a ser muy lento en problemas de optimización combinatoria debido a sus altos requisitos computacionales, su principal limitación es la falta de fuertes límites para cortar las ramas del árbol de enumeración lo más pronto posible y reducir el tiempo requerido.

3.2. ALGORITMOS HEURÍSTICOS

Son métodos de aproximación que encuentran soluciones cercanas al óptimo en un tiempo computacional razonable, la forma básica de los algoritmos de

aproximación se conoce como heurística, estas se dividen en heurísticas constructivas y de búsqueda local.

3.2.1. Métodos constructivos

Son procedimientos iterativos, que construyen soluciones de forma gradual agregando partes a la solución inicial hasta completar la solución, generalmente estas partes son operaciones. La ventaja de estos métodos es su rapidez en el tiempo de cálculo frente a otros métodos de aproximación, pero generalmente hallan soluciones de menor calidad que las técnicas de búsqueda local.

3.2.1.1. Reglas de prioridad de despacho

Es la heurística constructiva más utilizada en la solución de problemas de Scheduling debido a su fácil implementación y baja complejidad, se desempeña muy bien en la mayoría de los casos. Algunas de las reglas más comunes para la solución de problemas de Scheduling se relacionan en el Cuadro 1. La implementación más importante de estas reglas para el JSP es el algoritmo de Giffler y Thompson,[25] esta es la base común de todas las reglas de prioridad, donde en cada paso se elige una operación de un conjunto de operaciones en conflicto que compiten por la misma máquina; esta selección se hace mediante una regla específica que asigna una prioridad a las operaciones y determina cuál operación del conjunto debe ser elegida para ser procesada.

[25] GIFFLER, B. y THOMPSON, G.L. Op. cit.

Cuadro 1. Reglas de despacho para problemas de Scheduling

Regla	Descripción
SOT	(Shortest Operating Time) Se ejecuta la tarea con el tiempo de operación más corto.
LOT	(Longest Operating Time) Se ejecuta la tarea con el tiempo de procesamiento más largo.
FCFS	(First come, first servend) Primera tarea en llegar, primera en ser atendida.
SPT	(Sortest Processing Time) Se elige un trabajo con el tiempo total de procesamiento más corto.
LPT	(Longest Processing Time) Le da prioridad a al trabajo con el tiempo de procesamiento más largo.
EDD	(Earliest Due Date) Asigna la prioridad más alta al trabajo con fecha de entrega más temprana.
SRPT	(Shortest Remaining Processing Time) Se elige la operación con el tiempo más corto de procesamiento de las tareas restantes.
LRPT	(Longest Remaining Processing Time) Se elige la operación con el mayor tiempo de procesamiento de las tareas restantes.
RANDOM	Selecciona un trabajo aleatoriamente de la cola de espera.
LORPT	Ejecuta la operación con la cola más alta y menor tiempo de procesamiento.
SNRO	Da prioridad a la operación con el menor número de operaciones posteriores.
LOS	Ejecuta las operaciones con el mayor tiempo de proceso de operación posterior.

Fuente: Adaptado a partir de ZOBOLAS G, TARANTILIS, Christos y IOANNOU, George. Op cit. p.13

- **Algoritmo de Giffler y Thompson**

El método propuesto por Giffler y Thompson concentra la atención en obtener Schedules activos y el procedimiento para la generación de la planificación explora el espacio de búsqueda por medio de una estructura de árbol en el que los nodos del árbol representan los Schedules parciales, los arcos representan las posibles elecciones y las hojas del árbol el conjunto de planificaciones. En el proceso se identifica las operaciones que compiten por la misma máquina y en cada etapa se utiliza un procedimiento de numeración para resolver los conflictos de procesamiento que se presentan en los schedules parciales, generalmente son utilizadas las heurísticas de prioridad de despacho para solucionar estos conflictos.

Dada la siguiente notación[26]:

- *(i, j)* Operación del trabajo i a ejecutar en la máquina j.
- *S* es un Schedule parcial conteniendo las operaciones a programar. Comienza vacío, S=Ø.
- Ω es el conjunto de operaciones planificables. Inicia con las operaciones que no tienen predecesores. Luego elimina la operación programada y se añade su sucesor inmediato.
- $s_{(i,j)}$ es el tiempo más temprano en el que la operación (i, j) que pertenece a Ω puede iniciar. Comienza en $s_{(i,j)}=0$
- $p_{(i,\ j)}$ es el tiempo de procesamiento de la operación (i, j).

[26] SARMIENTO, Ardila, 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, 85 p. Tesis (Ingeniera Industrial).Universidad Industrial de Santander. Facultad de Ingeniería físico mecánicas. Escuela de Estudios Industriales y Empresariales.

- $f_{(i,\,j)}$ es el tiempo más temprano en el que la operación (i, j) que pertenece a Ω puede finalizar, $f_{(i,\,j)} = s_{(i,\,j)} + p_{(i,\,j)}$.

El procedimiento para obtener un Schedule activo consta de los siguientes pasos:

1. Comenzar con un Schedule parcial nulo, es decir, S=Ø y el conjunto de operaciones que no tienen predecesores Ω.
2. Hallar los tiempos $f_{(i,\,j)}$ de las operaciones que pertenecen al conjunto Ω y determinar el mínimo, denotado como $f^* = min_{(i,\,j)\in\Omega}\{f_{(i,j)}\}$ y la máquina m^* a la que corresponde esa operación con el mínimo valor f^*. Si existe más de una operación con el mismo tiempo $f_{(i,\,j)}$, se escoge una de ellas de manera aleatoria.
3. Para cada operación (i', j') Є Ω que requiera la máquina m^* y para la cual $s_{(i,j)} < f^*$ calcular un índice de prioridad de acuerdo a la regla de prioridad elegida. Escoger la operación con el índice más pequeño y agregarla a S. Asignar a $s_{(i,j)}$ el tiempo de finalización de (i, j).
4. Si se ha generado un Schedule completo, se detiene el algoritmo, de lo contrario, se elimina la operación (i, j) del conjunto Ω y añadir el sucesor inmediato en Ω y retornar al paso 2 hasta que se haya formado el Schedule completo.

Un ejercicio de aplicación del algoritmo G&T se muestra en el Anexo A.

3.2.1.2. Algoritmo cuello de botella móvil

Otra heurística usada para la solución del JSP es el algoritmo cuello de botella móvil propuesto por Adams et al[27] y mejorado por Balas et al.[28] Esta heurística descompone el problema en subproblemas de máquinas individuales, en cada iteración, un subproblema crítico es establecido y solucionado; el algoritmo primero optimiza la programación de la máquina cuello de botella inicial, después de esto, se imponen nuevas restricciones sobre las máquinas restantes, se identifica y se programa la nueva máquina cuello de botella entre las que no se han secuenciado y así sucesivamente hasta que todas las máquinas se han programado. En la minimización del makespan ninguna regla única muestra superioridad, la solución de calidad desmejora conforme aumenta el tamaño del problema.

Para desarrollar la heurística y obtener schedules factibles para el JSP, se considera un conjunto M que contiene i máquinas, un conjunto Mo al que pertenecen las máquinas que se han secuenciado, los tiempos de procesamiento de cada trabajo en cada máquina, las restricciones de precedencia y un grafo disyuntivo en el que se han establecido todos los arcos conjuntivos pero no se han determinado los disyuntivos.

El procedimiento para generar una solución aplicando este método se describe a continuación y se amplía en el Anexo B.

---

[27] ADAMS, Joseph, BALAS, Egon y ZAWACK, Daniel. The shifting bottleneck procedure for the job-shop scheduling. En: Management Science. Vol. 34. No3 (1988) .p. 391–401

[28] BALAS, Egon y VAZACOPOULOS, Alkis. Guided local search with shifting bottleneck for job-shop scheduling. En: Management Science. Vol. 44. No 2. (1998) p. 262-275

1) Condiciones iniciales

Inicialmente el conjunto Mo está vacío; el grafo disyuntivo del problema solo tiene los arcos conjuntivos y se establece el makespan Cmax como la ruta más larga en el grafo disyuntivo actual.

2) Análisis de las máquinas a planificar

En cada iteración cada máquina j del conjunto M se analiza como un problema independiente 1|rj|Lmax, para cada trabajo a ser procesado en esta máquina se determinan los tiempos rji (tiempo de liberación del trabajo i en la máquina j y es el camino más largo en el grafo desde el nodo fuente(S) hasta el nodo Oji) y el tiempo de finalización dji,

dji= Cmax-L ((Oji), F)+pji; donde L ((Oji), F) es el camino más largo desde Oji hasta el nodo sumidero F y pji es el tiempo de procesamiento del trabajo i en la máquina j.

Para determinar cuál es la máquina crítica se resuelve el problema de programación en cada una, se determina el tiempo de terminación Ci y el retraso Li.

Cuadro 2. Fórmulas para calcular dji, Ci y Li para cada trabajo i

J1	J2	…	Ji
pj1	pj2	…	pji
rj1	rj2	…	rji
dj1=Cmax-L((Oj1),F)+pj1	dj2=Cmax-L((Oj2),F)+pj2	…	dji= Cmax-L((Oji),F)+pji
C1= pj1+ rj1	C2= pj2+C1, si C1>rj2 pj2+rj2, si C1<rj2	…	Ci= pji+Ci-1, si Ci-1>rji pji+rji, si Ci-1<rji
L1= C1- dj1	L1= C2- dj2	…	Li= Ci-dji

Fuente: Adaptado de CASTILLO, G. y FANDIÑO, O. p.98

Se secuencian los trabajos en cada máquina. En cada secuencia de la máquina *j* se elige el mayor valor de Li, este será el Lmax para esa secuencia; cuando todas las combinaciones de trabajos han sido evaluadas para esa máquina se elige el menor Lmax de todas las programaciones con el propósito de obtener el mínimo Lmax (j) (mínimo retraso generado por la secuencia elegida en la máquina j).

Este procedimiento se realiza para i combinaciones o teniendo en cuenta alguna regla de prioridad o criterio que permita establecer un orden en la secuencia de los trabajos en la máquina analizada.

3) Elegir la máquina cuello de botella

Una vez determinado el Lmax para cada elemento de M, la máquina m que tenga el mayor Lmax es considerada el cuello de botella y entra a ser parte del conjunto Mo, se insertan en el gráfico los arcos disyuntivos de la secuencia de trabajos para esa máquina.

$$L\max(m) = \max_{j \in \{M - Mo\}} (L\max(j))$$

Se determina el nuevo makespan del problema como

Cmax (Mo U m)=Cmax (Mo)+Lmax (m)

4) Reprogramar las máquinas ya planificadas

Para cada máquina j $\in$ {Mo – m} eliminar en el grafo los arcos disyuntivos correspondientes y reformular cada máquina como un subproblema con los tiempos rji y dji determinados a partir de la ruta más larga en el grafo que contiene los arcos de m. Determinar la secuencia que minimiza Lmax(j), es decir, en cada secuencia de la máquina j se elige el mayor valor de Li; este será el Lmax para

esa máquina, cuando todas las secuencias han sido evaluadas y todas la máquinas analizadas, insertar los arcos de cada máquina en el grafo.

5) Criterio de parada

Si todas las máquinas han sido programadas es decir Mo=M, finaliza el algoritmo, de lo contrario, el procedimiento se repite desde el paso 2.

Esta heurística permite encontrar soluciones factibles, además se adapta fácilmente a diferentes problemas y funciones fitness; sin embargo, en la práctica el procedimiento es complejo y requiere de alto tiempo computacional a medida que aumenta el número de máquinas y trabajos.

3.2.2. Algoritmos de búsqueda local

Los algoritmos de búsqueda funcionan con estructuras de vecindario; para cada vecindario las soluciones con el mejor valor de la función objetivo se denominan óptimos locales. Dos Schedules son vecinos, si uno puede obtenerse mediante la modificación del otro. Estos métodos parten de una solución inicial y en cada iteración se reemplaza una parte o toda la solución por una mejor de un conjunto de soluciones vecinas evaluadas. Para llevar a cabo estos reemplazos los algoritmos de búsqueda local realizan unos movimientos que conducen a la formación de nuevas soluciones en el mismo vecindario, los movimientos más comunes son el 2-Opt, el intercambio 1-1 y el intercambio 1-0.[29] La principal desventaja de estos métodos es que quedan atrapados fácilmente en óptimos locales dejando espacios de solución sin explorar, para contrarrestar esto, los

[29] TARANTILIS, Christos y KIRANOUDIS, Chris. A list-based threshold accepting method for the job-shop scheduling problems. En: International Journal of Production Economics . Vol 77, No 2. (2002); p.158-171.

nuevos métodos de búsqueda local integran metaheurísticas que dirigen el proceso de búsqueda.

3.3. ALGORITMOS METAHEURÍSTICOS

Esta metodología explora de forma eficiente y efectiva los espacios de búsqueda a través de movimientos evitando quedar atrapado en óptimos locales, es decir, son una forma inteligente de explorar los espacios de búsqueda. El término metaheurísticas fue introducido por Glover[30]; es una estrategia que guía y modifica otras heurísticas proporcionando soluciones más robustas que las encontradas por optimización local, aunque son más difíciles de implementar ya que necesitan información especial acerca del problema a resolver.

Estos métodos utilizan dos estrategias para direccionar la búsqueda, diversificación e intensificación, la diversificación busca la exploración efectiva de todos los posibles vecindarios del espacio de solución y la intensificación se centra en la exploración de subespacios de solución más reducidos.

Diferentes algoritmos Metaheurísticos se han propuestos para resolver el problema de Scheduling, algunos de ellos son:

Umbral de Aceptación propuesto por Dueck y Scheuer.[31]

[30] GLOVER, Fred. Future paths for integer programming and links to artificial intelligence. En: Computers & Operations Research. Vol 13 (1986); p. 533-549

[31] DUECK, Gunter y SCHEUER, Tobias. Threshold accepting. A general purpose optimization algorithm appearing superior to simulated annealing. En: Journal of Computational Physics. Vol. 90. No 1. (1990).p. 161-175.

Búsqueda básica local donde el vecindario de una solución es explorado mediante un conjunto de movimientos y devuelve el óptimo local.[32]

Búsqueda exploratoria local, representada principalmente por el procedimiento de búsqueda adaptativa GRASP propuesto por Feo y Resenden[33], búsqueda de vecindario variable VNS propuesto por Hasen y Mladenovi[34] y búsqueda local iterativa introducida por Stützle.[35]

Entre los más populares para dar solución al problema de Job Shop Scheduling se encuentran:

3.3.1. Optimización por enjambre de partículas (PSO)

Es un método análogo al movimiento colectivo de las aves al volar y su transmisión de información, fue introducido por Eberhart y Kennedy[36]. Las aplicaciones de este método en JSP son relativamente recientes, aunque el PSO puede usarse generalmente con variables continuas, y por tanto no es adecuado para variables discretas, la mayoría de las investigaciones proponen variaciones al

[32] ZOBOLAS, G; TARANTILIS, Christos y IOANNOU, George. Op. cit. p. 16.

[33] FEO, Thomas y RESENDEN, Mauricio. Greedy randomized adaptive search procedures. En: Journal of Global Optimization. Vol. 6. No 2. (1995) p. 109-133.

[34] HANSEN, Pierre y MLADENOVIC, Nenad. Variable neighborhood search: Principles and Applications. En: European Journal of Operational Research. Vol. 130. No 3. (2001). p. 449-467

[35] STÜTZEL, Thomas. Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications. 1999.

[36] EBERHART, Russell y KENNEDY, James. A new optimizer using particle swarm theory, En: Proceedings of the Sixth International Symposium on Micro Machine and Human Science. Nagoya (1995) p. 39-43.

PSO original para que pueda ser aplicado al JSP. Lian et al.[37] presentan un algoritmo PSO con la característica de convertir un dominio continuo en uno discreto.

3.3.2. Optimización por Colonia de Hormigas (ACO)

Introducido por Dorigo[38], es un algoritmo inspirado en el comportamiento de colonias de hormigas, los agentes (hormigas) construyen soluciones iterativamente guiándose por un sendero de información que depende del problema tratado. En la solución del JSP es bastante limitado aunque ha sido aplicado con éxito en otro problema de optimización combinatoria como el TSP.

3.3.3. Redes Neuronales (NN)

Es un modelo inspirado en el comportamiento del cerebro, intentan imitar el aprendizaje humano asemejando la transmisión de información a través de redes de neuronas interconectadas[39]. En los últimos años ha decrecido el interés de su implementación para JSP. Los primeros trabajos realizados en este campo obtuvieron resultados de baja calidad, solo el estudio de Sabuncuoglu y Gurgun[40] generó buenos resultados en los problemas de referencia.

---

[37] LIAN, Zhigang; JIAO, Bin y GU, Xingsheng. (2006) A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. En: Applied Mathematics and Computation. Vol. 183. No. 1 (2006). p. 1008-1017

[38] DORIGO, Marco. Optimization, Learning and Natural Algorithms. Milán, 1992. Tesis PhD. Departamento de Electrónica. Politécnico de Milano.

[39] FOO, Yoon y TAKEFUJI, Y. Stochastic neural networks for solvin job shop scheduling: part 1. Problem representation. En: IEEE International Conference on Neural Networks. San Diego, USA. Vol. 2. (1988).p 275-282

[40] SABUNCUOGLU, Ihsan y GURGUN, Burckaan. A neural network model for scheduling problems. En: European Journal of Operational Research. Vol. 93. No. 2 (1996) .p. 288-299

3.3.4. Recocido Simulado (SA)

Propuesto por Kirkpatrick et al. [41] , es un algoritmo análogo al proceso de recocido (enfriamiento lento) de un sólido en metalúrgica, donde el proceso inicia a alta temperatura y hace vibrar las moléculas que se desplazan caóticamente hasta adoptar estructuras mínimas al disminuir la temperatura; en SA la posición de las moléculas son valores de la secuencia de tareas por procesar mientras que el papel de la energía lo asume la función objetivo a minimizar. Jain y Meeran[42] reconocen que esta técnica no puede alcanzar buenas soluciones rápidamente en problemas de JSP, por eso las investigaciones recientes combinan esta técnica con otros métodos con el fin de reducir el tiempo de cálculo.

3.3.5. Búsqueda Tabú (TS)

Es un enfoque iterativo definido por primera vez por Fred Glover[43] usado con frecuencia para la solución del JSP. Consiste en un proceso de búsqueda de soluciones que son modificadas en cada nueva posición y almacenadas en memoria con el fin de guiar inteligentemente el proceso de búsqueda, el término tabú se debe a que el algoritmo convierte a las regiones ya exploradas en prohibidas para evitar examinarlas nuevamente. Permite intensificar la búsqueda en áreas que históricamente tienen buenos resultados o dirigir la búsqueda a regiones no exploradas. El método propuesto por Nowicki y Smutnicki[44] es una de

[41] KIRKPATRICK, Scott; GELATT, C y VECCHI, M.P. Optimization by simulated annealing. En: Science. Vol. 220. No 4598 (1983). p. 671-680.

[42] JAIN, Anant y MEERAN, Sheik. Op. cit.

[43] GLOVER, Fred. Op. Cit.

[44] NOWICKI, Eugeniusz y SMUTNICKI, Czeslaw. A fast taboo search algorithm for the job shop problem. En: Management Science Vol. 42.(1996), p.797-813

las implementaciones más exitosas de TS que permite alcanzar buenas soluciones rápidamente

3.3.6. Computación evolutiva

El principio de Charles Darwin “Supervivencia del más apto” es usado como punto referencia para introducir el concepto de **Computación evolutiva (EC)**. La evolución es un proceso de optimización en el cual el objetivo es mejorar la capacidad de un organismo o sistema para sobrevivir en entornos cambiantes, dinámicos y competitivos. El término evolución y su concepto ha sido la causa de muchos debates por siglos. En el área específica de la evolución biológica han sido aceptadas las teorías Darwinianas y Lamarckiana; la primera tiene su enfoque en la supervivencia y reproducción de los individuos con las mejores características para que sean transmitidas a su descendencia, la segunda teoría establece su idea principal en la adaptación de los individuos a lo largo de su vida y la conservación de aquellos rasgos que sean importantes para ellos. Estos principios coinciden en la transferencia y conservación de los caracteres de los individuos en su descendencia.

Estos principios los resumen las técnicas de computación evolutiva en algoritmos que pueden ser usados para buscar soluciones óptimas a un problema. En estos algoritmos de búsqueda se tiene cierta cantidad de posibles soluciones en el problema y el objetivo es encontrar la mejor solución en una magnitud de tiempo determinado. Si el espacio de búsqueda tiene pocas soluciones posibles, todas las soluciones podrían ser exploradas en un tiempo razonable y se podría encontrar la solución óptima. No obstante, esta búsqueda resulta poco práctica si el espacio de búsqueda crece en gran tamaño.

Un aspecto clave por lo cual se puede distinguir un algoritmo de búsqueda evolutivo de uno tradicional, es que este es basado en la población. A través de la adaptación de las generaciones continuas de un gran número de individuos un

algoritmo evolutivo realiza una búsqueda eficiente directamente. La EC resuelve los problemas basada en el computador, a través de sistemas que utilizan modelos computacionales de procesos evolutivos (selección natural, supervivencia del más apto y reproducción) así como los componentes propios que poseen los sistemas de computación para la ejecución del algoritmo.

El proceso de búsqueda evolutiva está influenciado por los siguientes componentes de un **Algoritmo Evolutivo (EA):**

- **Codificación** de soluciones del problema como un cromosoma.
- **Función** para evaluar el fitness o supervivencia de los individuos.
- **Inicialización** de la población inicial.
- **Operadores** de selección.
- **Operadores** de reproducción.

Diferencias entre computación evolutiva y optimización clásica

Los algoritmos de optimización clásicos han demostrado ser exitosos en problemas lineales, cuadráticos, fuertemente convexos y otros problemas especiales, incluso mejores que los EA, sin embargo los algoritmos evolutivos se han mostrado eficientes en problemas discontinuos, no diferenciales y multimodales.

La optimización clásica y la computación evolutiva difieren significativamente en el proceso de búsqueda y la información acerca del espacio de búsqueda que utilizan para guiar el proceso[45].

[45] ENGELBRECHT, Andries. Computational Intelligence An introduction, University of Pretoria South Africa, Editorial Wiley, 2º edición, 2007.

Cuadro 3. Diferencias entre computación evolutiva y optimización clásica.

	COMPUTACIÓN EVOLUTIVA	OPTIMIZACIÓN CLÁSICA
PROCESO DE BÚSQUEDA	✓ Utiliza reglas de transición probabilística. ✓ Aplica búsqueda paralela en el espacio de búsqueda. ✓ La búsqueda parte de un conjunto diverso de puntos iniciales, que permite la búsqueda paralela de un gran área del espacio de búsqueda	✓ Usa reglas determinísticas para moverse de un punto a otro en el espacio de búsqueda. ✓ Usa una búsqueda secuencial en el espacio de búsqueda. ✓ La búsqueda comienza desde un punto y avanza sucesivamente hasta alcanzar el óptimo.
INFORMACIÓN DE SUPERFICIE DE BÚSQUEDA	✓ No utiliza la información derivada, los valores fitness de los individuos son usados para guiar la búsqueda.	✓ Utiliza la información derivada usualmente, de primer o segundo orden del espacio de búsqueda que guíe la ruta hacia el óptimo.

3.3.6.1. Desarrollo histórico de la computación evolutiva

En el caso de la computación evolutiva hay diversos paradigmas históricos que han servido de base para varias actividades realizadas en este campo y son las diferentes formas en las cuales son implementados los componentes de los EA.

- **Programación genética**

La programación genética (GP) es una técnica que se ha vuelto aún más popular en los últimos tiempos. Muchos investigadores toman este paradigma como una rama especializada de los algoritmos genéticos.

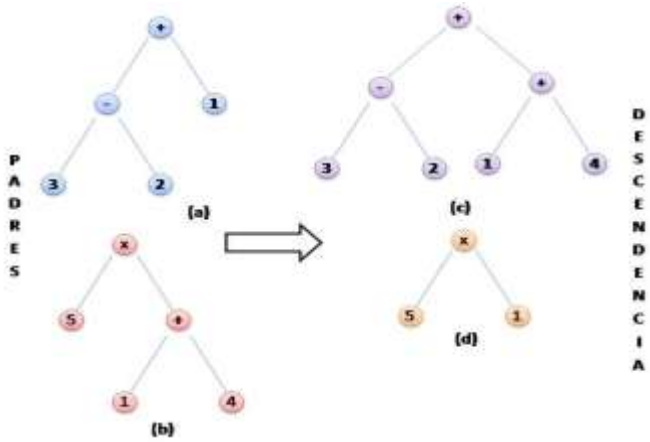
Koza[46] fue quien desarrolló originalmente la GP para programas de computador evolucionados. La principal diferencia entre estos dos paradigmas (GA y GP) es el esquema usado en la representación. Mientras que los algoritmos genéticos utilizan vectores o cadenas como formas de representación, la programación genética usa una representación en árbol. En un programa genético generalizado, la representación más utilizada es un árbol de funciones y valores de tamaño variable (Figura 5). El operador de reproducción comúnmente usado es el cruce de subárboles (*subtree crossover*), que consiste en el intercambio de un subárbol completo entre dos padres.

- **Programación evolutiva**

La programación evolutiva (EP) surge en la década de los 60 gracias a las investigaciones realizadas por Fogel (1962) como un enfoque alternativo a la inteligencia artificial. La EP toma la idea de representar los fenotipos de los individuos como unas máquinas de estado finito que sean capaces de responder a los estímulos del entorno y el desarrollo de operadores que permitan efectuar un cambio en el comportamiento y en la estructura durante el tiempo. Este proceso consiste en encontrar un conjunto de comportamientos óptimos de un espacio de comportamientos observables.

[46] KOZA. John R, Genetic programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems, En: Technical Report STAN-CS90-1314, Department of Computer Science, Stanford University, 1990.

Figura 5. Cruce de subárboles de padres (a) y (b) para formar una descendencia (c) y (d).



Fuente: Adaptado a partir de Sivanandam, S.N. Deepa, S.N. Introduction to Genetic Algorithms (2008).

Las representaciones usadas en la programación evolutiva se adaptan generalmente al dominio del problema, el vector de longitud fija de valor real es una de estas representaciones, que a su vez tiene aplicación en optimización de funciones continuas.

El método básico de este algoritmo evolutivo es la selección de todos los individuos en la población que serán los N padres, seguido de la mutación de cada uno de ellos y obtener la descendencia N, por último se realiza la selección probabilística basada en el fitness para formar la siguiente generación (Figura 6).

- **Estrategia evolutiva**

La estrategia evolutiva (ES) considera los aspectos de la evolución tanto genética como fenotípica, sin embargo, el énfasis se encuentra en el comportamiento de los individuos de forma similar a la programación evolutiva, igualmente su representación se hace por medio de un vector de longitud fija de valor real. Cada individuo es representado por bloques de construcción genética y un conjunto de

parámetros de estrategia que modela el comportamiento de los individuos en el entorno, en este proceso las características genéticas son controladas por los parámetros de estrategia.

Figura 6. Cuatro componentes principales de la programación evolutiva

El paradigma de la ES fue originalmente presentado por Rechenberg[47] (1960), no obstante, la mayor exploración en el tema fue realizada por Schwefel[48]. La ES incluye la recombinación dentro de los operadores de su sistema, lo cual permite la diferenciación del presente paradigma con los otros algoritmos evolutivos.

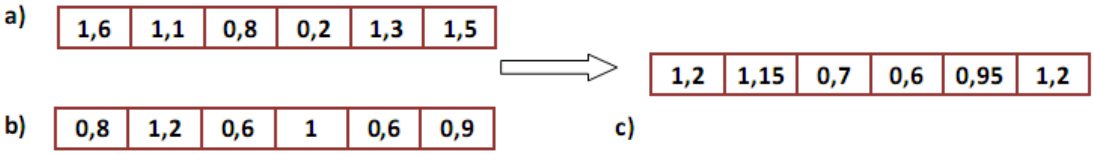
En el proceso de inicialización de la ES los padres son seleccionados de forma uniformemente aleatoria, es decir, no se realiza basado en el valor fitness; la nueva generación se realiza a través del uso del operador de recombinación y los individuos que sobreviven son seleccionados de forma determinística.

[47] RECHENBERG, Ingo. Evolutions strategie: Optimierung technischer Systeme nach Prinzipien der Biologischen Evolution. Frammann-Holzboog, Stuttgart, 1973.

[48] SCHWEFEL, Hans-Paul. Evolutions strategie und numerische Optimierung, Technical University Berlin, 1975.

El principal operador de reproducción es la *mutación Gaussiana*, en el que un valor aleatorio de una distribución Gaussiana es agregado a cada elemento de un vector de individuos para crear una generación. Otro operador importante es la *recombinación intermedia*, en esta fase los vectores de los dos padres son promediados y se origina una nueva descendencia.

Figura 7. Recombinación intermedia entre padres para formar una descendencia



- **Evolución diferencial**

La evolución diferencial (DE) es considerada un método de búsqueda local directa y ha sido usada para diversos estudios teóricos basados en su comportamiento estocástico, este paradigma de los algoritmos evolutivos desarrollado por Storn y Price [49] en 1995 ha demostrado eficiencia y simplicidad en sus procesos de optimización, además es aplicado en la resolución de problemas complejos principalmente aquellos de naturaleza continua.

Los algoritmos evolutivos tienen diversas similitudes entre ellos, por ejemplo la DE también es una estrategia de búsqueda basada en la población, sin embargo, se diferencia en el proceso de búsqueda ya que es guiado a partir de la información de distancia y dirección de la población actual. Una de las características atribuidas a este método es el uso de vectores de prueba después de ser aplicada la mutación, que consiste en generar nuevos vectores de parámetros añadiendo la

[49] STORN, Rainer y PRICE Kenneth, Op. cit.

diferencia ponderada entre dos vectores de población a un tercer vector, posteriormente se genera una descendencia usando el operador *de* cruce.

Los dos parámetros de control en la evolución diferencial son: el factor de escala (β) y la probabilidad de recombinación, debido que estos influyen en el rendimiento de DE, adicionalmente a estos parámetros se toma en cuenta el tamaño de la población, porque éste se encuentra asociado a la capacidad de exploración del algoritmo.

El proceso se desarrolla en la primera etapa con la población inicial, la cual se genera de forma aleatoria uniforme, seguida de la mutación, el cruce y la selección que son los operadores que intervienen en el desarrollo del algoritmo.

Algoritmos Culturales

Los algoritmos culturales (CA) se han basado en los principios de la evolución cultural para adaptarlos y utilizarlos en la solución de problemas diversos y complejos que buscan la optimización. Este tipo de algoritmo desarrollado por Reynolds permite que las sociedades se adapten a entornos cambiantes con una velocidad superior a la evolución biológica. En términos computacionales, la base de este EA se encuentra en el modelo de un origen de datos que influye en el comportamiento de todos los individuos dentro de una población.

Durham define la cultura como "*un sistema de fenómenos conceptuales codificado simbólicamente, que son transmitidos social e históricamente dentro y entre los grupos".*[50]

[50] DURHAM, William. Co-Evolution: Genes, Culture and Human Diversity. Stanford University Press, 1992.

De acuerdo al marco presentado por Reynolds[51] la evolución cultural puede describirse en dos niveles: micro-evolutivo que se basa en los componentes genéticos que son heredados de los padres y el macro-evolutivo es el conocimiento adquirido por los individuos.

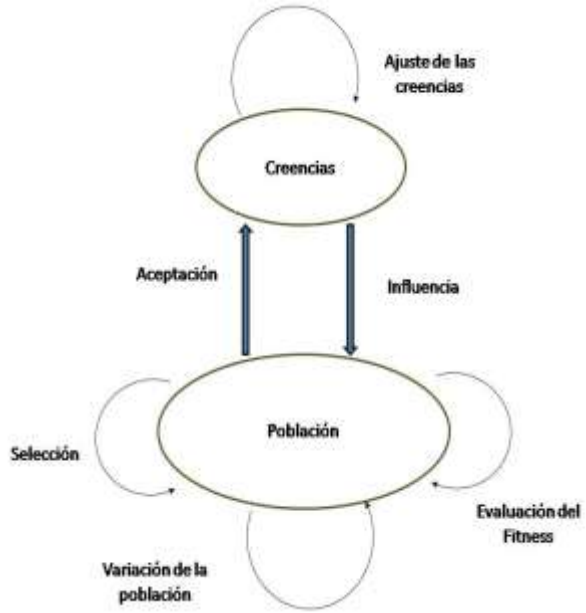
Un algoritmo cultural mantiene dos espacios de búsqueda: el primero es el espacio de la población (representado en niveles genéticos y fenotípicos) que contiene una serie de individuos de la misma forma como lo mantienen los demás algoritmos pertenecientes a la computación evolutiva. El segundo es el espacio de creencias (modelando el componente cultural) donde se almacena la información cultural y los conocimientos que han adquirido los individuos en generaciones previas, también es conocido como *meme pool; e*l meme es una unidad de información que es transmitida por medio de un comportamiento. Estos espacios evolucionan de forma paralela creando un protocolo de comunicación de tal manera que ambos tengan influencia en el otro.

3.3.6.1. Algoritmo genético

Uno de los paradigmas de la computación evolutiva más importante y utilizado en la solución de problemas generales son los algoritmos genéticos (GA), han arrojado buenos resultados en tiempo razonable a problemas que tienen un espacio de búsqueda de gran tamaño debido a su clasificación como problemas de tipo NP Hard.

[51] REYNOLDS, Robert G. An Introduction to Cultural Algorithms: Proceedings of the Third Annual Conference on Evolutionary Programming, En: World Scientific, River Edge, New Jersey, (1994), p. 131–139.

Figura 8. Algoritmo cultural de espacios de población y cultural



Fuente: Adaptada de ENGELBRECHT, Andries. Computational Intelligence An introduction. 2007

Holland en su libro “Adaptation in natural and artificial systems”[52] publicado en el año 1975 propone el algoritmo genético como una heurística basado en la supervivencia del más apto, estableciendo una guía de los algoritmos genéticos que se conocen hoy en día. Según el autor, este tipo de algoritmos permiten explorar una mayor cantidad de posibles soluciones que los programas tradicionales.

El GA es un método de búsqueda estocástico y de búsqueda global que incluye dentro de su proceso la aleatoriedad como factor fundamental, ya que es utilizado

[52] HOLLAND, John. Adaptation in natural and artificial systems. University of Michigan Press. Ann arbor, 1975.

en los procesos de selección y reproducción; no tiene requerimientos particulares, por tanto se puede aplicar a cualquier problema. No obstante, se desarrolla tomando en cuenta una representación específica y los operadores genéticos para el desarrollo de las posteriores descendencias.

El procedimiento general que se lleva a cabo en el algoritmo genético inicia con la selección de la población, la cual es realizada mediante un conjunto de cromosomas generados de forma aleatoria. Al realizar este procedimiento el espacio de búsqueda se incrementa y se pueden obtener planificaciones que no arrojan buenas soluciones. Por lo tanto, existe el interés de utilizar heurísticos para generar la población inicial y de esta manera se puedan obtener Schedule activos con el propósito de reducir el espacio de búsqueda y hallar al menos una solución óptima. Una heurística utilizada con frecuencia para construir Schedules factibles en el problema del JSP debido a su baja complejidad en la implementación es propuesta por Giffler y Thompson[53].

Los cromosomas, conocidos también como genotipos son los códigos de una solución que se representan generalmente como una cadena de números y cada uno de ellos es denominado *gen*. Se evalúa el fitness o función objetivo de todos los cromosomas para formar una nueva población, este ciclo de una población a otra es llamado generación. De acuerdo al valor de la función objetivo se seleccionan los cromosomas actuales y posteriormente son aplicados los operadores de cruce y mutación. Después que se ha desarrollado durante varios ciclos el sistema genético codifica el cromosoma que representa la solución óptima del problema.

[53] GIFFLER, B. y THOMPSON, G.L. Op.cit.

Los resultados obtenidos por los algoritmos genéticos suelen ser moderadamente buenos, sin embargo la calidad de las soluciones podrían mejorarse mediante la hibridación del GA con un método de búsqueda local, que permita intensificar en la búsqueda del espacio de soluciones. Este tipo de algoritmos híbridos suelen denominarse algoritmos meméticos.

- **Representación de la población**

La representación predeterminada de los algoritmos genéticos son los cromosomas, en el cual es codificada la información de acuerdo a los parámetros del problema. Una de las representaciones más usadas debido a su sencillez es la binaria, en la que el cromosoma da como resultado un vector de unos y ceros.

Para el problema del JSP se debe establecer una representación de tal manera que un individuo esté determinado por la planificación de todas las operaciones a ejecutar en el problema. Muchos autores han realizado trabajos con diferentes tipos de representación para el JSP clasificadas de acuerdo al enfoque de codificación, unas pueden ser directas y otras indirectas[54](Cuadro 4).

La representación basada en operaciones es la más utilizada en los estudios reportados en la literatura, se realiza la codificación del cromosoma en el que, cada gen representa una operación y el vector queda conformado por una secuencia de todas las operaciones.

[54] CHENG, Runwey; GEN, Mitsuo Y TSUJIMURA Yasuhiro. A tutorial a survey of job shop scheduling problems using genetic algorithms: representation. En: Computers and Industrial Engineering, Vol.30 No4, (1996) p 983-997.

Cuadro 4.Tipos de representación para el problema del Job shop Scheduling.

Enfoque directo	Enfoque indirecto
• Basada en operaciones • Basada en Trabajos • Basada en relaciones de parejas de trabajo • Basada en tiempos de completamiento • Claves aleatorias	• Basada en reglas de prioridad • Basada en listas de preferencia • Basada en grafo disyuntivo • Basada en máquinas

- **Operadores genéticos**

Selección

Naturalmente los individuos que integran las siguientes generaciones son aquellos que provienen de padres con mayor aptitud es decir que provean mejor fitness, medido por el valor de la función objetivo. Para seleccionar los individuos que formarán parte de la siguiente generación se emplea un esquema de selección del GA el cual determinará cuáles son los individuos que formarán parte de la nueva generación. Los tres esquemas de selección más común son: selección por torneo, selección aleatoria y selección de ruleta.

Selección por Torneo: Este tipo de selección consiste en elegir dos cromosomas del grupo que está disponible de forma aleatoria, en seguida se realiza la comparación entre ellos mediante la evaluación de la función de valor (fitness). El cromosoma que tenga el mejor valor es seleccionado y puede ser reproducido.

Selección Aleatoria: La selección aleatoria como su nombre lo indica consiste en seleccionar al azar los padres de un cromosoma de los que están disponibles. Se

evalúa cada uno de ellos tomando en cuenta el valor fitness y el que tome un valor por debajo del establecido es removido inmediatamente de la población. Este tipo de selección se utiliza para introducir la aleatoriedad en casos donde la población empiece a converger a una solución sub-óptima.

Selección de ruleta: Otro esquema de selección comúnmente utilizado, denominado también “selección de aptitud proporcional”, consiste en seleccionar los mejores individuos de acuerdo a la aptitud (fitness) relativa que presente el cromosoma respecto a la población y que éstos tengan mayor probabilidad de ser reproducidos. Es decir, si el valor fitness de un cromosoma es el doble comparado con otro, este tendrá el doble de probabilidad de ser reproducido respecto al otro, no obstante, como el proceso es probabilístico también pueden ser elegidos los individuos con peor fitness.

Operador de cruce

Se encuentra en la clasificación de operadores de reproducción, este proceso consiste en la creación de nuevas y mejores generaciones o descendencia a partir de los padres representados en cromosomas a través del intercambio de características genéticas provenientes de los mismos, ya que no es suficiente con el proceso de selección realizado anteriormente. Existen tres clases principales de cruces:

- *Asexual:* Un padre genera una descendencia.
- *Sexual:* Se genera una o dos descendencias a partir de dos padres.
- *Multi-recombinación:* Se generan dos o más generaciones a partir de dos o más padres.

Se debe tomar en cuenta el tipo de problema que se desea resolver para elegir el método adecuado de ejecución del operador de cruce, debido que éste puede ser limitado en problemas con restricciones basados en Schedule, como el JSP. Se

han desarrollado diversas formas de realizar la operación de recombinación de los cromosomas, dentro de las más conocidas se encuentra el cruce en un punto, cruce en dos puntos y el cruce uniforme.

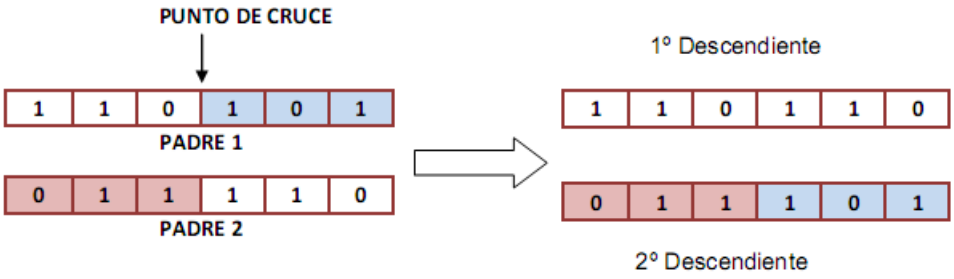
En el caso más sencillo donde la representación es binaria se pueden utilizar uno de los más comunes que es el cruce en un punto; este consiste en escoger una posición del cromosoma de forma aleatoria el cual se va a dividir en dos partes el vector de representación, se asigna la primera parte del vector del primer padre y la segunda parte del segundo padre al primer descendiente, para generar el segundo descendiente se realiza de forma similar, en este caso la primera parte del segundo padre y la segunda parte del primero son combinadas (Figura 9). La descendencia generada pertenece a la siguiente población.

Operador de mutación

Este operador consiste en la creación de nuevos individuos alterando la combinación genética a través de la modificación de uno de sus genes una vez realizada la etapa del operador de cruce con el fin de agregar diversidad a las características de la población. La mutación se realiza como un complemento del operador de cruce aplicando cierta probabilidad P_m a cada gen. De acuerdo a la naturaleza del problema a considerar, los investigadores diseñan su propio método para ejecutar este último operador, utilizando una probabilidad muy pequeña de manera constante durante todo el proceso con el fin de evitar una alteración significativa de la descendencia y conservar características de los padres, otra alternativa usada por investigadores es el uso de una alta probabilidad de mutación al inicio del proceso e ir decreciendo este valor de forma exponencial en toda su ejecución.

Este operador resulta muy importante en los algoritmos genéticos debido que aumenta la diversidad de la población, explorando mayor parte de la región factible y espacios de búsqueda que no hayan sido explorados con anterioridad.

Figura 9. Operador de cruce en un punto



3.4. MÉTODOS HÍBRIDOS

Los algoritmos híbridos metaheurísticos se basan en la combinación de varias metaheurísticas con el fin de aprovechar las fortalezas y eliminar las limitaciones individuales, generando métodos de búsqueda más potentes y flexibles. Se pueden distinguir tres formas principales de hibridación[55], la primera es llamada *componente de intercambio*; es la hibridación de métodos basados en población con métodos de búsqueda local, la segunda forma es la *búsqueda cooperativa* e implica el intercambio de información entre dos o más metaheurísticas diferentes y por último *la integración de algoritmos metaheurísticos y métodos sistemáticos*[56]*;* su implementación exitosa consiste en la combinación de metaheurísticas y restricciones de programación. La mayoría de los algoritmos híbridos pertenecen a la primera categoría de hibridación en la que varios componentes y características de la solución están compartidos entre dos o más métodos heurísticos y metaheurísticos.

[55] BLUM, Christian y ROLI, Andrea. Op. cit.

[56] PESANT, Gilles y GENDREAU, Michel. A view of local search in Constraint Programming. En: Principles and Practice of Constraint Programming. (1996)

4. ALGORITMOS MEMÉTICOS

Los algoritmos meméticos (MA) son un método de optimización en el que se combinan conceptos de distintas metaheurísticas con el fin de aprovechar las ventajas que proporcionan de acuerdo a estrategias implícitas en el enfoque de cada una de ellas. Dentro de estas metaheurísticas se encuentran los paradigmas que conforman la computación evolutiva (basados en la población) encargados de realizar la exploración del espacio de búsqueda, es decir, actúa como un método de búsqueda global y los métodos de búsqueda local desarrollan la explotación.

Este concepto fue introducido por Moscato y Norman basado en la idea expresada por Dawkins respecto al concepto de los memes como un análogo al tema genético desde la perspectiva de evolución cultural[57]. Dado que el algoritmo memético combina métodos de algoritmos evolutivos en su desarrollo, este adquiere algunos aspectos y características provenientes de los mismos; por lo tanto el procedimiento es similar al que se lleva a cabo en los EA para la solución de problemas, no obstante, difieren en algunos conceptos y extienden los términos derivados principalmente de los algoritmos genéticos (Cuadro 5), los cuales son utilizados con mayor frecuencia en la hibridación con el método de búsqueda local para la aplicación del algoritmo memético.

Cabe destacar que no existe un procedimiento sistemático para abordar el diseño de un MA en cierto problema ya que podría tener discrepancias con resultados teóricos ya conocidos.[58]

---

[57] MOSCATO, Pablo. Op. Cit.

[58] MOSCATO, Pablo Y COTTA, Carlos, Una Introducción a los Algoritmos meméticos, En: Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, (2003), p 131-148.

La idea central de los MA está fundamentada en las mejoras individuales de las soluciones de agentes (solución/es al problema aunado al mecanismo de mejora local) que se interrelacionan entre sí en un proceso que contiene fases de cooperación y competiciones del tipo poblacional.[59]

La fase competitiva es realizada en el proceso de selección y reemplazo, en el primero, varía la distribución de los *agentes* existentes y el segundo tiene la tarea de limitar el tamaño de la población, es decir, se permite la entrada de nuevos agentes mediante la eliminación de algunos y enfoca la búsqueda.

En la fase de cooperación se crean nuevos agentes, debido que es ejecutada en la aplicación de los operadores de cruce y mutación empleando información extraída de los agentes que han sido recombinados y en algunas ocasiones de información externa (como consecuencia del proceso de mutación).

Cuadro 5 Comparación entre aspectos característicos de los Algoritmos Genéticos y Algoritmos Meméticos.

Algoritmos genéticos	Algoritmos meméticos
Codificación: esquemas, cadenas lineales, alfabetos predefinidos. *Individuo:* Solución al problema *Cruce:* Intercambio no guiado *Mutación:* Introducción aleatoria de nueva información	*Representación:* Formas, no linealidad, cercanía al problema. *Agente:* Solución/es al problema + mecanismo de mejora local. *Recombinación:* Intercambio guiado de información. *Mutación:* Introducción sensible de nueva información. *Mejora local:* Aprendizaje lamarckiano.

Fuente: Adaptado de MOSCATO, P Y COTTA, C, Ibid. p.136

[59] Ibid. p. 131

4.1. ESTADO DEL ARTE

Desde principios de los años cincuenta diversas investigaciones se han centrado en resolver el Job Shop Scheduling Problem (JSP); no se conoce con certeza a quien se debe el origen de este problema. Jackson en 1956 generalizó el algoritmo de Flow shop de Johnson[60](1954) para construir un algoritmo Job Shop, Akers y Friedman[61] en 1955 aplicaron un enfoque algebraico para representar secuencias de procesamiento, posteriormente Giffler y Thompson [62] (1960) plantearon un algoritmo de reglas de prioridad de despacho, Roy y Sussmann[63] (1964) propusieron una representación mediante el grafo disyuntivo y en 1969 E. Balas aplicó un enfoque enumerativo basado en este grafo.

El problema del Job Shop Scheduling se presentó por primera vez como se conoce en su forma actual en el libro Industrial Scheduling editado por Muth y Thompson en 1964 constituyendo la base de las investigaciones posteriores en el problema del Job Shop, su utilidad práctica así como su complejidad computacional al ser NP-Hard (Garey y Johnson, 1979)[64] lo convierten en un

[60] JOHNSON S.M. Optimal two- and three-stage production schedules with setup times included. En: Naval Research Logistics Quarterly. Vol. 1. No. 1 (1954) p.61–68.

[61] AKERS, Sheldon y FRIEDMAN Joyce. A Non-Numerical Approach to Production Scheduling Problems. En: Journal of the Operations Research Society of America. Vol. 3. No. 4 (1955) p. 422-429.

[62] GIFFLER, B. y THOMPSON, G.L. Algorithms for solving production scheduling problems. En: Operations Research. Vol. 8. No 4 (1960), p.487-503

[63] ROY, Bernard y SUSSMANN, B. Les probl`emes d'ordonnancement avec constraintes disjonctives.En:Technical Report. Paris:SEMA, 1964.

[64] GAREY, Michael y JOHNSON, David. Computer and intractability: *A Guide to the Theory of NP-Completeness* .1979.

interesante tema de estudio; en Frutos y Tohmé [65] se presentan diferentes publicaciones relacionadas con su presentación (Sadeh y Fox ,1995; Chinyao y Yuling, 2009; Della Croce et al., 2011; Lin et al., 2011), con el fin de aplicar algoritmos como reglas de prioridad particulares (Panwalker y Iskander, 1977), cuellos de botella móviles (Adams et al., 1998), recocido simulado (Van Laarhoven et al., 1992), Búsqueda Tabú (Dell Amico y Trubian, 1993; Armentano y Scrich,2000; Nowicki y Smutnicki, 2005), Algoritmos Genéticos (Zalzala y Flemming, 1997), Optimización por Colonia de Hormigas (Merkle y Middendorf,2001), la selección clonal (Corte's Rivera et al., 2003), Solución guiada multipunto de búsqueda constructiva (Beck (2007), algoritmos híbridos (Zhang et al. 2008), etc.

En la última década se han desarrollado estudios que intentan resolver el problema del Job Shop utilizando metaheurísticas con enfoque evolutivo o algoritmos meméticos. Los algoritmos meméticos (MA) se originan a finales de los ochenta, el término "memético" se debe a R. Dawkins[66] en 1976, basado en el término inglés "meme", que expresa una analogía del gen en el contexto de la evolución cultural, antes de que un meme sea transferido es adaptado a los pensamientos de la persona mientras que un gen es transferido como un todo, Moscato[67] relacionó este concepto con el proceso de refinamiento local y por ello en 1989 introduce por primera vez el término Algoritmo memético haciendo referencia a algoritmos evolutivos que usan búsqueda local.

[65] FRUTOS, Mariano y TOHME, Fernando. A Multi-objective Memetic Algorithm for the Job-Shop Scheduling Problem. En: Operational Research. (2012), p. 1-18.

[66] DAWKINS, Richard. The Selfish Gene. Clarendon, Oxford.1976

[67] MOSCATO, Pablo. On evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Toward Memetic Algorithm, En: Caltech Concurrent Computation Program, Report 826, Caltech, Pasadena CA. 1989.

La característica más distintiva de los MAs es la explotación de todo el conocimiento disponible sobre el problema de optimización tratado, es por eso que en ocasiones se les llama Algoritmos Evolutivos Híbridos, ya que la palabra hibridación es usada con frecuencia para denotar el proceso de incorporar el conocimiento de un problema.

En 1988 se publicó el primer algoritmo que se designó como un MA, era un híbrido de los algoritmos genéticos tradicionales y recocido simulado que buscaba solucionar el Min Eucledean Traveling Salesman Problem (TSP). Los MAs han sido usados con éxito en diferentes áreas y diversos problemas; Moscato y Cotta[68] presentan problemas de optimización combinatoria como el de la mochila (Gallardo, Cotta y Fernandez, 2005), problemas de enrutamiento (Boudia et al., 2007; Creput y Koukam, 2008), asignación cuadrática (Drezner, 2008; Tang et al., 2007), árbol de expansión (Fischer y Merz, 2007; Rocha et al., 2006), entre otros. En Planeación, programación, horarios y manufactura, programación de una única máquina (Maheswaran et al.,2005; Moscato, et al., 2004), Job shop (Caumond et al.,2008; Gonzales et al.,2006; Qian et al.,2008; Yang et al.,2008), Flow shop (Franca et al,.2005; Li, Wang y Liu, 2008;Pan, Wang y Qian, 2008; Liu,Wang y Jin, 2007; Moscato et al.,2004), fabricación flexible (Amaya et al.,2008; Chen y Chen,2008), además se han aplicado en bioinformática, electrónica, telecomunicaciones, ingeniería, minería de datos y mecanismos de aprendizaje.

A continuación, se presenta una recopilación de las investigaciones relacionadas con el JSP y algoritmo memético para conocer las contribuciones previas y actuales tomadas como referencia para la ejecución del tema propuesto.

[68] MOSCATO, Pablo y COTTA, Carlos A Modern Introduction to Memetic Algorithms. En: GENDREAU, Michel y POTVIN, Jean-Yves. Handbook of Metaheuristics. Springer US, 2010. p. 141-183

- ***Hasan et al.***[69] desarrollan un Algoritmo Genético (GA) y un algoritmo memético para la solución del JSP, proponen tres versiones de algoritmo memético utilizando como técnica de búsqueda local tres reglas de prioridad, reordenamiento parcial (RP), intercambio restringido (RS) y reducción del espacio (GR). Resuelven 40 problemas de referencia con ambos algoritmos y los comparan con algoritmos conocidos en la literatura, los resultados muestran que el MA (GR-RS) en comparación con el GA y los demás algoritmos propuestos mejora la calidad de la solución y reduce el tiempo de cálculo.

- ***Frutos y Tohmé***[70] proponen un algoritmo memético multiobjetivo combinando un Algoritmo Evolutivo Multiobjetivos (MOEA) y Recocido Simulado Multi Objetivos (MOSA) para dar solución al JSP, prueban diferentes MOEAs demostrando que el NSGAII y el SPEAII superan en gran medida a sus predecesores; los resultados muestran que es un algoritmo útil para la solución de múltiples objetivos del JSP.

- ***Qian et al.***[71] buscan solucionar el JSP multiobjetivo mediante un algoritmo memético basado en evolución diferencial llamado MODEMA, proponen una regla SOV y una estrategia Metalamarkiana para la exploración y explotación de los vecindarios respectivamente. Los resultados se comparan con la Búsqueda

[69] HASAN, Kamrul et al. Memetic algoJrithms for solving job-shop scheduling problems. En: Memetic Computing. Vol. 1. No 1. (2009), p. 69-83

[70] FRUTOS, Mariano y TOHME, Fernando. Op. cit

[71] QIAN et al. Scheduling multi-objective job shops using a memetic algorithm based on differential evolution. En: The International Journal of Advanced Manufacturing Technology. Vol. 35. No 9-10 (2008), p. 1014-1027

Local Genética Multiobjetivo de Ishibuchi y Murata (IMMOGLS) demostrando que el MODEMA es más eficaz en la solución del JSP.

- ***Gao et al.***[72] proponen un algoritmo memético eficiente para minimizar el makespan en un problema Job Shop, utilizan la capacidad de búsqueda global del GA y un algoritmo de búsqueda local, presentan dos estructuras de vecindario, intercambio e inserción sobre la ruta crítica. Los resultados demuestran la eficiencia del MA sobre otros algoritmos presentados en la literatura.

- ***González et al.***[73] buscan minimizar el makespan en un problema Job Shop con secuencias dependientes de tiempos de preparación mediante un algoritmo memético que combina un algoritmo genético y un algoritmo de búsqueda tabú con una estructura de vecindad N^s. Los resultados se comparan con el algoritmo cuello de botella móvil propuesto por Balas et al. (2008) y un GA hibrido propuesto por González et al. (2008), considerando el modelo de evolución Baldwiniana y el Lamarkiano; los autores demostraron que el MA supera las demás técnicas y que la evolución Lamarkiana es mejor.

- ***Frutos et al***[74]***.*** presentan un algoritmo memético para la solución del Job Shop Flexible mediante un algoritmo genético NSGAII *(NonDominated Sorting*

[72] GAO, Liang et al. An efficient memetic algorithm for solving the job shop scheduling problem. En: Computers & Industrial Engineering. Vol 60. No 4 (2011), p.699-705

[73] GONZÁLEZ, Miguel; VELA, Camino y VARELA, Ramiro. A competent memetic algorithm for complex scheduling. En: Natural Computing. Vol. 11. No 1. (2012), p. 151-160

[74] FRUTOS, Mariano; OLIVERA, Ana Carolina y TOHMÉ, Fernando. A memetic algorithm based on a NSGAII scheme for the flexible job-shop scheduling problem. En: Annals of Operations Research. Vol. 181. No. 1 (2010), p. 745-765

Genetic Algorithm II) que actúa sobre dos cromosomas y en la etapa genética usa el Recocido simulado como búsqueda local. Se demuestra que el MA propuesto concurre rápidamente a la zona cercana a la solución, genera soluciones buenas a bajo costo, así como una distribución uniforme de los resultados.

- ***Zuo et al.***[75] introducen un algoritmo híbrido que combina el sistema inmune artificial AIS con búsqueda tabú para resolver el JSP, el AIS se basa en la selección clonal para hacer la exploración de soluciones, la búsqueda tabú usa un vecindario pequeño basado en una gráfico disyuntivo que explota las buenas soluciones mediante intercambio en la ruta crítica. Se evaluaron 43 problemas de referencia y los resultados se compararon con los obtenidos por otros algoritmos, demostrando que el algoritmo propuesto AISTS produce soluciones mejores y de calidad.

- ***Gao et al.***[76] presentan un algoritmo híbrido para resolver el JSP Flexible con restricciones de disponibilidad no fijas (FJSP-nfa), el algoritmo genético usa un método de representación parcial para mostrar sólo una sección de la solución candidata y dejar la parte no expresada a criterio de la heurística, aplica las operaciones genéticas de cruce y mutación para mejorar la herencia, se definen dos tipos de vecindario basado en la ruta crítica y se integra la búsqueda local, se simularon seis problemas de referencia, tres para FJSP y tres para el FJSP-

[75] ZUO, Xingquan; WANG, Chunlu y TAN Wei. Two heads are better than one: an AIS- and TS-based hybrid strategy for job shop scheduling problems. En: The International Journal of Advanced Manufacturing Technology. Vol. 63. No 1-4 (2012), p. 155-168

[76] GAO, Jie; GEN, Mitsuo y SUN, Linyan. Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. En: Journal of Intelligent Manufacturing. Vol. 17. No 4 (2006), p. 493-507

nfa, los resultados indican la eficacia del algoritmo híbrido en la solución de ambos problemas.

- ***Jouglet et al.***[77] buscan solucionar el Flow Shop Scheduling Híbrido de tareas con multiprocesadores mediante un algoritmo memético, proponen un GA y un algoritmo de programación con restricciones branch and bound para la búsqueda local, el algoritmo propuesto se basa en el algoritmo de Orguz y Ecan con algunas diferencias que hacen del MA más eficiente, se evalúa el rendimiento de los tres algoritmos (GA; Algoritmo basado en Branch and Bound y MA), los resultados demuestran que el algoritmo planteado genera mejores respuestas que los otros dos algoritmos simulados**.**

- ***Tang et al*** [78] **.** Mediante la combinación del algoritmo genético y la metaheurística PSO (optimización de partículas por enjambre) proponen un algoritmo híbrido. Presentan un nuevo método de inicialización basados en el esquema de Kacem, tomando en cuenta las características del problema del Job Shop Flexible haciendo uso de los operadores genéticos. Aplicando el método propuesto por los autores se logra mejorar el tiempo de ejecución comparado con otros métodos existentes.

[77] JOUGLET, Antoine; OĞUZ, Ceyda y SEVAUX, Marc. Hybrid Flow-Shop: a Memetic Algorithm Using Constraint-Based Scheduling for Efficient Search. En: Journal of Mathematical Modelling and Algorithms. Vol. 8. No 3 (2009), p. 271-292

[78] TANG, Jianchao, et al. A Hybrid Algorithm for Flexible Job-shop Scheduling Problem. En: Procedia Ingeniering. Vol.15, (2011), p. 3678-3683.

- ***Caumond et al.***[79] abordan el problema de JSP con retrasos mediante la modelización del problema como un grafo disyuntivo y un algoritmo memético, favoreciendo la exploración del espacio de búsqueda investigando en las soluciones no factibles durante el proceso. El framework presentado en el estudio permite abordar una gama amplia de problemas de Job Shop sin tener que limitarse a los problemas con tiempos de retraso.

- ***Qingdao-er-ji y Wang***[80] presentan un Algoritmo Genético Híbrido, denominado HGA para el problema del JSP, en el cual introducen un operador de selección mixto para aumentar la diversidad de la población y un nuevo algoritmo que permite encontrar la ruta crítica. Un operador de búsqueda local es diseñado con el fin de mejorar la capacidad de búsqueda superando a la del algoritmo genético. El algoritmo propuesto demuestra eficacia debido a que encuentra soluciones óptimas en instancias pequeñas y soluciones mejores o iguales en problemas grandes comparados con otros algoritmos.

- ***Yang et al.***[81] proponen un algoritmo memético, mediante la hibridación de un algoritmo apoyado en la selección clonal basada en la proliferación de anticuerpos y la incorporación de la búsqueda local con el algoritmo de recocido simulado en el vecindario de Nowicki y Smutnicki que permitan explorar y explotar los espacios de búsqueda en el problema del JSP, que tiene como

[79] CAUMOND, Anthony; LACOMME, Philippe y TCHERNEV, Nikolay. A memetic algorithm for the job-shop with time-lags. En: Computers & Operations Research. Vol 35, (2008), p. 2331-2356.

[80] QUIN-DAO-ER-JI, Ren y WANG, Yuping. A new hybrid genetic algorithm for job shop scheduling problem. En: Computers and Operations Research. Vol. 39, (2012) p.2291-2299.

[81] YANG, et al. Jin-hui. Clonal Selection Based Memetic Algorithm for Job Shop Scheduling Problems. En: Journal of Bionic Engineering. Vol. 5, (2008), p. 111-119.

objetivo minimizar el makespan. Los resultados arrojados en la simulación muestra la efectividad en el algoritmo al llegar a una buena solución en tiempos de ejecución muy cortos.

- ***Chiang et al.***[82] presentan un algoritmo memético llamado NNMA. Este algoritmo es integrado con el algoritmo multiobjetivo evolutivo NSGA-II y la heurística para mejorar el procedimiento de búsqueda local NEH en el problema de Flow Shop con permutación con el objetivo de minimizar el makespan y el tiempo de flujo total. Se comparan los resultados del modelo propuesto con algoritmos de Minella et al (2008) obteniendo resultados similares al simularlo en instancias pequeñas, sin embargo el NNMA supera el algoritmo mencionado cuando resuelve instancias de mediana y grande escala.

- ***Cheng et al.***[83] introducen un algoritmo hibrido entre reglas de despacho, la heurística cuello de botella móvil (SB) y algoritmo evolutivo. Realizan el procedimiento en dos etapas, utilizando el enfoque de algoritmo memético usando el algoritmo evolutivo y las reglas de despacho en la primera y en la segunda etapa realizan un proceso de re-optimización mediante la heurística SB que permite mejorar la calidad de la solución arrojada en la primera etapa para problemas JSP con múltiples objetivos (makespan y tardanza total). El

[82] CHIANG, Tsung-che; CHENG, Hsueh-Chien y FU, Li-Chen. NNMA: An effective memetic algorithm for solving multiobjective permutation flow shop scheduling problems. En: Expert Systems with Applications. Vol. 38, (2011), p. 5986-5989.

[83] CHENG, Hsueh-Chien; CHIANG, Tsung-che y FU, Li-Chen. A two-stage hybrid memetic algorithm for multiobjective job shop scheduling. En: Expert Systems with Applications. Vol. 38, (2011), p. 10983-10998.

hibrido propuesto muestra un mejor rendimiento al encontrar soluciones diversas y de alta calidad tomando en cuenta varios objetivos.

- ***Zhang et al.***[84] proponen un algoritmo memético (MA) para el problema de JSP flexible en ambientes dinámicos, es decir, cuando se presentan llegadas de trabajos aleatorios. El MA se aplica en los puntos de reprogramación de trabajos para optimizar el problema, de esta forma generar un Schedule razonable. La búsqueda local presentada en el algoritmo usa cinco clases de estructuras de vecindario para mejorar la calidad de la solución. El algoritmo propuesto es eficiente respecto al objetivo (minimización del makespan y la tardanza media), aunque la influencia en el makespan es significativa cuando existen pocas llegadas de trabajos.

- ***Raeesi et al.***[85] realizan una modificación en la representación del cromosoma utilizando una representación por listas de operación de máquinas (MOL) en el cual se basa el algoritmo memético propuesto por los autores, superior a las representaciones existentes. El algoritmo evolutivo utilizado en el estudio es el GA y proponen una heurística de búsqueda local utilizando la nueva representación, este procedimiento reconsidera todas las operaciones del trabajo reasignando el orden de secuencia para mejorar la solución. Se demuestra que el método propuesto es eficiente comparado con otros

[84] ZHANG, Liping; LI, Xinyu y WEN, Long. Scheduling flexible job shop in dynamic environment based on a memetic algorithm. En: Cognitive Informatics & Computing: IEEE 11th International Conference on. (2012), p. 407-412.

[85] RAEESI, N. Mohammad Y KOBTI, Ziad. A Machine Operation Lists Based Memetic Algorithm for Job Shop Scheduling. En: Evolutionary Computation (CEC): IEEE Congress on. (2011), p. 2436-2443.

algoritmos meméticos propuestos y la representación propuesta es posible aplicarla en la mayoría de problemas de Job Shop Scheduling.

- ***Zhang et al.***[86] presenta en su trabajo un híbrido entre un algoritmo genético y el método de búsqueda local denominado Ascenso estándar para realizar la exploración y la explotación en el espacio de búsqueda, utilizando la representación basada en lista de operaciones y un nuevo tipo de Schedule llamado FAS (*Full active Schedule*). Con el fin de presentar las características más significativas de las generaciones anteriores propone un nuevo operador crossover POX. El presente método demuestra su eficacia debido a que el error medio relativo es bajo comparado con otros algoritmos reportados en la literatura.

- ***Meeran y Morshed***[87] desarrollan un framework usando una técnica híbrida entre el algoritmo genético y la búsqueda tabú (TS) con el objetivo de minimización del makespan en problemas del Job shop. El sistema descrito por los autores logra obtener soluciones óptimas o cercanas a las mejores soluciones. El algoritmo fue utilizado y comparado en distintos problemas existentes en organizaciones de la vida real arrojando un mejor desempeño. Expresaron que este algoritmo permite solucionar cualquier problema arbitrario de Job shop Scheduling en vez de enfocarse a un problema específico.

[86] ZHANG, Chaoyong; RAO, Yunqing y LI, Peigen. An effective hybrid genetic algorithm for the job shop scheduling problem. En: The International Journal of Advanced Manufacturing Technology. Vol. 39, Ed. 9-10, (2008) p. 965-974.

[87] MEERAN, Sheik y MORSHED, M. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study**.** En: Journal of Intelligent Manufacturing., Vol. 23, Ed. 3-4, (2012),p. 1063-1078.

Los documentos encontrados en la literatura acerca de la implementación de algoritmos meméticos para solucionar problemas de Scheduling, evidencian la necesidad de estudiar este método para dar solucion al JSP debido que estos enfoques obtienen buenas soluciones en tiempo razonable a un problema que es cosiderado complejo, por esta razón en esta investigación se desarrollará un algoritmo memético basado en el método propuesto por Gao et al.[88], al cual se realizan algunas modificaciones en los operadores genéticos y la búsqueda local, utilizando una estructura de vecindario con el fin de minimizar el makespan.

[88] GAO, Liang et al. Op. cit.

5. DISEÑO DEL ALGORITMO

Basado en el algoritmo planteado por Gao et al[89] para dar solución al problema del JSP, se propone un algoritmo memético modificado. El algoritmo combina: el proceso genético y la búsqueda local.

La búsqueda local implementada se basa en el trabajo de Nowicki y Smutnicki[90] y es una estructura de vecindario que permite explotar las soluciones generadas por el algoritmo genético.

5.1. DESCRIPCIÓN DETALLADA DEL ALGORITMO MEMÉTICO PROPUESTO

5.1.1. Estrategia genética para generar un Schedule factible

El procedimiento comienza al generar una población inicial que evoluciona a través del algoritmo produciendo nuevos individuos mediante el uso de los operadores genéticos cuyo fin es mejorar la descendencia y explorar el espacio de búsqueda para obtener un mejor fitness, en este caso se busca disminuir el makespan.

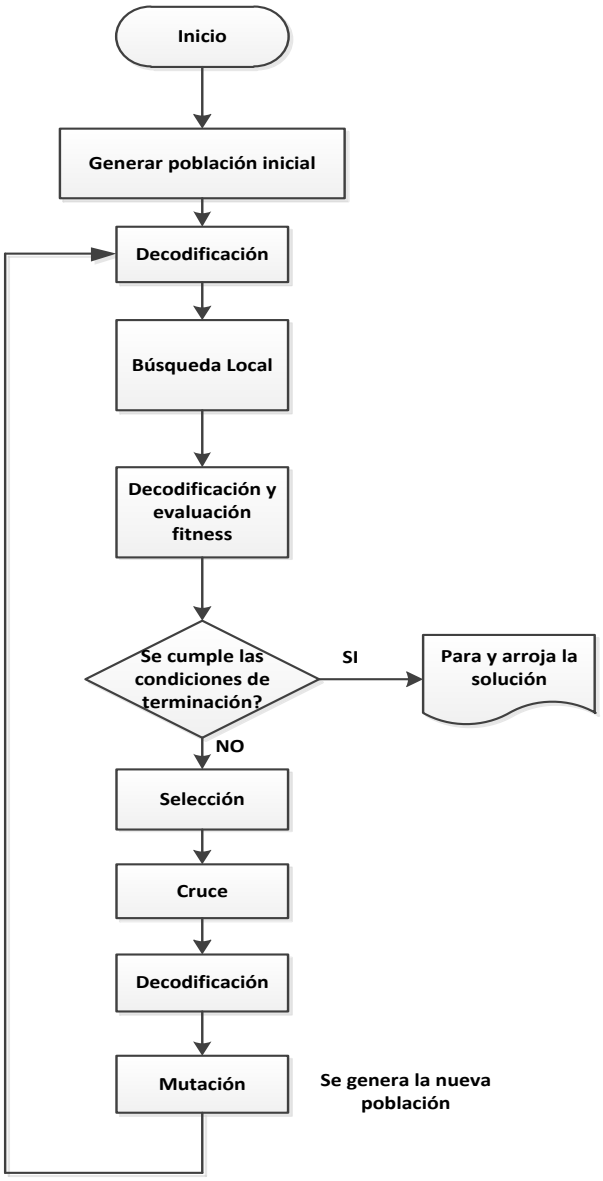
Cada individuo de la población inicial es decodificado y es aplicado el procedimiento de búsqueda local mediante una estructura de vecindario que genera soluciones vecinas; tanto la población inicial como la matriz de vecinos son evaluadas mediante el valor de la función fitness. Cada vez que se evalúa un cromosoma en cada generación el mejor valorg obtenido es guardado. Si el

[89] Ibid

[90] NOWICKI, Eugeniusz y SMUTNICKI Czeslaw. Op. cit.

criterio de parada se ha cumplido, el mejor valor guardado es la solución al problema, de lo contrario los operadores genéticos de selección, cruce y mutación se encargan de evolucionar los cromosomas en cada generación hasta que el criterio se cumpla. En la Figura 10 se muestra los pasos de esta estrategia, los cuales se discuten a continuación.

Figura 10. Diagrama de Flujo del MA propuesto



5.1.1. Representación genética de los individuos

La forma de representar a los individuos en el algoritmo genético es fundamental para la eficiencia de este, el propósito es generar soluciones factibles y así evitar hacer reparaciones a los individuos y además que los operadores genéticos usados se adapten fácilmente. Diferentes tipos de representaciones genéticas para el JSP han sido propuestas (Ver anexo C), en esta investigación se usa la representación basada en operaciones que permite codificar un Schedule como una secuencia de operaciones, la codificación se conoce como cromosoma y cada gen del cromosoma es una operación programada.

Para un problema de n trabajos y m máquinas, un cromosoma es una permutación con repetición de los trabajos, las operaciones de un mismo trabajo se representan por el número de este y aparecen en el cromosoma m veces, al leer el cromosoma de izquierda a derecha cada aparición de un mismo número indica una operación dentro de la secuencia de programación para ese trabajo. Este tipo de representación siempre genera programas factibles.

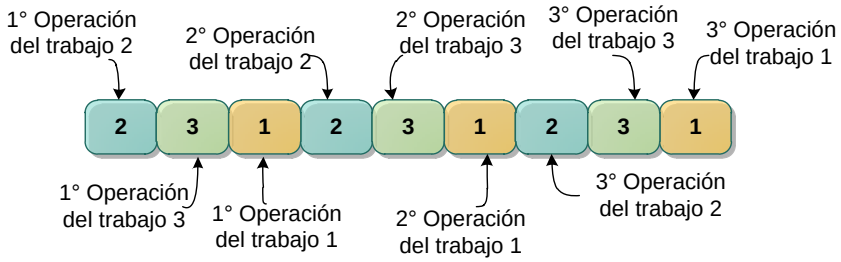
Tabla 1. Ejemplo de un problema con tres trabajos y tres máquinas.

Trabajos	Secuencia de máquinas	Tiempo de Procesamiento
J_1	m_1 - m_2 - m_3	3 -2- 5
J_2	m_1 - m_3 - m_2	3 -5- 2
J_3	m_2 - m_1 - m_3	2 -5- 3

Para el problema de la Tabla 1, se tiene el siguiente cromosoma [2 3 1 2 3 1 2 3 1], donde (1, 2 y 3) corresponden a los trabajos J_1, J_2 y J_3 respectivamente, cada trabajo se repite dentro del cromosoma tantas veces como el número de máquinas tenga el problema y representan las operaciones de cada trabajo. Al leer el cromosoma de izquierda a derecha, el primer gen 2 representa la primera

operación de este trabajo y debe ser procesada en la máquina 1, el cuarto gen es 2, indica la segunda operación de J_2 ya que es la segunda vez que se repite este número en el cromosoma y es la primera en asignarse a la máquina 3, de esta forma, el cromosoma [2 3 1 2 3 1 2 3 1] es una secuencia de O_{ij} (operación j del trabajo i), que se puede traducir a un vector de operaciones $[O_{21}\ O_{31}\ O_{11}\ O_{22}\ O_{32}\ O_{12}\ O_{23}\ O_{33}\ O_{13}]$.

Figura 11. Interpretación de un cromosoma del ejemplo de la Tabla 1.



Dos o más cromosomas pueden generarse a partir del mismo Schedule, ya que esto depende del escaneo de las operaciones en el diagrama de Gantt, pero todos ellos tendrán el mismo makespan.

Es posible realizar el proceso inverso y obtener un Schedule a partir de un cromosoma y la información del problema. El cromosoma se lee de izquierda a derecha y se traducen los genes en operaciones que son programadas teniendo en cuenta los tiempos de procesamiento y la secuencia de las máquinas, cada operación se asigna en el menor tiempo posible.

5.1.2. Generación de la población inicial

Existen diferentes métodos para generar la población inicial, heurísticas como el cuello de botella móvil, reglas de prioridad de despacho o enfoques aleatorios. Los métodos heurísticos aunque más avanzados consumen mayor tiempo

computacional, por esta razón en la presente investigación se usa la generación de individuos de forma aleatoria ya que cualquier permutación de trabajos será un Schedule factible debido al tipo de representación usada. Un individuo es un vector que contiene $n * m$ posiciones y se genera asignando de forma aleatoria el número de cada trabajo m veces, esto se repite hasta completar el tamaño de la población. Estos vectores forman una matriz que es considerada la población inicial en el algoritmo. El tamaño de la población se mantiene constante en cada generación.

5.1.3. Operador de selección

El operador de selección se encarga de elegir los individuos que se cruzarán para obtener descendencia. Los enfoques de selección comúnmente usados son: selección por torneo, selección aleatoria y selección de ruleta. En esta investigación se utiliza un esquema de selección adaptado que combina la selección por torneo y selección de ruleta. Consiste en elegir los padres mediante un porcentaje de selección de los mejores individuos de la población pero permitiendo que algunos individuos no tan buenos entren a ser parte de los seleccionados.

Una matriz de población de los mejores vecinos y los individuos iniciales son los posibles padres, esta matriz se ordena de forma ascendente de acuerdo al valor del makespan. Se aplica el porcentaje de selección a la primera mitad de la matriz ordenada que son los mejores individuos (este porcentaje por lo general se establece por encima del 70%), el complemento de ese porcentaje se aplica a la segunda mitad de la matriz, con el fin de diversificar la población de padres; de esta forma la mitad de la población es elegida para hacer parte de los progenitores. Si el número de individuos seleccionados es impar se elige un cromosoma aleatoriamente y se elimina ya que es necesario para ejecutar el

cruce, esto ocasiona que en cada generación el tamaño de la población sea un número par aun cuando el tamaño de la inicial no lo sea.

5.1.4. Operador de Cruce

Los operadores de cruce permiten intercambiar información entre dos individuos con el fin de obtener descendencia. Dos cromosomas son elegidos aleatoriamente de la matriz de padres para ser apareados y generar hijos que conservarán algunas de sus características. En esta investigación se utiliza el operador Job Order Crossover (JOX) descrito por Bierwirth[91] este método elige un subconjunto de trabajo de forma aleatoria y mantiene los genes de este trabajo en el primer hijo en la mismas posiciones que en el primer padre y los genes faltantes se toman del segundo padre en el orden en que se encuentran en este, el segundo hijo se genera de la misma forma intercambiando el papel de los padres.

No se considera una probabilidad de cruce ya que en la etapa de selección se introduce este parámetro para determinar qué individuos entrarán a ser parte de la matriz de padres, se determina que a todos los padres se les aplica el operador de cruce, es decir, se aplica con probabilidad 1. Cada pareja de padres genera dos hijos, el número de hijos al cruzar todas las parejas será igual al tamaño de la población inicial.

Para explicar el procedimiento discutido se consideran dos padres P1 y P2 (Figura 12) que generan dos hijos H1 y H2 aplicando el operador JOX como sigue:

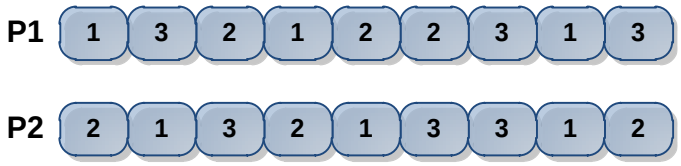
- Se elige del conjunto de números de trabajos (1,2,…, n) uno de ellos aleatoriamente, por ejemplo el 3 que corresponde al trabajo J_3.

[91] BIERWIRTH, Christian. A generalized permutation approach to job shop scheduling with genetic algorithms. En: Operations-Research-Spektrum, Vol 17 (1995) p. 87-92.

- Cualquier gen que tiene este valor en el padre 1(P1) se copiará en la misma posición para el hijo 1(H1) y las posiciones con el número 3 en el padre 2 (P2) se mantendrá fijas en el hijo 2 (H2).
- Las posiciones vacías en H1 se llenan con los genes que son diferentes al dígito del trabajo seleccionado en el orden en que aparecen en el padre 2, así mismo H2 es completado asignando en las posiciones vacías los genes del padre 1 conservando su orden y omitiendo los genes con el mismo dígito del trabajo elegido.

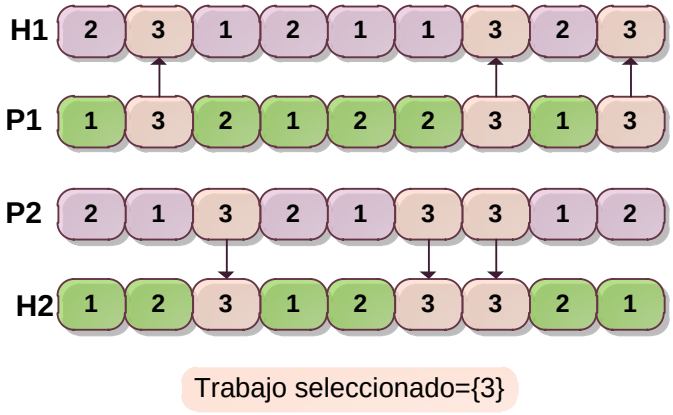
Siguiendo el ejemplo de la Tabla 1, en la Figura 13 se muestra la aplicación del operador de cruce JOX. P1 y P2 son los cromosomas [1 3 2 1 2 2 3 1 3] y [2 1 3 2 1 3 3 1 2]. El trabajo seleccionado es el 3, las posiciones con este dígito se mantendrán. La aplicación del operador genera dos hijos H1 [2 3 1 2 1 1 3 2 3] y H2 [1 2 3 1 2 3 3 2 1], se puede observar que H1 conserva los genes del trabajo 3 en la misma posición que en P1 y los demás genes del vector pertenecen a P2 manteniendo el orden en que aparecen en este, H2 conserva la misma posición de P2 para el trabajo 3 y los demás genes son obtenidos de P1.

Figura 12. Cromosomas de los padres



Antes de continuar el proceso genético, los hijos generados en el cruce son decodificados, el mejor valor encontrado para la función fitness es guardado con el fin de evitar que este se pierda si el individuo es mutado en el siguiente paso.

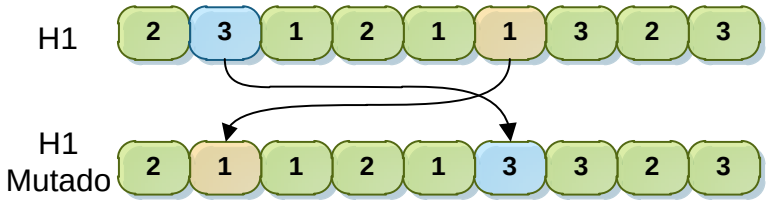
Figura 13. Generación de hijos mediante el operador de cruce JOX



5.1.5. Operador de Mutación

Este operador se usa para diversificar la población mediante la modificación arbitraria de algunos genes usando una distribución de probabilidad, el objetivo de su uso es evitar que la solución converja a un óptimo local. En el operador de cruce se genera una matriz de hijos, cada uno tiene asociado una probabilidad de ser mutado (número aleatorio entre 0 y 1). La probabilidad de mutación es un parámetro del algoritmo que generalmente no supera el 10%, esta probabilidad se compara con la probabilidad que tiene cada hijo de ser mutado, si esta última es menor o igual a la probabilidad de mutación el operador se ejecuta, de lo contrario el individuo se mantiene sin cambios. El procedimiento de mutación intercambia la posición de dos genes escogidos aleatoriamente dentro del cromosoma; en la Figura 14 se observa cómo se realiza la mutación de H1. Se eligen dos números aleatorios entre 1 y $n * m$ (tamaño del cromosoma); para el ejemplo, los números obtenidos son 2 y 6 por tanto estas son las posiciones a intercambiar, en la posición 2 del cromosoma se encuentra la primera operación del trabajo J_3 que será movida a la posición 6, la última operación del trabajo J_1 que está en la posición 6 se mueve a la posición 2, de esta forma un hijo es mutado y el Schedule se modifica.

Figura 14. Operador de mutación aplicado al Hijo 1.



Finalizada la mutación de los individuos, la matriz de los hijos (mutados o no) se convierte en la población inicial para la siguiente generación.

5.1.6. Búsqueda local

La búsqueda local (BL) es un método usado con frecuencia en el campo de la inteligencia artificial debido a que se obtienen buenos resultados en la resolución de problemas de optimización de tipo NP- HARD y la aplicación de su enfoque para ser utilizada con otras heurísticas en algoritmos meméticos para la búsqueda de soluciones óptimas.

La idea principal de la aplicación de algoritmos de BL es realizar una búsqueda en un espacio de soluciones potenciales del problema, que han sido generadas inicialmente y que buscan ser mejoradas a través de modificaciones para tratar de minimizar o maximizar una función objetivo determinada.

Los algoritmos de BL presentan ventajas, dentro de las principales se tiene que debido a su generalidad pueden ser aplicables a diversos problemas de optimización, a diferencia de heurísticas constructivas que requieren de propiedades específicas de acuerdo al problema a solucionar. Como la búsqueda local es realizada en un Schedule actual, estos algoritmos son capaces de solucionar problemas de tamaños más grandes. Asimismo, se considera como desventaja la necesidad de tiempo exponencial para encontrar un mínimo local.

- **Estructura de vecindario**

Un vecindario es una función que define una transición simple a partir de una solución *x* a otra solución mediante la inducción de un cambio que generalmente puede ser visto como una pequeña perturbación (Glover y Laguna, 1997).

Se han propuesto varias estructuras de vecindarios en los métodos de búsqueda local que se usan en la solución del JSP para la minimización del makespan. Uno de los propósitos de estas estrategias es perturbar de manera progresiva el Schedule actual a través de una sucesión de vecinos con el fin de dirigir la búsqueda a una solución mejorada, sin embargo, cada modificación realizada a la solución es parcial y difiere levemente de la solución de la cual es originado el vecino.

La mayoría de las estructuras de vecindario están basadas en el modelo del grafo disyuntivo y toman en cuenta dos conceptos muy importantes: *ruta crítica* y *bloque crítico* de un Schedule dado, se define como ruta crítica a la ruta más larga desde el inicio hasta el final del nodo y se compone de una determinada cantidad de bloques críticos; la longitud de ésta representa el makespan y todas las operaciones pertenecientes a la ruta crítica son denominadas *operaciones críticas;* para dos operaciones críticas adyacentes i y j puede afirmarse que si i es la operación que precede j, entonces el tiempo de inicio de la operación j es exactamente el tiempo de finalización de la operación i, el *bloque crítico* es la máxima secuencia de operaciones que son procesadas en una misma máquina. Una operación crítica no puede ser retrasada sin incrementar el makespan del Schedule. Las estrategias propuestas consisten en la ejecución de movimientos de las operaciones pertenecientes a la ruta crítica generando un Schedule con un nuevo orden de procesamiento, cabe destacar que cualquier intercambio realizado en la programación va a generar una solución factible.

- **Estructura de Vecindario de Nowicki y Smutnicki**

En el enfoque planteado por Nowicki y Smutnicki (1996), se considera una única ruta crítica u, seleccionada aleatoriamente, de las que arroja el grafo disyuntivo $(G(\pi))$ y los bloques pertenecientes a esta ruta crítica $B_1,...,B_r$. Se intercambian las dos primeras (dos últimas), en cada bloque $B_2,\ldots,B_{r-1}$, cada uno de los cuales contiene al menos dos operaciones. En el primer bloque B_1, se intercambian únicamente las dos últimas operaciones y de la misma manera en el último bloque se intercambian las dos primeras. Entonces, se define el conjunto de movimientos de π, como $G(\pi) = U_{j=1}^{r} V_j(\pi)$, donde

$$V_1(\pi) = \begin{cases} \{(u_{b_1-1}\} \; si \; a_1 < \; b_1 \; y \; r > 1 \\ \emptyset \qquad de \; lo \; contrario \end{cases}$$

$$V_j(\pi) = \begin{cases} \left\{\left(u_{a_j}, u_{a_{j+1}}\right), \left(u_{b_{j-1}}, u_{b_j}\right) \right\} \; si \; a_j < \; b_j, \\ \emptyset \qquad de \; lo \; contrario \\ j = 2, \ldots, r-1 \end{cases}$$

$$V_r(\pi) = \begin{cases} \left\{\left(u_{a_r}, u_{a_{r+1}}\right) \right\} \; si \; a_r < \; b_r \; \; y \; r > 1, \\ \emptyset \qquad de \; lo \; contrario \\ \end{cases}$$

El conjunto de movimientos no es vacío slo si el número de bloques es mayor que 1 (r>1) y si existe al menos un bloque con un número de operaciones mayor a 1. El vecindario $H(\pi)$, de π es definida como todas las ordenes de procesamiento obtenidas mediante la aplicación de los movimientos $V(\pi)$.

- **Método de búsqueda local propuesto**

En el algoritmo memético propuesto se aborda la estrategia en el vecindario presentada por Nowicki y Smutnicki[92], también denominado N_5, sin embargo la elección de la ruta crítica del Schedule a la cual será aplicada la búsqueda local no es elegida de manera aleatoria. La idea principal de aplicar esta estructura de vecindario es reducir la cantidad de movimientos mediante la eliminación de algunos de los movimientos propuestos por Van Laarhoven et al. (1992) en los que se conoce desde un principio que no mejorará inmediatamente el makespan. Los intercambios serán realizados únicamente en las fronteras de los bloques en una sola ruta crítica.

Se elige una ruta crítica de una solución inicial (población inicial o proveniente del procedimiento realizado mediante el algoritmo genético y sus respectivos operadores), que es la ruta más larga de todas las conocidas en el Schedule, es decir, con el mayor número de operaciones críticas, si más de una ruta cumple este criterio, es seleccionada la que menor número de bloques contiene. Si se presentan dos o más rutas críticas que cumplen con las mismas condiciones señaladas se elige una de ellas aleatoriamente. Posteriormente, se realiza el intercambio en las dos primeras operaciones del último bloque de la ruta crítica y las dos últimas operaciones del primer bloque, siempre y cuando cada uno de ellos tenga más de dos operaciones críticas, si el primer o último bloque contiene únicamente 2 operaciones, estas son intercambiadas. En caso que el primer bloque B_1 no contenga al menos 2 operaciones, se elige el bloque siguiente (bloque B_2) y así sucesivamente para elegir el primer vecino, si en la elección del segundo vecino, el último bloque B_k no contiene al menos 2 operaciones se elige el bloque B_{k-1} y se realiza el procedimiento de intercambio. Cada intercambio es

[92] NOWICKI, Eugeniusz y SMUTNICKI Czeslaw. Op. cit.

realizado de manera independiente, es decir, se genera un nuevo vecino, una nueva solución (makespan) y un nuevo Schedule.

Empleando el método de búsqueda local descrito, se obtienen 3 (tres) vecinos V_i por cada ruta crítica seleccionada del Schedule generado en la solución inicial, uno por cada intercambio realizado al inicio y final de los bloques correspondientes y el último vecino se consigue realizando los intercambios simultáneamente. Cuando la ruta crítica ha sido seleccionada y contiene un solo bloque, se obtiene un único vecino, debido que solo es posible realizar un intercambio, y en esta situación se realiza el intercambio en las dos últimas operaciones de ese bloque.

Una vez han sido ejecutados los intercambios, se procede a realizar la modificación del orden de programación en el vector de operaciones (cromosomas) para su posterior decodificación en el diagrama de Gantt y evaluar el makespan. Cuando son 3 los vecinos obtenidos por cada Schedule, se selecciona el que arroja el mejor valor de acuerdo a la función fitness y pasa a hacer parte de los posibles padres a los que se les aplicará los operadores genéticos; si únicamente es generado un vecino, este continúa hacia el proceso genético.

El método de BL aplicando la estrategia de vecindario es detallado mediante el siguiente ejemplo:

Tabla 2. Ejemplo de un problema con cuatro trabajos y tres máquinas.

Trabajos	Secuencia de máquinas	Tiempo de Procesamiento
$\boldsymbol{J_1}$	$m_1 – m_2 – m_3$	4 - 3 -2
$\boldsymbol{J_2}$	$m_2 – m_1 – m_3$	1 - 4 – 4
$\boldsymbol{J_3}$	$m_3 – m_2 – m_1$	3 -2 – 3
$\boldsymbol{J_4}$	$m_2 – m_3 – m_1$	3 -3 – 1

En el ejemplo de la Tabla 2, se observa un problema de tamaño $4x3$ es decir, 4 trabajos y 3 máquinas se obtiene el Schedule representado por un vector de números denominado cromosoma para realizar el proceso evolutivo por medio de los operadores del algoritmo genético; [3 4 1 4 2 2 3 1 3 1 2 4] $\longrightarrow$ [O_{31} O_{41} O_{11} O_{42} O_{21} O_{22} O_{32} O_{12} O_{33} O_{13} O_{23} O_{43}], su decodificación arroja el diagrama de Gantt de la Figura 15, con un makespan de 15 unidades de tiempo.

Figura 15. Diagrama de Gantt para el ejemplo de la Tabla 2

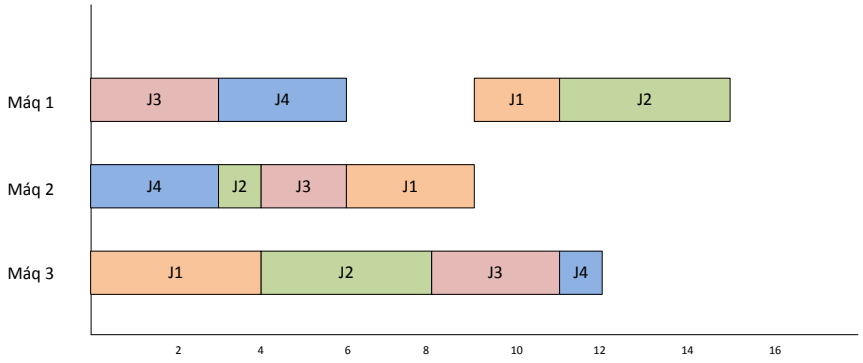
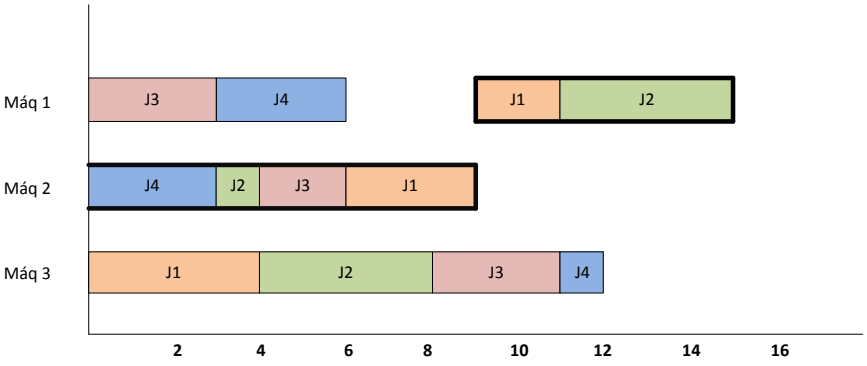
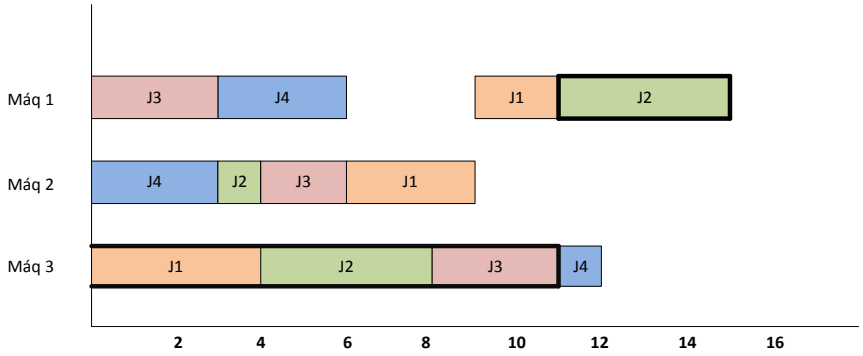


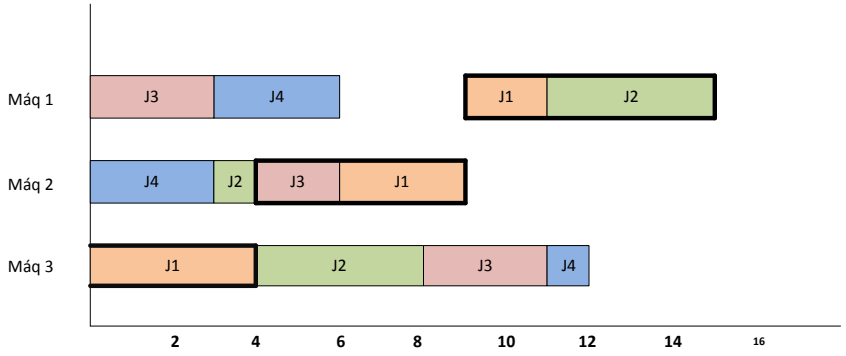
Figura 16. Rutas críticas para el ejemplo de la Tabla 2. (a) primera ruta crítica encontrada, b segunda ruta crítica y c tercera ruta crítica



(a)



(b)



(c)

Las operaciones resaltadas en la Figura 16a, 16b y 16c muestran las tres rutas críticas con sus respectivos bloques obtenidos del Schedule:

R_1: $O_{41} - O_{21} - O_{32} - O_{12} - O_{13} - O_{23}$

$Bloque_1$: $O_{41} - O_{21} - O_{32} - O_{12} \longrightarrow$ Maq 2

$Bloque_2$: $O_{13} - O_{23} \longrightarrow$ Maq 3

R_2: $O_{11} - O_{22} - O_{32} - O_{33} - O_{23}$

$Bloque_1$: $O_{11} - O_{22} - O_{32} - O_{33} \longrightarrow$ Maq 1

$Bloque_2$: $O_{23} \longrightarrow$ Maq 3

R_3: $O_{11} - O_{32} - O_{12} - O_{13} - O_{23}$

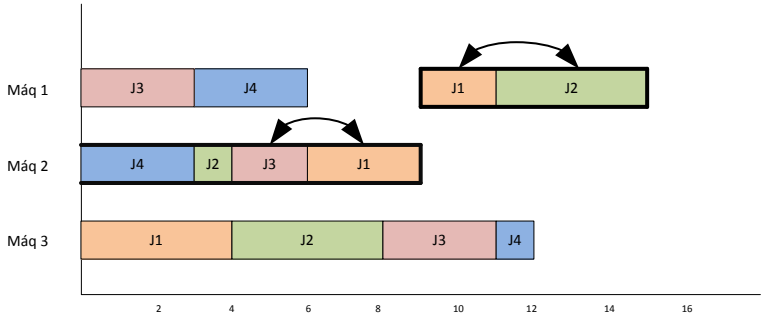
$Bloque_1$: $O_{11} \longrightarrow$ Maq 1

$Bloque_2$: $O_{32} - O_{12} \longrightarrow$ Maq 2

$Bloque_3$: $O_{13} - O_{23} \longrightarrow$ Maq 3

Se elige la ruta con mayor número de operaciones para realizar los intercambios en los bloques correspondientes, para el ejemplo se escoge la ruta R_1, que contiene 6 operaciones. Se intercambian las dos últimas operaciones (O_{32} – O_{12}) pertenecientes al primer bloque de la ruta crítica en la máquina 2 y en el último bloque, debido que contiene únicamente 2 operaciones, se intercambian (O_{13} – O_{23}) como se muestra en la Figura 17.

Figura 17. Intercambio de bloques



Los vecinos obtenidos (Schedules) pueden ser representados directamente realizando los intercambios en el diagrama de Gantt (método de decodificación utilizado en el algoritmo memético) o por medio de la representación genética del Schedule (cromosoma) y posteriormente se realiza la decodificación para conocer el valor de la función fitness (makespan).

Para el ejemplo, los cromosomas son modificados tomando en cuenta las operaciones del trabajo respectivo al cual se aplica el intercambio de acuerdo a la estructura de vecindario, es decir, para el primer bloque, que genera el primer vecino se intercambia la posición en la que se encuentra la operación 2 del trabajo 3 con la operación 2 del trabajo 1, concluyendo de la siguiente manera:

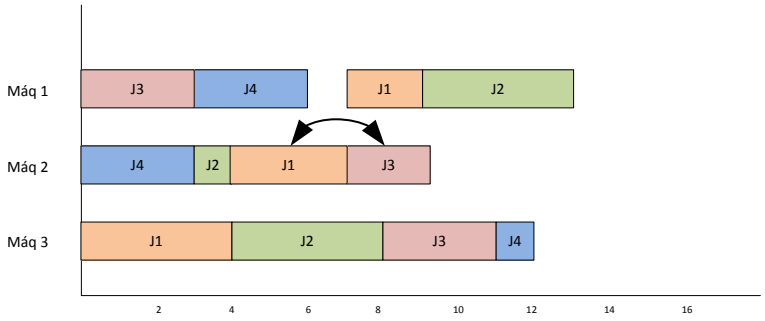
V_1: [3 4 1 4 2 2 **3 1** 3 1 2 4] ⟶ [3 4 1 4 2 2 **1 3** 3 1 2 4]

V_2: [3 4 1 4 2 2 3 1 **3** 1 **2** 4] ⟶ [3 4 1 4 2 2 3 1 **2** 1 **3** 4]

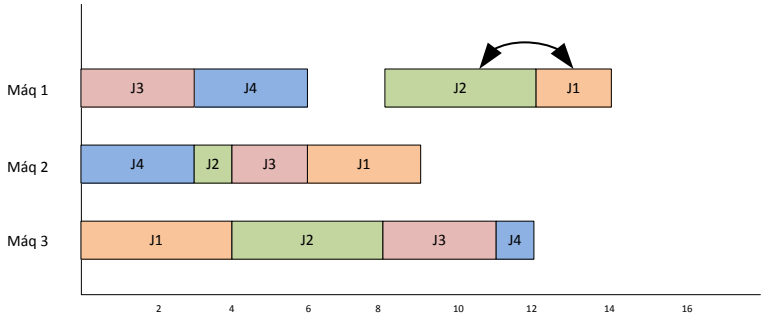
V_3: [3 4 1 4 2 2 **3 1** 3 1 **2** 4] ⟶ [3 4 1 4 2 2 **1 3 2** 1 **3** 4]

La decodificación de los vecinos que se han creado arrojan makespan de 13, 14 y 14 unidades de tiempo para los vecinos V_1, V_2 y V_3 respectivamente como lo muestran las figuras 18a, 18b y 18c, por lo tanto continúa hacia el proceso de selección el vecino V_1. Este proceso es realizado para cada uno de los individuos generados en la población inicial de cada generación.

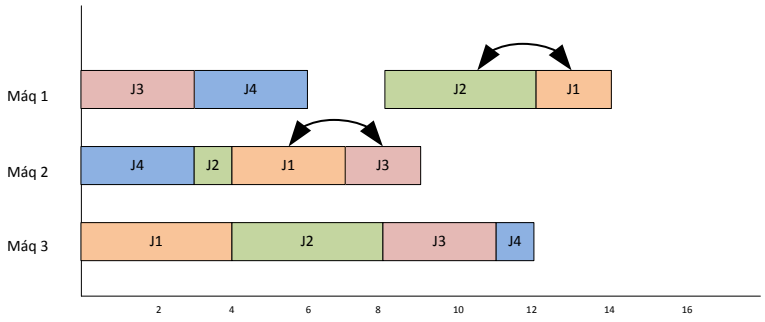
Figura 18. Diagrama de Gantt de los vecinos generados: (a) Diagrama del vecino $\boldsymbol{V_1}$, (b) Diagrama del vecino $\boldsymbol{V_2}$; (c) Diagrama del vecino $\boldsymbol{V_3}$.



(a)



(b)


Máq 1
Máq 2
Máq 3
J3
J4
J2
J1
J4
J2
J1
J3
J1
J2
J3
J4
2
4
6
8
10
12
14
16

6. VALIDACIÓN DEL ALGORITMO

Para evaluar el desempeño del algoritmo memético se utilizaron 9 instancias del benchmarking estas son el Ft6, Ft10 y Ft20 de Fisher y Thompson (1963) y La01, La02, La03, La04, La06 y La21 de Lawrence (1984), se eligieron estas familias de problemas por ser las más estudiadas en la literatura, separadas en dos grupos: baja y media complejidad (Ft6, La01, La02, La03 y La06) y de alta complejidad (Ft10, Ft20 y La21). Se establecieron dos valores para cada parámetro del MA con el fin de observar el comportamiento de las variables.

Los parámetros usados son el tamaño de la población (10 y 20 individuos), el número de generaciones (5 y 20 generaciones), el porcentaje de selección (70% y 90 %) y el porcentaje de mutación (5% y 10%), se ejecutó el programa 10 veces para cada instancia con las posibles combinaciones de los parámetros. Los resultados se pueden consultar en el Anexo E y Anexo F. El lenguaje de programación que se usó para probar la eficiencia de los algoritmos es Matlab® versión R2013a.

Se compara la mejor solución obtenida con el MA propuesto y la mejor solución conocida para cada problema, calculando el porcentaje de diferencia entre las soluciones y el promedio de los datos arrojados por cada combinación utilizada.

Se llevó a cabo una comparación entre el MA y el GA en las mismas instancias del benchmarking; los resultados entre los dos métodos se muestran en el Anexo E y Anexo F, cabe aclarar que se usaron los mismos parámetros y combinaciones para ambos algoritmos.

6.1. RESULTADOS DE LA VALIDACIÓN DE LAS INSTANCIAS

La notación usada en las tablas de resultados de la validación se describe a continuación

P: Tamaño de la población
G: Número de generaciones
%S: Porcentaje de Selección
%M: Porcentaje de Mutación
%Dif: Porcentaje de diferencia
MC: Mejor solución conocida
MO: Mejor Obtenido
M: Media
DS: Desviación estándar

6.1.1. Problemas de baja y media complejidad

Tabla 3. Datos del Problema Ft06, Algoritmo memético

Instancia								
Ft 06 (6X6)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	55	**55**	57,9	1,45	0,00
20	5	0,7	0,05	55	**55**	56,3	1,25	0,00
10	20	0,7	0,05	55	**55**	56,4	1,07	0,00
20	20	0,7	0,05	55	**55**	55,5	1,08	0,00
10	5	0,9	0,05	55	**55**	57,6	2,12	0,00
20	5	0,9	0,05	55	**55**	56,9	1,73	0,00
10	20	0,9	0,05	55	**55**	57,6	1,43	0,00
20	20	0,9	0,05	55	**55**	55,3	0,95	0,00
10	5	0,7	0,1	55	**55**	57,7	1,25	0,00
20	5	0,7	0,1	55	**55**	57,1	1,37	0,00
10	20	0,7	0,1	55	**55**	56,9	1,20	0,00
20	20	0,7	0,1	55	**55**	55,4	0,84	0,00
10	5	0,9	0,1	55	**55**	57,1	2,02	0,00
20	5	0,9	0,1	55	**55**	56,3	1,25	0,00
10	20	0,9	0,1	55	**55**	55,7	1,16	0,00
20	20	0,9	0,1	55	**55**	55,2	0,63	0,00

Tabla 4. Datos del Problema Ft06, Algoritmo genético

Instancia								
Ft06 (6x6)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	55	58	59,9	0,994	0,0545
20	5	0,7	0,05	55	55	58,2	1,549	0,0000
10	20	0,7	0,05	55	57	58,5	1,080	0,0364
20	20	0,7	0,05	55	55	58,1	1,792	0,0000
10	5	0,9	0,05	55	55	58,0	2,404	0,0000
20	5	0,9	0,05	55	55	58,2	1,619	0,0000
10	20	0,9	0,05	55	55	58,3	2,111	0,0000
20	20	0,9	0,05	55	55	57,9	1,449	0,0000
10	5	0,7	0,1	55	55	58,9	1,792	0,0000
20	5	0,7	0,1	55	57	59,0	1,414	0,0364
10	20	0,7	0,1	55	55	58,1	1,912	0,0000
20	20	0,7	0,1	55	55	56,9	1,792	0,0000
10	5	0,9	0,1	55	58	60,5	1,434	0,0545
20	5	0,9	0,1	55	57	58,9	1,101	0,0364
10	20	0,9	0,1	55	58	58,9	0,876	0,0545
20	20	0,9	0,1	55	55	57,3	1,889	0,0000

Comparando los resultados del GA y MA para la instancia Ft06, se observa que el algoritmo memético siempre encuentra el mejor valor conocido, independientemente de los valores que toman los parámetros, sin embargo, el algoritmo genético aunque arroja buenos resultados para algunas combinaciones de los parámetros, presenta falencia cuando es aplicado los valores bajos del tamaño de población y generaciones al mismo tiempo.

Tabla 5. Datos del Problema La01, Algoritmo memético

Instancia								
La 01 (10X5)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	666	689	714,1	17,18	0,03
20	5	0,7	0,05	666	675	694,3	14,47	0,01

Instancia								
La 01 (10X5)								
10	20	0,7	0,05	666	**666**	682,2	14,00	0,00
20	20	0,7	0,05	666	**666**	672,1	8,86	0,00
10	5	0,9	0,05	666	679	697,1	15,41	0,02
20	5	0,9	0,05	666	**666**	690,9	21,02	0,00
10	20	0,9	0,05	666	**666**	683,4	13,93	0,00
20	20	0,9	0,05	666	**666**	668,3	4,06	0,00
10	5	0,7	0,1	666	677	701	18,62	0,02
20	5	0,7	0,1	666	670	695,8	18,84	0,01
10	20	0,7	0,1	666	**666**	689,4	18,09	0,00
20	20	0,7	0,1	666	**666**	670,6	4,90	0,00
10	5	0,9	0,1	666	681	711,7	17,35	0,02
20	5	0,9	0,1	666	669	691,5	11,18	0,00
10	20	0,9	0,1	666	**666**	677,4	10,16	0,00
20	20	0,9	0,1	666	**666**	669,2	7,315	0,00

Tabla 6. Datos del Problema La01, Algoritmo genético

Instancia								
La 01 (10X5)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	666	712	729,6	14,539	0,0691
20	5	0,7	0,05	666	705	722,8	11,942	0,0586
10	20	0,7	0,05	666	700	720,4	12,791	0,0511
20	20	0,7	0,05	666	675	701,6	15,335	0,0135
10	5	0,9	0,05	666	692	739,5	27,077	0,0390
20	5	0,9	0,05	666	675	705,0	19,737	0,0135
10	20	0,9	0,05	666	**669**	712,3	31,910	0,0045
20	20	0,9	0,05	666	684	705,2	19,205	0,0270
10	5	0,7	0,1	666	703	722,3	13,442	0,0556
20	5	0,7	0,1	666	679	709,7	23,089	0,0195
10	20	0,7	0,1	666	674	712,4	20,392	0,0120
20	20	0,7	0,1	666	672	698,1	15,029	0,0090
10	5	0,9	0,1	666	695	729,1	22,263	0,0435
20	5	0,9	0,1	666	693	717,6	16,399	0,0405
10	20	0,9	0,1	666	686	714,8	17,806	0,0300
20	20	0,9	0,1	666	686	701,4	12,213	0,0300

Para el problema La 01, el algoritmo memético encuentra los valores del makespan MC al utilizar al menos uno de los parámetros (tamaño de población o generaciones) en su valor máximo, principalmente cuando se ejecuta con 20 generaciones. Por su parte el mejor valor obtenido con el algoritmo genético corresponde al utilizar el parámetro de generación en su máximo valor, sin lograr el mejor valor conocido de la instancia.

Tabla 7. Datos del Problema La02, Algoritmo memético

Instancia								
La02 (10X5)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	655	714	743,6	19,323	0,0901
20	5	0,7	0,05	655	695	733,7	20,336	0,0611
10	20	0,7	0,05	655	689	724,2	18,390	0,0519
20	20	0,7	0,05	655	**667**	699,4	22,051	0,0183
10	5	0,9	0,05	655	714	746,0	21,685	0,0901
20	5	0,9	0,05	655	722	735,8	7,997	0,1023
10	20	0,9	0,05	655	686	720,6	21,093	0,0473
20	20	0,9	0,05	655	681	696,8	12,136	0,0397
10	5	0,7	0,1	655	726	744,6	12,167	0,1084
20	5	0,7	0,1	655	726	744,6	12,167	0,1084
10	20	0,7	0,1	655	692	718,3	16,591	0,0565
20	20	0,7	0,1	655	677	708,2	13,726	0,0336
10	5	0,9	0,1	655	711	736,6	22,843	0,0855
20	5	0,9	0,1	655	696	729,1	16,052	0,0626
10	20	0,9	0,1	655	675	707,5	22,372	0,0305
20	20	0,9	0,1	655	677	694,9	14,587	0,0336

Tabla 8. Datos del Problema La02, Algoritmo genético

Instancia								
La02 (10x5)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	655	744	773,8	23,902	0,1359
20	5	0,7	0,05	655	730	761,6	18,957	0,1145
10	20	0,7	0,05	655	713	775,8	30,272	0,0885
20	20	0,7	0,05	655	707	747,6	22,500	0,0794
10	5	0,9	0,05	655	758	781,7	16,614	0,1573
20	5	0,9	0,05	655	730	757,3	14,576	0,1145
10	20	0,9	0,05	655	733	769,0	22,637	0,1191
20	20	0,9	0,05	655	750	758,2	10,748	0,1450
10	5	0,7	0,1	655	757	787,4	19,033	0,1557
20	5	0,7	0,1	655	731	761,9	21,789	0,1160
10	20	0,7	0,1	655	737	764,0	18,968	0,1252
20	20	0,7	0,1	655	721	738,8	15,950	0,1008
10	5	0,9	0,1	655	744	770,3	18,391	0,1359
20	5	0,9	0,1	655	734	761,8	17,132	0,1206
10	20	0,9	0,1	655	753	771,2	14,868	0,1496
20	20	0,9	0,1	655	**688**	735,0	25,910	0,0504

Para la instancia La02 ninguno de los dos algoritmos logran hallar el mínimo valor conocido, no obstante los mejores valores obtenidos en la ejecución del GA y MA se da con los parámetros más altos en el tamaño de población y el número de generaciones. La diferencia entre el MC y el MO para el algoritmo memético es menos del 2%.

Tabla 9. Datos del Problema La03, Algoritmo memético

Instancia								
La03(10x5)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	597	638	665,3	16,118	0,0687
20	5	0,7	0,05	597	635	661,7	11,441	0,0637
10	20	0,7	0,05	597	639	659,8	13,685	0,0704
20	20	0,7	0,05	597	628	641,6	10,426	0,0519

Instancia								
La03(10x5)								
10	5	0,9	0,05	597	651	672,4	13,632	0,0905
20	5	0,9	0,05	597	652	660,8	9,897	0,0921
10	20	0,9	0,05	597	628	649,9	13,119	0,0519
20	20	0,9	0,05	597	618	633,1	16,107	0,0352
10	5	0,7	0,1	597	651	670,3	14,236	0,0905
20	5	0,7	0,1	597	649	666,5	10,617	0,0871
10	20	0,7	0,1	597	632	654,8	13,943	0,0586
20	20	0,7	0,1	597	621	637,5	14,324	0,0402
10	5	0,9	0,1	597	641	661,7	13,039	0,0737
20	5	0,9	0,1	597	649	658,5	9,443	0,0871
10	20	0,9	0,1	597	636	658,5	13,285	0,0653
20	20	0,9	0,1	597	**613**	638,8	17,028	0,0268

Tabla 10. Datos del Problema La03, Algoritmo genético

Instancia								
La03(10x5)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	597	679	691,8	9,705	0,1374
20	5	0,7	0,05	597	664	684,0	11,879	0,1122
10	20	0,7	0,05	597	658	672,9	11,040	0,1022
20	20	0,7	0,05	597	654	671,0	14,712	0,0955
10	5	0,9	0,05	597	688	705,5	11,424	0,1524
20	5	0,9	0,05	597	661	679,4	11,578	0,1072
10	20	0,9	0,05	597	672	688,1	14,004	0,1256
20	20	0,9	0,05	597	**628**	658,3	18,349	0,0519
10	5	0,7	0,1	597	669	699,4	16,933	0,1206
20	5	0,7	0,1	597	655	683,9	13,908	0,0972
10	20	0,7	0,1	597	664	683,6	14,623	0,1122
20	20	0,7	0,1	597	634	664,3	17,727	0,0620
10	5	0,9	0,1	597	665	696,5	17,822	0,1139
20	5	0,9	0,1	597	619	672,5	24,419	0,0369
10	20	0,9	0,1	597	650	686,7	21,980	0,0888
20	20	0,9	0,1	597	638	669,8	17,856	0,0687

Para el problema La03 ocurre una situación similar a la instancia La02, debido que los dos algoritmos obtienen su mejor valor con los parámetros máximos del Tamaño de población y el número de generaciones. Pese a que no se encontró el MC, la diferencia con el mejor obtenido en el MA no supera el 3%.

Tabla 11. Datos del Problema La04, Algoritmo memético

Instancia								
La 04 (10X5)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	590	641	651,5	8,772	0,0864
20	5	0,7	0,05	590	615	636,7	15,471	0,0424
10	20	0,7	0,05	590	607	635,3	24,097	0,0288
20	20	0,7	0,05	590	610	620,1	8,582	0,0339
10	5	0,9	0,05	590	632	653,3	11,963	0,0712
20	5	0,9	0,05	590	625	646,3	10,688	0,0593
10	20	0,9	0,05	590	611	623,6	10,987	0,0356
20	20	0,9	0,05	590	600	608,5	6,835	0,0169
10	5	0,7	0,1	590	620	660,0	24,143	0,0508
20	5	0,7	0,1	590	623	644,2	16,054	0,0559
10	20	0,7	0,1	590	611	624,8	9,739	0,0356
20	20	0,7	0,1	590	607	622,8	9,438	0,0288
10	5	0,9	0,1	590	641	653,2	8,377	0,0864
20	5	0,9	0,1	590	621	639,6	16,748	0,0525
10	20	0,9	0,1	590	**598**	624,8	13,155	0,0136
20	20	0,9	0,1	590	600	613,5	7,322	0,0169

Tabla 12. Datos del Problema La04, Algoritmo genético

Instancia								
La 04 (10X5)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	590	661	683,8	7,326	0,1203
20	5	0,7	0,05	590	651	671,6	16,521	0,1034
10	20	0,7	0,05	590	639	664,6	28,151	0,0831
20	20	0,7	0,05	590	620	647,5	7,182	0,0508
10	5	0,9	0,05	590	631	689,5	25,189	0,0695

Instancia								
La 04 (10X5)								
20	5	0,9	0,05	590	646	677,8	26,968	0,0949
10	20	0,9	0,05	590	659	680,4	22,051	0,1169
20	20	0,9	0,05	590	623	651,8	12,550	0,0559
10	5	0,7	0,1	590	636	679,8	32,866	0,0780
20	5	0,7	0,1	590	642	668,1	8,302	0,0881
10	20	0,7	0,1	590	614	666,9	29,871	0,0407
20	20	0,7	0,1	590	**611**	643,5	24,103	0,0356
10	5	0,9	0,1	590	645	688,4	26,090	0,0932
20	5	0,9	0,1	590	635	668,2	18,534	0,0763
10	20	0,9	0,1	590	650	682,2	21,374	0,1017
20	20	0,9	0,1	590	621	644,7	14,855	0,0525

El algoritmo memético arroja mejores soluciones en todas las combinaciones de los parámetros respecto a los valores obtenidos por el algoritmo genético, ya que el MO en el MA supera tan solo en 1% de diferencia al makespan mejor conocido.

Tabla 13. Datos del Problema La06, Algoritmo memético

Instancia								
La 06 (15X5)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	926	**926**	926,0	0,00	0,00
20	5	0,7	0,05	926	**926**	926,9	2,85	0,00
10	20	0,7	0,05	926	**926**	926,0	0,00	0,00
20	20	0,7	0,05	926	**926**	926,0	0,00	0,00
10	5	0,9	0,05	926	**926**	926,0	0,00	0,00
20	5	0,9	0,05	926	**926**	926,0	0,00	0,00
10	20	0,9	0,05	926	**926**	926,0	0,00	0,00
20	20	0,9	0,05	926	**926**	926,0	0,00	0,00
10	5	0,7	0,1	926	**926**	926,8	1,69	0,00
20	5	0,7	0,1	926	**926**	926,8	1,69	0,00
10	20	0,7	0,1	926	**926**	926,0	0,00	0,00
20	20	0,7	0,1	926	**926**	926,0	0,00	0,00
10	5	0,9	0,1	926	**926**	926,0	0,00	0,00
20	5	0,9	0,1	926	**926**	926,0	0,00	0,00

Instancia								
La 06 (15X5)								
10	20	0,9	0,1	926	**926**	926,0	0,00	0,00
20	20	0,9	0,1	926	**926**	926,0	0,00	0,00

Tabla 14. Datos del Problema La06, Algoritmo genético

Instancia								
La06 (15x10)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	926	**926**	947,0	20,997	0,0000
20	5	0,7	0,05	926	**926**	929,1	6,045	0,0000
10	20	0,7	0,05	926	**926**	931,8	9,053	0,0000
20	20	0,7	0,05	926	**926**	926,0	0,000	0,0000
10	5	0,9	0,05	926	**926**	938,5	9,618	0,0000
20	5	0,9	0,05	926	**926**	927,4	3,098	0,0000
10	20	0,9	0,05	926	**926**	932,8	8,791	0,0000
20	20	0,9	0,05	926	**926**	926,0	0,000	0,0000
10	5	0,7	0,1	926	**926**	936,8	8,548	0,0000
20	5	0,7	0,1	926	**926**	934,6	10,135	0,0000
10	20	0,7	0,1	926	**926**	929,9	6,757	0,0000
20	20	0,7	0,1	926	**926**	928,4	3,718	0,0000
10	5	0,9	0,1	926	**926**	936,1	10,535	0,0000
20	5	0,9	0,1	926	**926**	927,3	2,983	0,0000
10	20	0,9	0,1	926	**926**	932,6	8,086	0,0000
20	20	0,9	0,1	926	**926**	926,5	1,581	0,0000

Cuando se ejecuta el algoritmo memético y el algoritmo genético para el problema La06 se obtiene con cualquier combinación de parámetros el mejor makespan, sin embargo, los datos obtenidos mediante la ejecución del algoritmo genético en cada combinación de parámetros tienen mayor dispersión, (basados en la desviación estándar del makespan) que las soluciones arrojadas por el MA.

6.1.2. Problemas de alta complejidad

Tabla 15. Datos del Problema Ft10, Algoritmo memético

Instancia								
Ft10 (10x10)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0.7	0.05	930	1095	1124,3	16,627	0,1774
20	5	0.7	0.05	930	1064	1098,6	21,803	0,1441
10	20	0.7	0.05	930	1039	1081,2	25,006	0,1172
20	20	0.7	0.05	930	1018	1040,2	13,831	0,0946
10	5	0.9	0.05	930	1078	1111,3	23,276	0,1591
20	5	0.9	0.05	930	1072	1091,5	18,368	0,1527
10	20	0.9	0.05	930	998	1073,9	41,744	0,0731
20	20	0.9	0.05	930	1011	1037,8	14,398	0,0871
10	5	0.7	0.1	930	1071	1109,8	25,341	0,1516
20	5	0.7	0.1	930	1085	1097,4	13,850	0,1667
10	20	0.7	0.1	930	1045	1070,8	17,015	0,1237
20	20	0.7	0.1	930	1021	1052,3	20,407	0,0978
10	5	0.9	0.1	930	1041	1096,1	35,476	0,1194
20	5	0.9	0.1	930	1067	1096,3	13,217	0,1473
10	20	0.9	0.1	930	1028	1063,1	24,759	0,1054
20	20	0.9	0.1	930	**982**	1029,5	26,542	0,0559

Tabla 16. Datos del Problema Ft10, Algoritmo genético

Instancia								
Ft 10 (10X10)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	930	1089	1134,8	27,628	0,1710
20	5	0,7	0,05	930	1085	1109,9	18,847	0,1667
10	20	0,7	0,05	930	1076	1123,5	22,182	0,1570
20	20	0,7	0,05	930	1040	1079,6	25,735	0,1183
10	5	0,9	0,05	930	1120	1156,8	22,895	0,2043
20	5	0,9	0,05	930	1084	1116,1	15,438	0,1656
10	20	0,9	0,05	930	1063	1129,1	39,295	0,1430
20	20	0,9	0,05	930	1035	1096,9	33,594	0,1129
10	5	0,7	0,1	930	1094	1136,5	30,511	0,1763
20	5	0,7	0,1	930	1050	1121,7	29,044	0,1290

Instancia								
Ft 10 (10X10)								
10	20	0,7	0,1	930	1064	1103,2	23,385	0,1441
20	20	0,7	0,1	930	**1026**	1068,3	24,486	0,1032
10	5	0,9	0,1	930	1074	1154,6	33,906	0,1548
20	5	0,9	0,1	930	1052	1103,7	31,760	0,1312
10	20	0,9	0,1	930	1083	1113,8	25,494	0,1645
20	20	0,9	0,1	930	1060	1091,8	22,419	0,1398

El algoritmo memético tiene un mejor desempeño en los datos de makespan obtenidos cuando se ejecuta con los parametros más altos del tamaño de población y generaciones, de la misma manera que el algoritmo genético, aunque el valor más cercano al MC lo arrojó el MA propuesto con una diferencia aproximadamente del 6% respecto a este valor.

Tabla 17. Datos del Problema Ft20, Algoritmo memético

Instancia								
Ft20 (20x5)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	1165	1389	1426,2	24,031	0,1923
20	5	0,7	0,05	1165	1364	1420,0	36,733	0,1708
10	20	0,7	0,05	1165	1361	1395,8	26,174	0,1682
20	20	0,7	0,05	1165	1307	1360,9	25,666	0,1219
10	5	0,9	0,05	1165	1367	1429,0	36,881	0,1734
20	5	0,9	0,05	1165	1355	1396,9	27,380	0,1631
10	20	0,9	0,05	1165	1343	1389,6	27,953	0,1528
20	20	0,9	0,05	1165	1325	1359,4	21,490	0,1373
10	5	0,7	0,1	1165	1392	1420,5	24,496	0,1948
20	5	0,7	0,1	1165	1391	1425,8	30,184	0,1940
10	20	0,7	0,1	1165	1375	1413,1	30,249	0,1803
20	20	0,7	0,1	1165	1335	1363,7	19,545	0,1459
10	5	0,9	0,1	1165	1353	1419,8	36,560	0,1614
20	5	0,9	0,1	1165	1381	1407,2	19,066	0,1854
10	20	0,9	0,1	1165	1369	1402,8	25,516	0,1751
20	20	0,9	0,1	1165	**1313**	1357,6	35,312	0,1270

Tabla 18. Datos del Problema Ft20, Algoritmo genético

Instancia								
Ft 20 (20X5)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	1165	1394	1459,3	31,298	0,1966
20	5	0,7	0,05	1165	1386	1428,5	28,465	0,1897
10	20	0,7	0,05	1165	1369	1432,7	42,098	0,1751
20	20	0,7	0,05	1165	**1373**	1396,8	15,569	0,1785
10	5	0,9	0,05	1165	1387	1450,7	34,680	0,1906
20	5	0,9	0,05	1165	1400	1444,5	23,904	0,2017
10	20	0,9	0,05	1165	1387	1439,8	29,785	0,1906
20	20	0,9	0,05	1165	1379	1420,4	33,874	0,1837
10	5	0,7	0,1	1165	1404	1468,0	39,169	0,2052
20	5	0,7	0,1	1165	1393	1434,4	30,736	0,1957
10	20	0,7	0,1	1165	1353	1418,2	30,749	0,1614
20	20	0,7	0,1	1165	1366	1398,6	28,918	0,1725
10	5	0,9	0,1	1165	1393	1444,1	26,752	0,1957
20	5	0,9	0,1	1165	1381	1435,6	36,827	0,1854
10	20	0,9	0,1	1165	1368	1454,9	40,862	0,1742
20	20	0,9	0,1	1165	1371	1412,4	22,420	0,1768

El mejor resultado obtenido para el problema Ft 20 es de 1313 unidades de tiempo al ser ejecutado con el algoritmo memético y presentando una diferencia del 13% aproximadamente respecto al mejor valor conocido para esta instancia. De la misma manera como ocurre en problemas anteriores, la mejor solución y los mejores valores son arrojados por el algoritmo cuando se ajustan los parámetros máximos en el Tamaño de población y número de generaciones.

Tabla 19. Datos del Problema La21, Algoritmo memético

Instancia								
La21 (15x10)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	1046	1217	1255,4	25,483	0,1635
20	5	0,7	0,05	1046	1226	1249,8	22,444	0,1721

Instancia								
La21 (15x10)								
10	20	0,7	0,05	1046	**1166**	1231,7	24,833	0,1147
20	20	0,7	0,05	1046	1178	1210,2	16,006	0,1262
10	5	0,9	0,05	1046	1234	1263,9	23,407	0,1797
20	5	0,9	0,05	1046	1204	1244,3	19,334	0,1511
10	20	0,9	0,05	1046	1186	1239,8	30,058	0,1338
20	20	0,9	0,05	1046	1183	1217,0	20,966	0,1310
10	5	0,7	0,1	1046	1230	1269,7	25,232	0,1759
20	5	0,7	0,1	1046	1183	1250,2	30,451	0,1310
10	20	0,7	0,1	1046	1189	1220,3	22,598	0,1367
20	20	0,7	0,1	1046	1195	1216,0	17,282	0,1424
10	5	0,9	0,1	1046	1212	1249,5	23,472	0,1587
20	5	0,9	0,1	1046	1186	1243,6	23,992	0,1338
10	20	0,9	0,1	1046	1199	1226,9	17,311	0,1463
20	20	0,9	0,1	1046	1167	1198,1	19,203	0,1157

Tabla 20. Datos del Problema Ft20, Algoritmo genético

Instancia								
La21 (15x10)								
P	**G**	**%S**	**%M**	**MC**	**MO**	**M**	**DS**	**%Dif**
10	5	0,7	0,05	1046	1250	1290,2	25,068	0,1950
20	5	0,7	0,05	1046	1247	1277,1	18,297	0,1922
10	20	0,7	0,05	1046	1240	1275,0	19,715	0,1855
20	20	0,7	0,05	1046	1225	1251,8	15,697	0,1711
10	5	0,9	0,05	1046	1268	1301,3	20,532	0,2122
20	5	0,9	0,05	1046	1192	1252,3	30,696	0,1396
10	20	0,9	0,05	1046	1227	1265,2	30,025	0,1730
20	20	0,9	0,05	1046	1196	1246,7	35,783	0,1434
10	5	0,7	0,1	1046	1218	1279,9	34,213	0,1644
20	5	0,7	0,1	1046	1246	1279,3	27,685	0,1912
10	20	0,7	0,1	1046	**1120**	1262,7	56,647	0,0707
20	20	0,7	0,1	1046	1194	1242,1	30,311	0,1415
10	5	0,9	0,1	1046	1238	1295,0	37,298	0,1836
20	5	0,9	0,1	1046	1257	1279,5	21,485	0,2017
10	20	0,9	0,1	1046	1236	1291,6	30,008	0,1816
20	20	0,9	0,1	1046	1190	1250,8	32,231	0,1377

De acuerdo con los resultados presentados, con ninguna de las combinaciones de parámetros establecidas se encontró el valor mejor conocido en los problemas anteriores denominados de alta complejidad, se observa que en los 3 casos la mejor solución se obtiene con los parámetros altos en el tamaño de la población y el número de generaciones.

7. DISEÑO EXPERIMENTAL PARA LA DETERMINACIÓN DE LOS FACTORES INCIDENTES EN EL PROBLEMA DE JSP

Para el análisis de los factores que influyen en la respuesta de la función objetivo (makespan), se eligió un diseño factorial fraccionado 2^{k-1}, es decir, una fracción a la mitad del diseño factorial completo, considerando 4 factores *k* y dos niveles para cada uno de ellos.

7.1. INSTANCIAS DEL BENCHMARKING ANALIZADAS

Observando los resultados arrojados mediante la validación del algoritmo con instancias de baja, media y alta complejidad, y tomando en cuenta el comportamiento de la respuesta obtenida de acuerdo a los parámetros establecidos en el programa, se decidió realizar dos diseños independientes para las instancias considerando su nivel de complejidad. Los diseños de experimentos estuvieron conformados por problemas del benchmark de Fisher y Thompson, mejor conocidas como “Ft” y de Lawrence conocidas como “La”, tomados de la OR-Library[93]. Para las instancias del primer diseño se consideraron los problemas Ft06, La01, La06, La09, La11, La14 y La17, que de acuerdo a la literatura son de baja y media complejidad, mientras que los problemas de alta complejidad para el segundo diseño de experimentos son el Ft10, Ft20, La16, La21, La 24, La25 y La 38, la información de las instancias usadas se encuentran en el Anexo G. Los experimentos numéricos fueron desarrollados en el lenguaje de programación Matlab® en un PC con procesador Intel core i7, 3.40GHz y 8GB de memoria. El diseño experimental fue analizado en Statgraphics Centurion. Los datos utilizados para ello se encuentran en el Anexo H.

[93] BEASLEY, John. OR-Library. Job Shop Scheduling. [Libreria de problemas en linea]. [consultado 1 de Agosto de 2013]. Disponible en: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt

7.2. FACTORES PRINCIPALES

Los factores a ser analizados son aquellos que presentan mayor influencia en la calidad de la respuesta y los niveles serán establecidos según la complejidad, para ello fueron identificados cuatro (4) parámetros: Tamaño de población, el número de generaciones, porcentaje de selección y porcentaje de mutación. El número de réplicas que se realizó para cada combinación o tratamiento fue de 8 para las instancias de baja y media complejidad y 5 veces para las de alta complejidad.

Tabla 21. Niveles de los factores del diseño de experimentos

Factores	Baja y Media complejidad		Alta complejidad	
	Bajo	Alto	Bajo	Alto
Tamaño de la población	10	30	80	150
Generaciones	20	30	100	170
% Selección	0,9	0,7	0,9	0,8
%Mutación	0,1	0,05	0,1	0,05

A continuación se lista el diseño factorial completo $2^{4-1}= 2^3$, que muestra los 8 primeros tratamientos de los 16 posibles del diseño factorial completo 2^4.

Cuadro 6. Tratamientos del diseño factorial 2^{4-1}

	A TamPob	B Generaciones	C %Selección	D %Mutación
1	-	-	-	-
2	+	-	-	+
3	-	+	-	+
4	+	+	-	-
5	-	-	+	+
6	+	-	+	-
7	-	+	+	-
8	+	+	+	+

7.3. ANÁLISIS DE LOS EFECTOS PRINCIPALES DE LOS PARÁMETROS DEL ALGORITMO.

7.3.1. Análisis de varianza (ANOVA) para el problema Ft06

Figura 19. Diagrama de Pareto de efectos estimados para Makespan del problema Ft06

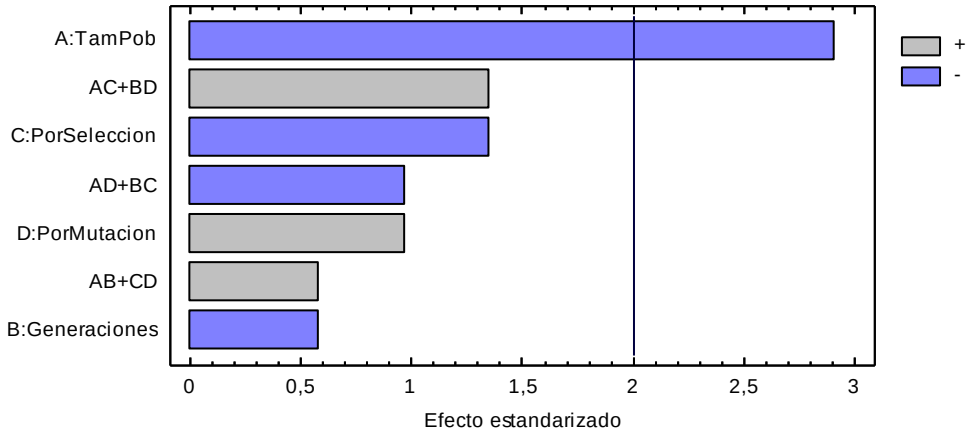


Tabla 22. Análisis de Varianza para el problema Ft06

Fuente	Suma de Cuadrados	Grados de Libertad	Cuadrado Medio	Razón-F	Valor-P (Significancia)
A:TamPob	3,51563	1	3,51563	8,42	0,0053
B:Generaciones	0,140625	1	0,140625	0,34	0,5640
C:PorSeleccion	0,765625	1	0,765625	1,83	0,1811
D:PorMutacion	0,390625	1	0,390625	0,94	0,3375
AB+CD	0,140625	1	0,140625	0,34	0,5640
AC+BD	0,765625	1	0,765625	1,83	0,1811
AD+BC	0,390625	1	0,390625	0,94	0,3375
Error total	23,375	56	0,417411		
Total (corregido)	29,4844	63			

Se observa en la Tabla 22 que únicamente el efecto generado por el tamaño de la población (A) es significativo en la variable Makespan al ser menor que 0.05,

cuando el tamaño de la población usa su nivel alto el makespan converge a la mejor solución encontrada al problema. El factor que menos incide en la respuesta es el número de generaciones, los demás efectos principales y las interacciones de estos no inciden en la variable de salida.

7.3.2. Análisis de Varianza (ANOVA) para el problema La01

Figura 20. Diagrama de Pareto de efectos estimados para Makespan del problema La01

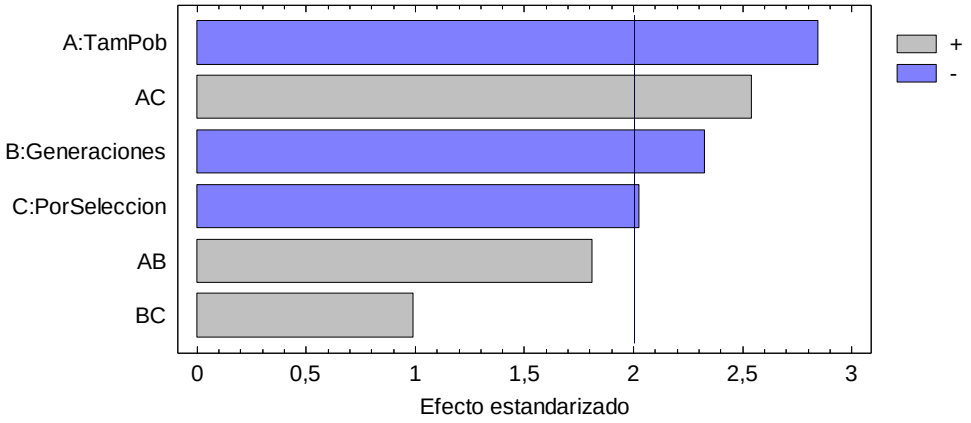


Tabla 23. Análisis de Varianza para el problema La16

Fuente	Suma de Cuadrados	Grados de Libertad	Cuadrado Medio	Razón-F	Valor-P (Significancia)
A:TamPob	272,25	1	272,25	8,29	0,0056
B:Generaciones	182,25	1	182,25	5,55	0,0220
C:PorSeleccion	138,063	1	138,063	4,20	0,0451
D:PorMutacion	76,5625	1	76,5625	2,33	0,1325
AB+CD	110,25	1	110,25	3,36	0,0723
AC+BD	217,563	1	217,563	6,62	0,0127
AD+BC	33,0625	1	33,0625	1,01	0,3201
Error total	1839,75	56	32,8527		
Total(corregido)	2869,75	63			

De la Tabla 23 se puede concluir que son 4 los factores cuyos efectos son significativos en la respuesta al tener un valor-P menor que 0.05, el tamaño de la población (A), el número de generaciones (B) , el porcentaje de selección (C) y los efectos confundidos AC+BD tienen incidencia en la variable de salida; como el factor D (Porcentaje de Mutación) es el menos significativo se espera que tampoco contribuya en las interacciones y por tanto el efecto AC+BD corresponderá únicamente a la interacción entre el Tamaño de la población y el Porcentaje de Selección. La combinación de los factores significativos que minimiza el makespan es A^+, B^+ y C^+ (TamPob, Generaciones y PorSeleccion en su nivel alto), en cuanto a la interacción AC la mejor combinación es A^+C^- (A en alto y C en bajo). Las demás interacciones entre los factores analizados no son significativas.

7.3.3. Análisis de varianza (ANOVA) para el problema La09

Figura 21. Diagrama de Pareto de efectos estimados para Makespan del problema La09

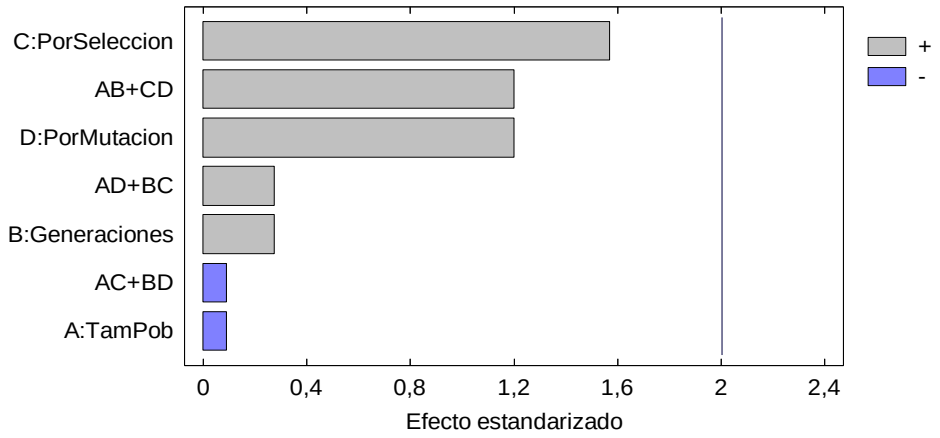


Tabla 24. Análisis de Varianza para el problema La09

Fuente	Suma de Cuadrados	Grados de Libertad	Cuadrado Medio	Razón-F	Valor-P
A:TamPob	0,015625	1	0,015625	0,01	0,9267
B:Generaciones	0,140625	1	0,140625	0,08	0,7825
C:PorSeleccion	4,51563	1	4,51563	2,47	0,1217
D:PorMutacion	2,64063	1	2,64063	1,44	0,2345
AB+CD	2,64063	1	2,64063	1,44	0,2345
AC+BD	0,015625	1	0,015625	0,01	0,9267
AD+BC	0,140625	1	0,140625	0,08	0,7825
Error total	102,375	56	1,82813		
Total (corregido.)	112,484	63			

De la Tabla 24 se puede concluir que tanto las interacciones como los efectos principales no inciden en la variable de respuesta ya que el valor de significancia para todos está por encima de 0.05.

7.3.4. Análisis de varianza (ANOVA) para el problema La17

Figura 22. Diagrama de Pareto de efectos estimados para Makespan del problema La17

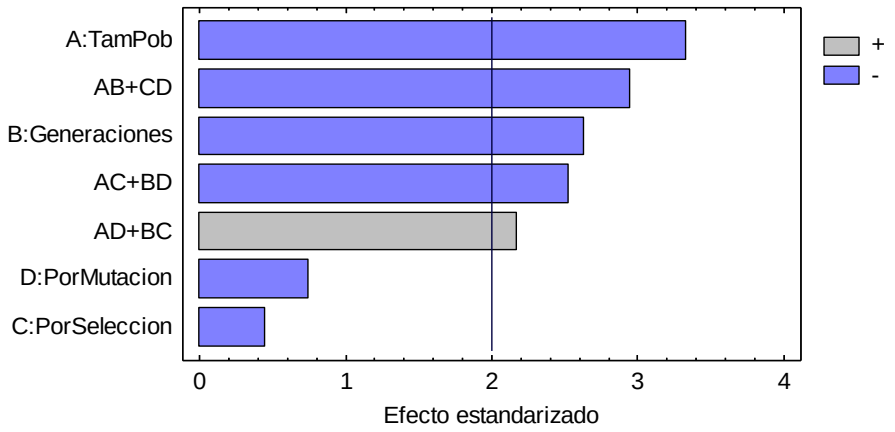


Tabla 25. Análisis de Varianza para el problema La17

Fuente	Suma de Cuadrados	Grados de Libertad	Cuadrado Medio	Razón-F	Valor-P (Significancia)
A:TamPob	1732,64	1	1732,64	11,08	0,0016
B:Generaciones	1080,77	1	1080,77	6,91	0,0111
C:PorSeleccion	31,6406	1	31,6406	0,20	0,6546
D:PorMutacion	87,8906	1	87,8906	0,56	0,4567
AB+CD	1359,77	1	1359,77	8,69	0,0047
AC+BD	1000,14	1	1000,14	6,39	0,0143
AD+BC	735,766	1	735,766	4,70	0,0344
Error total	8760,88	56	156,444		
Total (corregido)	14789,5	63			

En la Tabla 25 se observa que el efecto del tamaño de la población, el número de generaciones y las interacciones es significativo en la respuesta al tener un valor-P menor que 0.05, los efectos principales C (Porcentaje de selección) y D (Porcentaje de mutación) no son significativos. Ya que el Porcentaje de selección es el que menos incide sobre el Makespan se espera que tampoco contribuya en las interacciones, por tanto los efectos CD, AC y BC son nulos; se concluye que las interacciones que inciden son AB, BD y AD. El makespan se minimiza cuando los efectos principales significativos se encuentran en su nivel alto, las combinaciones en las interacciones que permiten una disminución en la variable de salida son A^+B^+, A^+D^- y B^+D^+ .

En los problemas La06, La11 y La14 los valores encontrados del Makespan para cada instancia son iguales, por tanto no se pudo realizar el análisis de varianza ya que no es posible estimar el efecto de los factores en cada problema sin variabilidad en los datos.

7.3.5. Análisis de varianza (ANOVA) para el problema Ft10

Figura 23. Diagrama de Pareto de efectos estimados para Makespan del problema Ft10

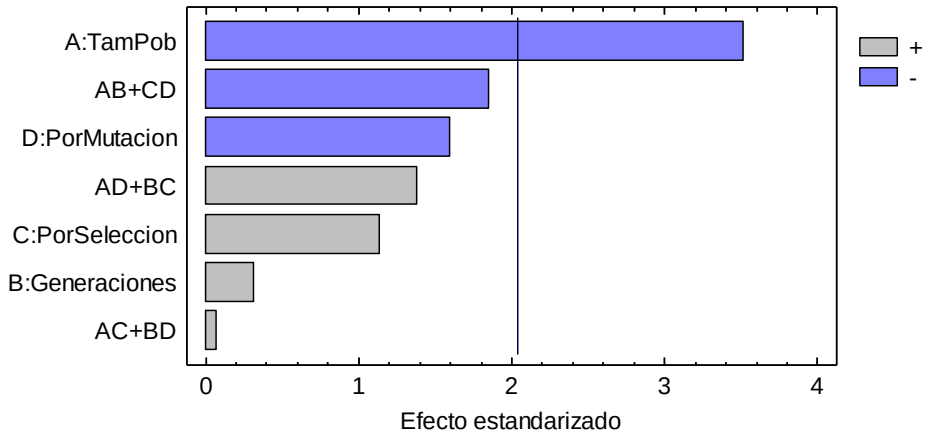


Tabla 26. Análisis de Varianza para el problema Ft10

Fuente	Suma de Cuadrados	Grados de Libertad	Cuadrado Medio	Razón-F	Valor-P (Significativo)
A:TamPob	1651,23	1	1651,23	12,40	0,0013
B:Generaciones	13,225	1	13,225	0,10	0,7547
C:PorSeleccion	172,225	1	172,225	1,29	0,2640
D:PorMutacion	342,225	1	342,225	2,57	0,1188
AB+CD	455,625	1	455,625	3,42	0,0737
AC+BD	0,625	1	0,625	0,00	0,9458
AD+BC	255,025	1	255,025	1,91	0,1760
Error total	4262,8	32	133,213		
Total (corregido)	7152,98	39			

En la Tabla 26 se observa que únicamente el efecto generado por el tamaño de la población (A) es significativo en la variable Makespan al ser menor que 0.05, al usar el nivel alto en el tamaño de la población el makespan se minimiza. El factor que menos incide en la respuesta es el número de generaciones, los demás

efectos principales y las interacciones de estos no inciden en la respuesta del problema.

7.3.6. Análisis de Varianza (ANOVA) para el problema La16

Figura 24. Diagrama de Pareto de efectos estimados para Makespan del problema La16

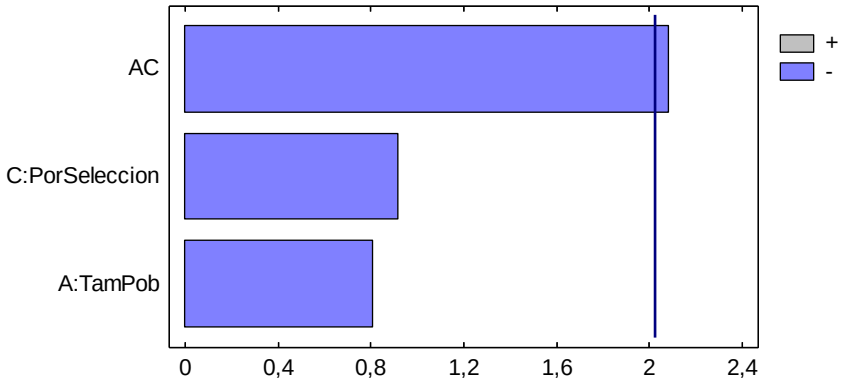


Tabla 27. Análisis de Varianza para el problema La16

Fuente	Suma de Cuadrados	Grados de Libertad	Cuadrado Medio	Razón-F	Valor-P (Significancia)
A:TamPob	46,225	1	46,225	0,65	0,4251
C:PorSeleccion	60,025	1	60,025	0,85	0,3641
AC	308,025	1	308,025	4,34	0,0445
Error total	2557,1	36	71,0306		
Total (corregido)	2971,38	39			

De la Tabla 27 se puede concluir que solo la interacción entre el Tamaño de la población y el Porcentaje de selección tiene un efecto significativo sobre la variable de salida Makespan al tener un valor-P menor que 0.05, los demás efectos principales y las interacciones de estos no inciden en la respuesta del problema. Los factores Porcentaje de Mutación y Generaciones fueron excluidos, igual que las interacciones en las que participan porque sus efectos son los más

pequeños, esta eliminación determinó que AC contribuye a explicar el comportamiento de la variable Makespan y que ésta se minimiza cuando se usa el nivel alto en ambos factores (150 como tamaño de la población y 0.9 en el Porcentaje de selección).

7.3.7. Análisis de Varianza (ANOVA) para el problema Ft 20

Figura 25. Diagrama de Pareto de efectos estimados para Makespan del problema Ft 20

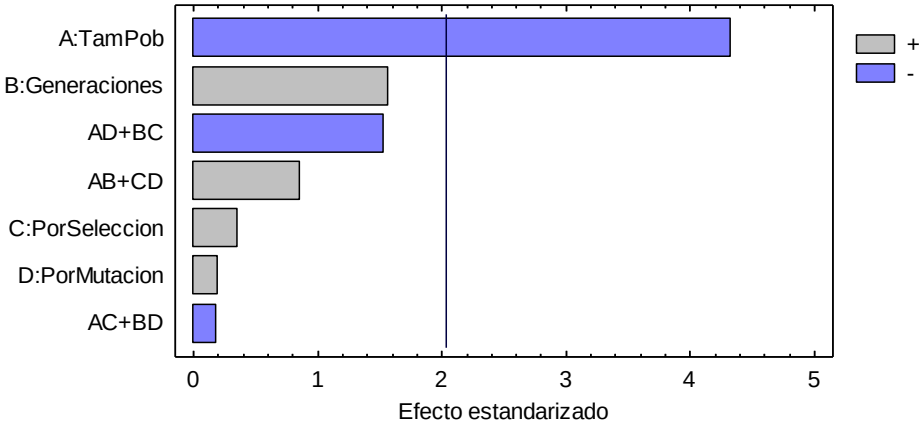


Tabla 28. Análisis de Varianza para el problema Ft 20

Fuente	Suma de Cuadrados	Grados de Libertad	Cuadrado Medio	Razón-F	Valor-P (Significancia)
A:TamPob	11628,1	1	11628,1	18,70	0,0001
B:Generaciones	1537,6	1	1537,6	2,47	0,1256
C:PorSeleccion	78,4	1	78,4	0,13	0,7248
D:PorMutacion	22,5	1	22,5	0,04	0,8503
AB+CD	462,4	1	462,4	0,74	0,3949
AC+BD	19,6	1	19,6	0,03	0,8602
AD+BC	1464,1	1	1464,1	2,35	0,1347
Error total	19895,2	32	621,725		
Total (corregido)	35107,9	39			

De la Tabla 28 se puede concluir que únicamente el efecto del tamaño de la población (A) es significativo en la variable de salida Makespan al ser inferior a

0,05. El nivel alto del tamaño de la población encuentra los mejores resultados de la variable de salida. Los demás efectos principales y las interacciones de estos no inciden en la respuesta del problema.

7.3.8. Análisis de Varianza (ANOVA) para el problema La21

Figura 26. Diagrama de Pareto de efectos estimados para Makespan del problema La25

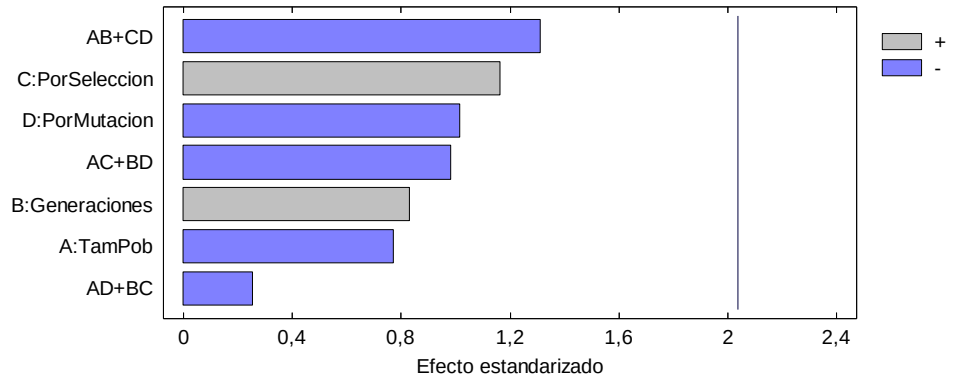


Tabla 29. Análisis de Varianza para el problema Ft 20

Fuente	Suma de Cuadrados	Grados de Libertad	Cuadrado Medio	Razón-F	Valor-P (Significancia)
A:TamPob	220,042	1	220,042	0,60	0,4441
B:Generaciones	255,025	1	255,025	0,70	0,4104
C:PorSeleccion	497,025	1	497,025	1,36	0,2528
D:PorMutacion	378,225	1	378,225	1,03	0,3173
AB+CD	632,025	1	632,025	1,72	0,1985
AC+BD	354,025	1	354,025	0,97	0,3331
AD+BC	24,025	1	24,025	0,07	0,7996
Error total	11728,8	32	366,525		
Total (corregido)	14380,4	39			

De la Tabla 29 se puede concluir que tanto las interacciones como los efectos principales no inciden en la variable de respuesta ya que el valor de significancia para todos está por encima de 0.05.

7.3.1. Análisis de Varianza (ANOVA) para el problema La24

Figura 27. Diagrama de Pareto de efectos estimados para Makespan del problema La24

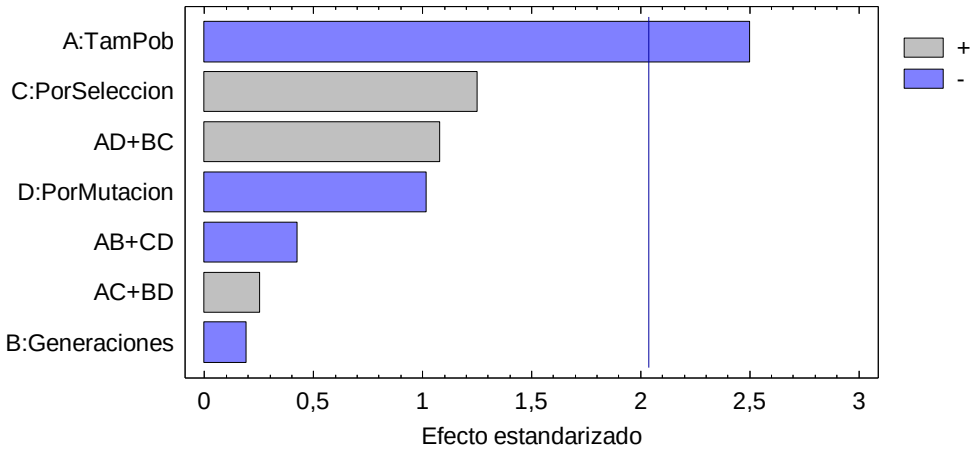


Tabla 30. Análisis de Varianza para el problema La24

Fuente	Suma de Cuadrados	Grados de Libertad	Cuadrado Medio	Razón-F	Valor-P (Significancia)
A:TamPob	1392,4	1	1392,4	6,26	0,0177
B:Generaciones	8,1	1	8,1	0,04	0,8499
C:PorSeleccion	348,1	1	348,1	1,56	0,2200
D:PorMutacion	230,4	1	230,4	1,04	0,3164
AB+CD	40,0	1	40,0	0,18	0,6744
AC+BD	14,4	1	14,4	0,06	0,8008
AD+BC	260,1	1	260,1	1,17	0,2876
Error total	7118,0	32	222,438		
Total (corregido)	9411,5	39			

De la Tabla 30 se puede concluir que únicamente el efecto del tamaño de la población (A) es significativo en la variable de salida Makespan al ser menor que

0,05. El nivel alto del tamaño de la población encuentra los mejores resultados y el efecto menos significativo es el número de generaciones.

7.3.2. Análisis de Varianza (ANOVA) para el problema La25

Figura 28. Diagrama de Pareto de efectos estimados para Makespan del problema La25

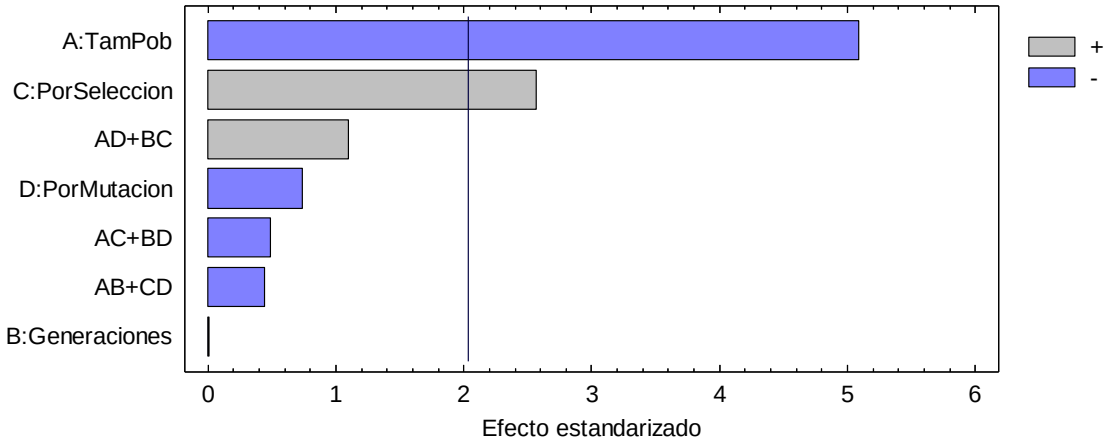


Tabla 31. Análisis de Varianza para el problema La25

Fuente	Suma de Cuadrados	Grados de Libertad	Cuadrado Medio	Razón-F	Valor-P (Significancia)
A:TamPob	5040,02	1	5040,02	25,93	0,0000
B:Generaciones	0,025	1	0,025	0,00	0,9910
C:PorSeleccion	1288,22	1	1288,22	6,63	0,0149
D:PorMutacion	105,625	1	105,625	0,54	0,4664
AB+CD	38,025	1	38,025	0,20	0,6613
AC+BD	46,225	1	46,225	0,24	0,6291
AD+BC	235,225	1	235,225	1,21	0,2795
Error total	6220,4	32	194,387		
Total (corregido)	12973,8	39			

En la Tabla 31 se observa que son 2 los efectos principales significativos en la respuesta al tener un valor-P menor que 0.05; el tamaño de la población (A) y el porcentaje de selección (C), los mejores resultados para el makespan se obtiene cuando TamPob está en el nivel alto y PorSeleccion en el nivel bajo; el efecto

principal menos significativo es el número de generaciones. El efecto AC no incide en la respuesta, de la misma manera sucede con los demás factores analizados y las interacciones entre ellos.

7.3.1. Análisis de Varianza (ANOVA) para el problema La38

Figura 29. Diagrama de Pareto de efectos estimados para Makespan del problema La38

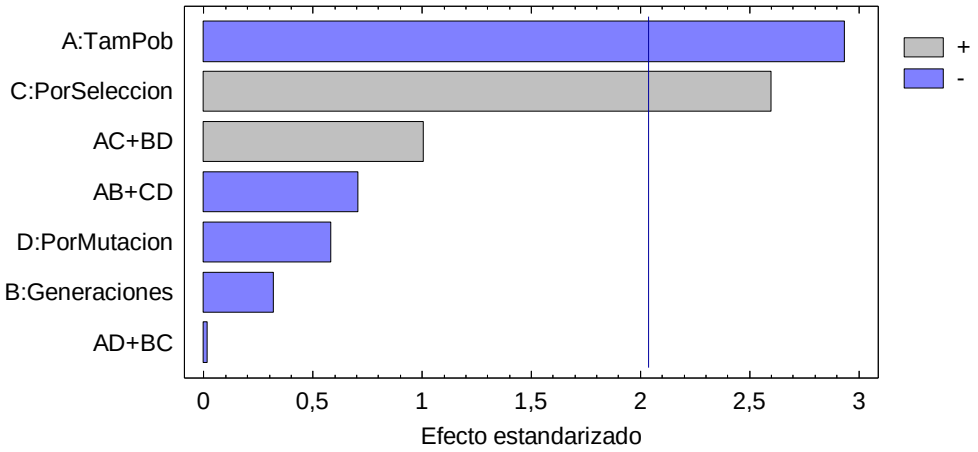


Tabla 32. Análisis de Varianza para el problema La38

Fuente	Suma de Cuadrados	Grados de Libertad	Cuadrado Medio	Razón-F	Valor-P (Significancia)
A:TamPob	2755,6	1	2755,6	8,62	0,0061
B:Generaciones	32,4	1	32,4	0,10	0,7522
C:PorSeleccion	2160,9	1	2160,9	6,76	0,0140
D:PorMutacion	108,9	1	108,9	0,34	0,5635
AB+CD	160,0	1	160,0	0,50	0,4843
AC+BD	324,9	1	324,9	1,02	0,3208
AD+BC	0,1	1	0,1	0,00	0,9860
Error total	10225,2	32	319,538		
Total (corregido)	15768,0	39			

En la Tabla 32 se observa que son 2 efectos principales los significativos en la respuesta al tener un valor-P menor que 0.05, el tamaño de la población (A) y el

porcentaje de selección (C), los mejores resultados para el makespan se obtienen cuando TamPob está en el nivel alto y PorSeleccion en el nivel bajo, el efecto principal menos significativo es el número de generaciones. Los demás factores analizados y las interacciones entre ellos no inciden en la variable de respuesta.

8. RESULTADOS COMPUTACIONALES

El MA propuesto se ejecutó para solucionar el JSP, el diseño de experimentos permitió ajustar el valor de los parámetros del algoritmo y así encontrar en algunos casos mejores soluciones a las halladas en la validación y la experimentación. En la Tabla 33 se muestran los resultados encontrados con el MA para 30 problemas del benchmark tomados de la OR-Library; los resultados se contrastan con los reportados por otros enfoques demostrando la competitividad del MA. En el Anexo I se encuentra el schedule correspondiente al valor hallado para cada instancia.

Tabla 33. Comparación entre los resultados encontrados con el MA y los hallados por otros enfoques.

Problema	Tamaño	Tam Pob	Gen	%Sel	%Mut	MC	MA Propuesto	MA(GR-RS) (Hasan et al.)	TS (Geyik F. y Cedimoglu I.)	PGA(Park et al.)	%Error
Ft06	6x6	10	10	0,7	0,1	55	55	55	55	55	0,00%
Ft10	10x10	500	100	0,9	0,1	930	937	930	971	936	0,75%
Ft20	20x5	500	80	0,9	0,1	1165	1182	1165	1165	1177	1,46%
La01	10x5	20	20	0,9	0,1	666	666	666	666	666	0,00%
La02	10x5	50	50	0,9	0,1	655	655	655	655	666	0,00%
La03	10x5	200	100	0,9	0,1	597	597	597	604	597	0,00%
La04	10x5	300	150	0,9	0,1	590	590	590	598	590	0,00%
La05	10x5	5	1	0,7	0,05	593	593	593	593	593	0,00%
La06	15x5	10	1	0,9	0,1	926	926	926	936	926	0,00%
La07	15x5	20	20	0,9	0,1	890	890	890	910	890	0,00%
La08	15x5	30	20	0,9	0,1	863	863	863	863	863	0,00%
La09	15x5	10	10	0,9	0,05	951	951	951	951	951	0,00%
La10	15x5	10	2	0,7	0,1	958	958	958	1034	958	0,00%
La11	20x5	10	5	0,7	0,05	1222	1222	1222	1222	1222	0,00%
La12	20x5	10	5	0,9	0,1	1039	1039	1039	1039	1039	0,00%
La13	20x5	12	6	0,9	0,1	1150	1150	1150	1159	1150	0,00%
La14	20x5	10	5	0,7	0,05	1292	1292	1292	1374	1292	0,00%
La15	20x5	50	50	0,9	0,1	1207	1207	1207	1207	1207	0,00%

Problema	Tamaño	Tam Pob	Gen	%Sel	%Mut	MC	MA Propuesto	MA(GR-RS) (Hasan et al.)	TS (Geyik F. y Cedimoglu I.)	PGA(Park et al.)	%Error
La16	10x10	80	170	0,8	0,1	945	946	945	959	977	0,11%
La17	10x10	40	50	0,9	0,1	784	784	784	784	787	0,00%
La18	10x10	150	170	0,9	0,1	848	858	848	861	848	1,18%
La21	15x10	150	170	0,9	0,1	1046	1081	1079	1099	1047	3,35%
La22	15x10	300	80	0,9	0,1	927	954	960	962	936	2,91%
La23	15x10	350	100	0,9	0,1	1032	1032	1032	1032	1032	0,00%
La24	15x10	150	100	0,8	0,1	935	976	959	989	955	4,39%
La25	15x10	300	100	0,9	0,1	977	999	991	995	1004	2,25%
La31	30x10	250	60	0,9	0,1	1784	1784	1784	1784	1784	0,00%
La32	30x10	280	80	0,9	0,1	1850	1865	1850	1850	1850	0,81%
La35	30x10	300	70	0,9	0,1	1888	1901	1888	1888	1888	0,69%
La38	15x15	250	100	0,8	0,1	1196	1258	1266	1254	1248	5,18%

9. CONCLUSIONES

Los objetivos planteados para esta investigación se cumplieron en su totalidad. El algoritmo memético propuesto encuentra las mejores soluciones conocidas para la minimización del makespan en instancias de baja y mediana complejidad y soluciones cercanas a las mejores encontradas aplicando otros enfoques en las instancias de alta complejidad, demostrando ser competitivo ante otros algoritmos presentados en la literatura.

Se pudo comprobar que el algoritmo memético aprovecha la capacidad de la estrategia genética que realiza la exploración del espacio de solución y el método de búsqueda local que intensifica la búsqueda explotando cada una de las soluciones obtenidas para evitar quedar atrapado en un óptimo local, haciéndolo más eficiente que la estrategia genética pura.

El tipo de representación genética siempre genera Schedules factibles por tanto permite usar un método aleatorio para crear los individuos iniciales, con el propósito de diversificar la población; además es sencillo de programar y usa menos tiempo computacional que la implementación de una heurística en la generación de la población, esto resulta muy competitivo para el caso de instancia de alta complejidad donde el tiempo de cómputo es elevado.

La estructura de vecindario de búsqueda local propuesta en la presente investigación es una modificación de la estrategia presentada por Nowicki y Smutnicki, se considera en el algoritmo propuesto la ruta crítica más larga y los intercambios planteados permiten generar hasta 3 vecinos a partir de la solución actual, aplicando un enfoque elitista para mantener las mejores soluciones en el desarrollo del algoritmo.

Un porcentaje de selección se aplica sobre la población para elegir los mejores individuos y diversificarla, todos los cromosomas seleccionados son apareados en la siguiente etapa por tanto se omite la probabilidad de cruce.

Para obtener el valor fitness y evaluar los individuos, la decodificación de las soluciones se hace mediante el diagrama de Gantt; este proceso se realiza a los vecinos generados en cada generación y después del cruce con el fin de evitar perder una buena solución al modificar los individuos.

A partir de la ejecución del algoritmo se concluye que:

- El proceso de decodificación es el que más tiempo computacional toma de todas la funciones empleadas, por tanto, es conveniente revisar si es posible realizar mejoras que permitan disminuir el tiempo de ejecución.

A partir de los resultados del diseño de experimentos se puede concluir que:

- El tamaño de la población es significativo en la mayoría de los problemas analizados ya sea como efecto principal o como interacción con otro parámetro.

- Para los problemas de Fisher y Thompson (Ft06, Ft10 y Ft20) el tamaño de la población es el único factor significativo sobre la variable de salida Makespan, el cambio de valores en este parámetro puede favorecer la minimización de la función objetivo. Los resultados señalan que de acuerdo a la complejidad del problema el tamaño de la población debe aumentar para generar mejores soluciones. En el problema Ft6 y Ft10 el parámetro que menos incide en la respuesta es el número de generaciones.

- Los resultados indican que el tamaño de la población está asociado con la capacidad de exploración del algoritmo propuesto.

- Los niveles del factor porcentaje de mutación no tienen efecto significativo sobre la variable de respuesta en las instancias analizadas, corroborando lo encontrado en la literatura acerca de la poca contribución en los resultados del algoritmo.

- En las instancias de alta complejidad no existe una interacción significativa entre los factores estudiados, excepto para el problema La16 donde la interacción entre el tamaño de la población y el porcentaje de selección es el único efecto significativo.

10. RECOMENDACIONES

El código de programación del algoritmo queda abierto al usuario para que sea tomado como base en el desarrollo de otros métodos más eficientes o modificaciones que permitan reducir el tiempo computacional en la determinación de soluciones.

Incentivar la ejecución de proyectos de grado que aborden investigaciones relacionadas con problemas tipo Job Shop, Flow Shop y sus variantes usando otros métodos evolutivos para la solución.

Desarrollar software a la medida para la pequeña y mediana industria usando el algoritmo propuesto altamente competitivo, como una herramienta económica y eficiente para la asignación de operaciones.

Incentivar el uso de lenguajes de programación en los estudiantes de ingeniería industrial que permita adquirir las competencias necesarias al momento de desarrollar este tipo de investigaciones.

Revisar y modificar la función de decodificación para reducir el tiempo computacional que toma su ejecución. Determinar si la omisión de esta función después del cruce tiene incidencia en la respuesta, de tal forma que el tiempo de ejecución se reduzca pero se mantenga la calidad de la solución.

En trabajos futuros adaptar esta heurística a otros problemas de Scheduling como Job Shop Flexible o con tiempos de setup para una aplicación más cercana a casos reales de la industria. Comprobar si la aplicación de métodos heurísticos como Giffler y Thompson en la generación de la población inicial puede reducir el espacio de solución y mejorar el tiempo de ejecución obteniendo soluciones igual de buenas o mejores a las encontradas con el algoritmo actual.

BIBLIOGRAFÍA

ADAMS, Joseph, BALAS, Egon y ZAWACK, Daniel. The shifting bottleneck procedure for the job-shop scheduling. En: Management Science. Vol. 34. No3 (1988) .p. 391–401

AKERS, Sheldon y FRIEDMAN Joyce. A Non-Numerical Approach to Production Scheduling Problems. En: Journal of the Operations Research Society of America. Vol. 3. No. 4 (1955) p. 422-429.

BALAS, Egon y VAZACOPOULOS, Alkis. Guided local search with shifting bottleneck for job-shop scheduling. En: Management Science. Vol. 44. No 2. (1998) p. 262-275

BEASLEY, John. OR-Library. Job Shop Scheduling. [Libreria de problemas en linea]. [consultado 1 de Agosto de 2013]. Disponible en: <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt>

BLUM, Christian y ROLI, Andrea. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. En: ACM Computing Surveys. Vol 35, N° 3. (2003), p. 268-308.

BÜLBÜL, Kerem. A hybrid shifting bottleneck-tabu search heuristic for the job shop total weighted tardiness problem. En: Computers & Operations Research. Vol 38 (2011) p. 967-983.

BURKARD, Rainer, Efficiently solvable special cases of hard combinatorial optimization problems, En: Mathematical Programming, 1997, Vol. 79, p 55-69.

CASTILLO, Guillermo y FANDIÑO, Oscar. Diseño de un modelo general para la planeación operativa y la secuenciación de actividades en pequeñas empresas con procesos de manufactura intermitentes. Bucaramanga, 2005, p 197. Tesis (Ingeniero Industrial). Universidad Industrial de Santander. Escuela de estudios industriales y empresariales

CAUMOND, Anthony; LACOMME, Philippe y TCHERNEV, Nikolay. A memetic algorithm for the job-shop with time-lags. En: Computers & Operations Research. Vol 35, (2008), p. 2331-2356.

CHENG, Hsueh-Chien; CHIANG, Tsung-che y FU, Li-Chen. A two-stage hybrid memetic algorithm for multiobjective job shop scheduling. En: Expert Systems with Applications. Vol. 38, (2011), p. 10983-10998.

CHENG, Runwey; GEN, Mitsuo Y TSUJIMURA Yasuhiro. A tutorial a survey of job shop scheduling problems using genetic algorithms: Representation. En: Computers and Industrial Engineering, Vol.30 No4, (1996) p 983-997.

CHIANG, Tsung-che; CHENG, Hsueh-Chien y FU, Li-Chen. NNMA: An effective memetic algorithm for solving multiobjective permutation flow shop scheduling problems. En: Expert Systems with Applications. Vol. 38, (2011), p. 5986-5989.

CONWAY, Richard W; MAXWELL, William L y MILLER, Louis W. Theory of scheduling. Addison-Wesley, 1967.

DAWKINS, Richard. The Selfish Gene. Clarendon, Oxford.1976

DORIGO, Marco. Optimization, Learning and Natural Algorithms. Milán, 1992. Tesis PhD. Departamento de Electrónica. Politécnico de Milano

DUECK, Gunter y SCHEUER, Tobias. Threshold accepting. A general purpose optimization algorithm appearing superior to simulated annealing. En: Journal of Computational Physics. Vol. 90. No 1. (1990).p. 161-175.

DURHAM, William. Co-Evolution: Genes, Culture and Human Diversity. Stanford University Press, 1992.

EBERHART, Russell y KENNEDY, James. A new optimizer using particle swarm theory, En: Proceedings of the Sixth International Symposium on Micro Machine and Human Science. Nagoya (1995) p. 39-43

ENGELBRECHT, Andries. Computational Intelligence An introduction, University of Pretoria South Africa, Editorial Wiley, 2º edición, 2007.

FEO, Thomas y RESENDEN, Mauricio. Greedy randomized adaptive search procedures. En: Journal of Global Optimization. Vol. 6. No 2. (1995) p. 109-133.

FOGEL, Lawrence, OWENS, Alvin y WALSH, Michael. Artificial Intelligence Through Simulated Evolution, John Wiley, UK, 1966.

FOGEL, Lawrence. Autonomous Automata. En: Industrial Research. Vol.4, Ed. 2, (1962), p.14-19.

FOO, Yoon y TAKEFUJI, Y. Stochastic neural networks for solvin job shop scheduling: part 1. Problem representation. En: IEEE International Conference on Neural Networks. San Diego, USA. Vol. 2. (1988).p 275-282

FRUTOS, Mariano y TOHME, Fernando. A Multi-objective Memetic Algorithm for the Job-Shop Scheduling Problem. En: Operational Research. (2012), p. 1-18.

FRUTOS, Mariano; OLIVERA, Ana Carolina y TOHMÉ, Fernando. A memetic algorithm based on a NSGAII scheme for the flexible job-shop scheduling problem. En: Annals of Operations Research. Vol. 181. No. 1 (2010), p. 745-765

GALLARDO, José; COTTA Carlos; FERNANDEZ, Antonio. Solving the Multidimensional Knapsack Problem Using an Evolutionary algorithm hybridized with branch and bound. En: MIRA, José y ÁLVAREZ,Jose. Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach. (2005) .21-30

GAO, Jie; GEN, Mitsuo y SUN, Linyan. Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. En: Journal of Intelligent Manufacturing. Vol. 17. No 4 (2006), p. 493-507

GAO, Liang et al. An efficient memetic algorithm for solving the job shop scheduling problem. En: Computers & Industrial Engineering. Vol 60. No 4 (2011), p.699-705

GAREY, Michael y JOHNSON, David. Computer and intractability: *A Guide to the Theory of NP-Completeness* .1979.

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

GEYIK, Faruk y CEDIMOGLU, Ismail. The strategies and parameters of tabu search for job-shop scheduling. En: Journal of Intelligent Manufacturing, Vol. 15 (2004) p. 439-448.

GIFFLER, B. y THOMPSON, G.L. Algorithms for solving production scheduling problems. En: Operations Research. Vol. 8. No 4 (1960), p.487-503

GLOVER, Fred y GREENBERG, Harvey. New approaches for heuristic search: A bilateral linkage with artificial intelligence. En: European Journal of Operational Research. Vol. 39. No 2 (1989). p.119-130

GLOVER, Fred. Future paths for integer programming and links to artificial intelligence. En: Computers & Operations Research. Vol 13 (1986); p. 533-549

GLOVER, Fred, y LAGUNA, Manuel. Tabu search. Boston: Kluwer academic publishers, Vol. 22 (1997). p. 3261-3362

GONZÁLEZ, Miguel; VELA, Camino y VARELA, Ramiro. A competent memetic algorithm for complex scheduling. En: Natural Computing. Vol. 11. No 1. (2012), p. 151-160

GRAHAM, R.L et. al. Optimization and approximation in deterministic sequencing and scheduling: a survey. En: Annals of Discrete Mathematics, Vol. 5. (1979). p.287–326

HAMALAINEN, Wilhemiina. Class NP, NP-complete, and NP-hard problems. [en linea]. (2006). [Consultado 12 Feb. 2013]. Disponible en < http://cs.joensuu.fi/pages/whamalai/daa/npsession.pdf >

HANSEN, Pierre y MLADENOVIC, Nenad. Variable neighborhood search: Principles and Applications. En: European Journal of Operational Research. Vol. 130. No 3. (2001). p. 449-467

HASAN, Kamrul et al. Memetic algorithms for solving job-shop scheduling problems. En: Memetic Computing. Vol. 1. No 1. (2009), p. 69-83

HOLLAND, John. Adaptation in natural and artificial systems. University of Michigan Press. Ann arbor, 1975.

JAIN, Anant y MEERAN, Sheik. Deterministic job-shop scheduling: Past, present. En: European Journal of Operational Research. Vol 113 (1999) p. 390-434.

JENSEN, Mikkel T. Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures. En: Applied Soft Computing, Vol. 1 (2001) p. 35-52

JOHNSON S.M. Optimal two- and three-stage production schedules with setup times included. En: Naval Research Logistics Quarterly. Vol. 1. No. 1 (1954) p.61–68.

JOUGLET, Antoine; OĞUZ, Ceyda y SEVAUX, Marc. Hybrid Flow-Shop: a Memetic Algorithm Using Constraint-Based Scheduling for Efficient Search. En: Journal of Mathematical Modelling and Algorithms. Vol. 8. No 3 (2009), p. 271-292

KIRKPATRICK, Scott; GELATT, C y VECCHI, M.P. Optimization by simulated annealing. En: Science. Vol. 220. No 4598 (1983). p. 671-680.

KOZA. John R, Genetic programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems, En: Technical Report

STAN-CS90-1314, Department of Computer Science, Stanford University, 1990.

LANGE, Kenneth. Applied Probability: Combinatorial Optimization. En: Springer Science Business Media. Cap. 5, (2010), p.103-122.

LIAN, Zhigang; JIAO, Bin y GU, Xingsheng. (2006) A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. En: Applied Mathematics and Computation. Vol. 183. No. 1 (2006). p. 1008-1017

MARTÍ, Rafael, Procedimientos Metaheurísticos en optimización combinatoria, [En línea] Universidad de Valencia, Departamento de Estadística e investigación operativa. (2003). p. 1-60 [Consultado 28 Ene. 2013].Disponible en: < http://www.uv.es/~rmarti/paper/cpapers.html>

MATTFELD, Dirk y BIERWIRTH, Christian. An efficient genetic algorithm for job shop scheduling with tardiness objectives. En: European Journal of Operational Research. Vol. 155 (2004). p. 616-630

MEERAN, Sheik y MORSHED, M. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study**.** En: Journal of Intelligent Manufacturing., Vol. 23, Ed. 3-4, (2012),p. 1063-1078.

MOSCATO, Pablo y COTTA, Carlos A Modern Introduction to Memetic Algorithms. En: GENDREAU, Michel y POTVIN, Jean-Yves. Handbook of Metaheuristics. Springer US, 2010. p. 141-183

_____ Una Introducción a los Algoritmos meméticos, En: Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, (2003), p 131-148.

MOSCATO, Pablo. On evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Toward Memetic Algorithm, En: Caltech Concurrent Computation Program, Report 826, Caltech, Pasadena CA. 1989.

NOWICKI, Eugeniusz y SMUTNICKI, Czeslaw. A fast taboo search algorithm for the job shop problem. En: Management Science Vol. 42.(1996), p.797-813

PAPADIMITRIOU, Christos. Computational Complexity. Addison-Wesley. 1994

PARK, Byung J; Choi, Hyung R y Kim, Hyun S. A hybrid genetic algorithm for the job shop scheduling problems. En: Computers & Industrial Engineering, Vol. 45 (2003), p. 597-613

PESANT, Gilles y GENDREAU, Michel. A view of local search in Constraint Programming. En: Principles and Practice of Constraint Programming. (1996)

PINEDO, Michael L. Job Shops (Deterministic). En: _____. Scheduling: Theory, Algorithms, and Systems. Springer New York, 2008. p. 179-216

QIAN et al. Scheduling multi-objective job shops using a memetic algorithm based on differential evolution. En: The International Journal of Advanced Manufacturing Technology. Vol. 35. No 9-10 (2008), p. 1014-1027

QUIN-DAO-ER-JI, Ren y WANG, Yuping. A new hybrid genetic algorithm for job shop scheduling problem. En: Computers and Operations Research. Vol. 39, (2012) p.2291-2299.

RAEESI, N. Mohammad Y KOBTI, Ziad. A Machine Operation Lists Based Memetic Algorithm for Job Shop Scheduling. En: Evolutionary Computation (CEC): IEEE Congress on. (2011), p. 2436-2443

RECHENBERG, Ingo. Evolutions strategie: Optimierung technischer Systeme nach Prinzipien der Biologischen Evolution. Frammann-Holzboog, Stuttgart, 1973.

REYNOLDS, Robert G. An Introduction to Cultural Algorithms: Proceedings of the Third Annual Conference on Evolutionary Programming, En: World Scientific, River Edge, New Jersey, (1994), p. 131–139

ROY, Bernard y SUSSMANN, B. Les probl`emes d'ordonnancement avec constraintes disjonctives.En:Technical Report. Paris:SEMA, 1964.

SABUNCUOGLU, Ihsan y GURGUN, Burckaan. A neural network model for scheduling problems. En: European Journal of Operational Research. Vol. 93. No. 2 (1996) .p. 288-299

SANCHEZ, Miguel, Optimización combinatorial. En: Matemáticas del siglo XX: Una mirada en 101 artículos [En línea] Universidad de la Laguna, 2000, p. 115-120 [Consultado 11 Feb. 2013].Disponible en: < http://www.sinewton.org/numeros/numeros/43-44/Articulo22.pdf>

SCHWEFEL, Hans-Paul. Evolutions strategie und numerische Optimierung, Technical University Berlin, 1975.

STORN, Rainer y PRICE, Kenneth. Diferential Evolution : A simple and efficient heuristic for global optimization over continuous spaces. En: Journal of Global Optimization vol. 11 (1997). p. 341-459.

STÜTZEL, Thomas. Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications. 1999.

TANG, Jianchao, et al. A Hybrid Algorithm for Flexible Job-shop Scheduling Problem. En: Procedia Ingeniering. Vol.15, (2011), p. 3678-3683

TARANTILIS, Christos y KIRANOUDIS, Chris. A list-based threshold accepting method for the job-shop scheduling problems. En: International Journal of Production Economics . Vol 77, No 2. (2002); p.158-171.

VAN LAARHOVEN et al. Job Shop Scheduling by Simulated Annealing. En: Operations Research. Vol. 40. N° 1.(1992) p. 113-125.

YANG, et al. Jin-hui. Clonal Selection Based Memetic Algorithm for Job Shop Scheduling Problems. En: Journal of Bionic Engineering. Vol. 5, (2008), p. 111-119.

ZHANG, Chaoyong; RAO, Yunqing y LI, Peigen. An effective hybrid genetic algorithm for the job shop scheduling problem. En: The International Journal of Advanced Manufacturing Technology. Vol. 39, Ed. 9-10, (2008) p. 965-974.

ZHANG, Liping; LI, Xinyu y WEN, Long. Scheduling flexible job shop in dynamic environment based on a memetic algorithm. En: Cognitive Informatics & Computing: IEEE 11th International Conference on. (2012), p. 407-412.

ZOBOLAS G; TARANTILIS, Christos y IOANNOU, George. Exact, Heuristic and Meta-heuristic Algorithms for Solving Shop Scheduling Problems. En: XHAFA, Fatos y ABRAHAM, Ajith. Metaheuristics for Scheduling in Industrial and Manufacturing Applications. Springer Berlin Heidelberg, 2008. p. 1- 40

ZUO, Xingquan; WANG, Chunlu y TAN Wei. Two heads are better than one: an AIS- and TS-based hybrid strategy for job shop scheduling problems. En: The International Journal of Advanced Manufacturing Technology. Vol. 63. No 1-4 (2012), p. 155-168

ANEXOS

ANEXO A. APLICACIÓN DEL HEURÍSTICO DE GIFFLER Y THOMPSON EN UNA INSTANCIA 3X3

En el siguiente ejemplo se muestra el procedimiento del algoritmo G&T para la generación de schedules factibles activos en una instancia 3x3 (3 trabajos y 3 máquinas).

La Tabla 34 contiene la información del problema: orden de secuenciamiento y tiempos de procesamiento de cada operación en su respectiva máquina.

Tabla 34. Ejemplo instancia 3x3

Trabajos	Secuencia de máquinas	Tiempo de Procesamiento
J_1	m_1 - m_2 - m_3	3 -2- 5
J_2	m_1 - m_3 - m_2	3 -5 - 2
J_3	m_2 - m_1 - m_3	2 -5 - 3

El índice de prioridad será tomado en cuenta de acuerdo a la regla de prioridad SPT (Tiempo de procesamiento más corto), para ello se genera aleatoriamente un cromosoma C_k con base en la representación basada en listas de preferencias (ver Anexo C).

$$C_k = \{(231)\ (132)\ (213)\}$$
$$\downarrow \quad \downarrow \quad \downarrow$$
$$m_1 \quad m_2 \quad m_3$$

La secuencia de operaciones que son actualizadas en el Schedule parcial, se basan en la información del problema presentado en la Tabla 34.

Secuencia de operaciones (trabajo, máquina)

Para el trabajo 1 (J_1): (1,1) → (1,2) → (1,3)

Para el trabajo 2 (J_2): (2,1) → (2,3) → (2,2)

Para el trabajo 3 (J_3): (3,2) → (3,1) → (3,3)

Inicialización

El conjunto de schedules parciales inicia vacío, es decir, S=Ø y $s_{(i,j)}$ =0, y se asignan al conjunto Ω las operaciones que no tienen predecesor.

Ω= {(1,1), (2,1), (3,2)}

1º Iteración

- Calcular el tiempo de finalización más temprano para cada operación del conjunto Ω, $f_{(i,j)} = s_{(i,j)} + p_{(i,j)}$ y escoger el mínimo valor.

  $f_{(1,1)} = 0 + 3 = 3 \quad f_{(2,1)} = 0 + 3 = 3 \quad f_{(3,2)} = 0 + 2 = 2$

  $f^* = min\,(f_{(1,1)}, f_{(2,1)}, f_{(3,2)}) = 2$

  Asignar m^* a la máquina correspondiente. Para el ejemplo y el valor de f^* calculado, corresponde a la operación (3,2), es decir, el trabajo J3 en la máquina 2. Por tanto, $m^*=2$.

- Hallar las demás operaciones que pertenecen al conjunto Ω y que también deben ser realizadas en la máquina *2*.

  Operaciones= {(3,2)}

  En el caso que existiera más de una operación a ejecutar en la máquina 2, se elige la que tenga mayor preferencia, basada en el cromosoma C_k. La operación escogida es (3,2).

  Agregar la operación (3,2) al Schedule parcial *S*, es decir, *S= {(3,2)}*

- Actualizar el valor $s_{(i,j)}$ por el tiempo de finalización $f_{(3,2)}$ de la operación (3,2), de la siguiente manera $s_{(3,2)}=2$. Este nuevo valor se debe tener en cuenta en la siguiente iteración cuando se vaya a calcular los $f_{(i,j)}$ de las operaciones que pertenezcan al trabajo 3 o aquellas que son ejecutadas en la máquina 2.

- Se elimina la operación que fue programada del conjunto Ω, es decir, la operación (3,2) y se añade su sucesor inmediato de acuerdo a la secuencia de operaciones que para el ejemplo corresponde a (3,1). El nuevo Ω queda de la siguiente manera:

  Ω= *{(1,1), (2,1), (3,1)}*

  Se realiza nuevamente los pasos descritos anteriormente, hasta obtener el Schedule completo, es decir, hasta que se hayan programado todas las operaciones.

2° Iteración

- $f_{(1,1)} = 0 + 3 = 3$ $f_{(2,1)} = 0 + 3 = 3$ $f_{(3,1)} = 2 + 5 = 7$

  $f^* = min\,(f_{(1,1)}, f_{(2,1)}, f_{(3,1)}) = 3$ $m^* = 1$

  Se elige una operación de manera aleatoria, para este caso será (2,1).

- *{(1,1)(2,1)(3,1)}*
  De acuerdo a la lista de preferencia en la máquina 1, se elige la operación (2,1).
- *S={(3,2),(2,1)}*
- $s_{(2,1)} = 3$
- Ω= {(1,1), (2,3), (3,1)}

3° Iteración

- $f_{(1,1)} = 3 + 3 = 6$ $f_{(2,3)} = 3 + 5 = 8$ $f_{(3,1)} = 3 + 5 = 8$

  $f^* = min\ (f_{(1,1)}, f_{(2,3)}, f_{(3,1)}) = 6$ $m^* = 1$

- {(1,1)}
- (1,1)
- *S={(3,2),(2,1),(1,1)}*

- $s_{(1,1)}$=6
- Ω= {(1,2), (2,3), (3,1)}

4° Iteración

- $f_{(1,2)} = 6 + 2 = 8 \quad f_{(2,3)} = 3 + 3 = 6 \quad f_{(3,1)} = 6 + 5 = 11$

$f^* = min\ (f_{(1,2)}, f_{(2,3)}, f_{(3,1)}) = 6 \quad m^* = 3$

- {(2,3)}
- (2,3)
- *S={(3,2),(2,1),(1,1),(2,3)}*
- $s_{(2,3)}$=6
- Ω= {(1,2), (2,2), (3,1)}

5° Iteración

- $f_{(1,2)} = 6 + 2 = 8 \quad f_{(2,2)} = 6 + 2 = 8 \quad f_{(3,1)} = 6 + 5 = 11$

$f^* = min\ (f_{(1,2)}, f_{(2,2)}, f_{(3,1)}) = 8 \quad m^* = 2$

- {(1,2),(2,2)}
- (1,2)
- *S={(3,2),(2,1),(1,1),(2,3),(1,2)}*
- $s_{(1,2)}$=8
- Ω= {(1,3), (2,2), (3,1)}

6° Iteración

- $f_{(1,3)} = 8 + 5 = 7 \quad f_{(2,2)} = 6 + 2 = 8 \quad f_{(3,1)} = 6 + 5 = 11$

$f^* = min\ (f_{(1,3)}, f_{(2,2)}, f_{(3,1)}) = 7 \quad m^* = 3$

- {(1,3)}

- (1,3)
- *S={(3,2),(2,1),(1,1),(2,3),(1,2),(1,3)}*
- $s_{(1,3)}$*=7*
- Ω= {(2,2), (3,1)}

7° Iteración

- $f_{(2,2)} = 6 + 2 = 8$ $f_{(3,1)} = 6 + 5 = 11$

  $f^* = min\,(f_{(2,2)}, f_{(3,1)}) = 8$ $m^* = 2$

- {(2,2)}
- (2,2)
- *S={(3,2),(2,1),(1,1),(2,3),(1,2),(1,3),(2,2)}*
- $s_{(2,2)}$*=8*
- Ω= {(3,1)}

8° Iteración

- $f_{(3,1)} = 6 + 5 = 11$
- $f^* = min\,(f_{(3,1)}) = 11$ $m^* = 1$
- (3,1)
- *S={(3,2),(2,1),(1,1),(2,3),(1,2),(1,3),(2,2),(3,1)}*
- $s_{(3,1)}$*=11*
- Ω= {(3,3)}

9° Iteración

- $f_{(3,3)} = 11 + 5 = 16$

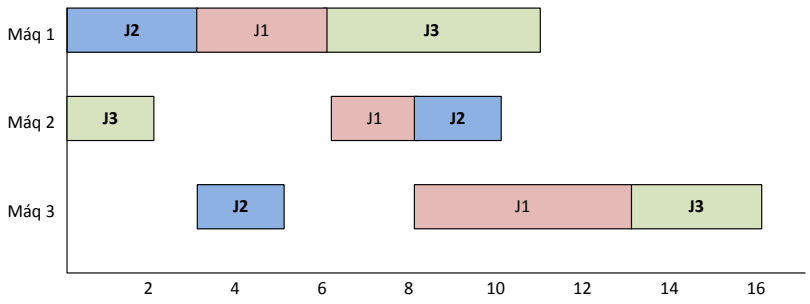
  $f^* = min\,(f_{(3,3)}) = 16$ $m^* = 3$

- {(3,3)}
- (3,3)
- *S={(3,2),(2,1),(1,1),(2,3),(1,2),(1,3),(2,2),(3,1),(3,3)}*
- $s_{(3,3)}$*=16*
- Ω= { Ø }

Una vez ha sido generado un Schedule completo, se procede a realizar la decodificación por medio del diagrama de Gantt para obtener el valor de la función objetivo, en este caso, el makespan.

> Schedule Completo= *{(3,2), (2,1), (1,1), (2,3), (1,2), (1,3), (2,2), (3,1), (3,3)}*

Figura 30. Decodificación del Schedule completo obtenido mediante la heurística G&T



Programando una a una las operaciones de acuerdo a la secuencia arrojada en el Schedule completo se obtiene un makespan de 16 unidades de tiempo.

ANEXO B. APLICACIÓN HEURÍSTICA CUELLO DE BOTELLA MÓVIL

En el siguiente ejemplo se desarrolla el procedimiento del algoritmo Cuello de Botella Móvil para una instancia de 3 máquinas y 3 trabajos.

Para este ejemplo se usa la siguiente nomenclatura:

p_{ji}: tiempo de procesamiento del trabajo i en la máquina j.

r_{ji}: tiempo de liberación del trabajo i en la máquina j

dji: tiempo de finalización esperado para el trabajo i en la máquina j.

Ci: tiempo en que se completa el trabajo i

Li: retraso del trabajo i

$L\ ((Oji),F)$: camino más largo desde Oji hasta el nodo sumidero F

$Lmax\ (j)$: mínimo tiempo de tardanza en la máquina j

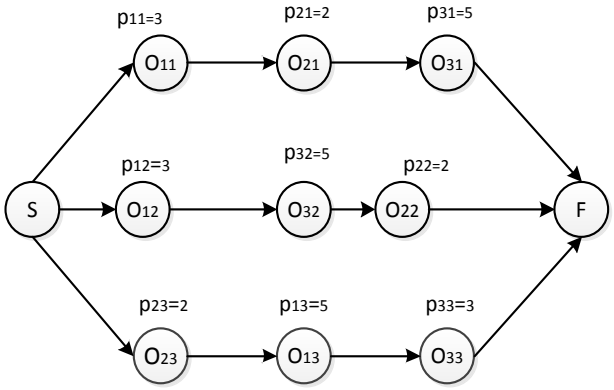
$Cmax$: makespan del problema

En la Tabla 34 se encuentran los datos con los que se desarrolla este ejemplo.

Se tiene un conjunto de máquinas M= {m_1, m_2, m_3}, un conjunto Mo= { $\varnothing$ } y un grafo disyuntivo del problema sin aristas de recursos con los tiempos de procesamiento de cada trabajo, en este caso el nodo Oji es el trabajo i en la máquina j.

Primera iteración: Inicialmente el conjunto Mo está vacío. El makespan se determina por la ruta más larga desde el nodo S hasta el nodo F; como no se ha programado ninguna máquina el $Cmax$= 10 por cualquiera de las rutas que se elija.

Figura 31. Grafo disyuntivo sin asignación de recursos para el problema de la Tabla 34.



Cada máquina se considera un problema independiente $1|rj|Lmax$. Se analizan todas las combinaciones de los trabajos y se elige la secuencia que genera el menor $Lmax\ (j)$.

Para cada máquina se hallan $r_{ji}\ dji, Ci\ y\ Li$. El Lmax de cada secuencia será el mayor Li hallado con esa programación.

- **Máquina 1**

r_{ji} es la ruta más larga desde el nodo S hasta el nodo Oji en la Figura 31.

r11=0 r12=0 r13=2.

dji= $Cmax$-$L\ ((Oji), F) + pji$

d11=10-10+3=3 d12=10-10+3=3 d13=10-8+5=7

Para determinar el tiempo Ci se usa $C_i = \begin{cases} p_{ji} + C_{i-1} \text{ , si } C_{i-1} > r_{ji} \\ p_{ji} + r_{ji} \text{ , si } C_{i-1} < r_{ji} \end{cases}$

El tiempo de tardanza es Li=Ci-dji

Secuencia 1-2-3

Jobs	J1	J2	J3
pji	3	3	5
rji	0	0	2
dji	3	3	7
Ci	3	6	11
Li	0	3	4

C1=3+0=3 C2=3+3=6 C3=5+6=11

L1=3-3=0 L2=6-3=3 L3=11-7=4

Lmax (1) =4

El Lmax es 4 porque es el mayor valor entre los Li encontrados en esta combinación de trabajos.

Secuencia 1-3-2

Jobs	J1	J3	J2
pji	3	5	3
rji	0	2	0
dji	3	7	3
Ci	3	8	11
Li	0	1	8

C1=3+0=3 C2=3+5=8 C3=3+8=11

L1=3-3=0 L2=8-7=1 L3=11-3=8

Lmax (1) =8

Secuencia 2-1-3

Jobs	J2	J1	J3
pji	3	3	5
rji	0	0	2
dji	3	3	7
Ci	3	6	11
Li	0	3	4

C1=3+0=3 C2=3+3=6 C3=6+5=11

L1=3-3=0 L2=6-3=3 L3=11-7 =4

Lmax (1)=4

Secuencia 2-3-1

Jobs	J2	J3	J1
pji	3	5	3
rji	0	2	0
dji	3	7	3
Ci	3	8	11
Li	0	1	8

C1=3+0=3 C2=3+5=8 C3=3+8=11

L1=3-3=0 L2=8-7=1 L3=11-3=8

Lmax (1) =8

El Lmax es 8 porque es el mayor valor entre los Li encontrados en esta combinación de trabajos.

Secuencia 3-1-2

Jobs	J3	J1	J2
pji	5	3	3
rji	2	0	0
dji	7	3	3
Ci	7	10	13
Li	0	7	10

C1=5+2=7 C2=3+7=1 C3=3+10=13

L1=7-7=0 L2=10-3=7 L3=13-3=10

Lmax (1) =10

Secuencia 3-2-1

Jobs	J3	J2	J1
pji	5	3	3
rji	2	0	0
dji	7	3	3
Ci	7	10	13
Li	0	7	10

C1=5+2=7 C2=3+7=1 C3=3+10=13

L1=7-7=0 L2=10-3=7 L3=13-3=10

Lmax (1) =10

El mínimo **Lmax (1) es 4,** dos secuencias generan este valor por tanto se elige una de ellas arbitrariamente, 2-1-3 es la secuencia de los trabajos seleccionada.

- **Máquina 2**

r_{ji} es la ruta más larga desde el nodo S hasta el nodo Oji en la Figura 31.

r21=3 r22=8 r23=0

$dji = Cmax - L\,((Oji), F) + pji$

d21=10-7+2=5 d22=10-2+2=10 d23=10-10+2=2

Para determinar el tiempo Ci se usa $C_i = \begin{cases} p_{ji} + C_{i-1} \text{ , si } C_{i-1} > r_{ji} \\ p_{ji} + r_{ji} \text{ , si } C_{i-1} < r_{ji} \end{cases}$

El tiempo de tardanza es $Li=Ci-dji$

Secuencia 1-2-3

Jobs	J1	J2	J3
pji	2	2	2
rji	3	8	0
dji	5	10	2
Ci	5	10	12
Li	0	0	10

C1=2+3=5 C2=2+8=10 C3=2+10=12

L1=5-5=0 L2=10-10=7 L3=12-2=10

Lmax (2) =10

Secuencia 1-3-2

Jobs	J1	J3	J2
pji	2	2	2
rji	3	0	8
dji	5	2	10
Ci	5	7	10
Li	0	5	0

C1=2+3=5 C2=2+5=7 C3=2+8=10

L1=5-5=0 L2=7-2=5 L3=10-10=0

Lmax (2) = 5

El Lmax es 5 porque es el mayor valor entre los Li encontrados en esta combinación de trabajos.

Secuencia 2-1-3

Jobs	J2	J1	J3
pji	2	2	2
rji	8	3	0
dji	10	5	2
Ci	10	12	14
Li	0	7	12

C1=2+8=10 C2=2+10=12 C3=2+12=14

L1=10-10=0 L2=12-5=7 L3=14-2=12

Lmax (2) =12

Secuencia 2-3-1

Jobs	J2	J3	J1
pji	2	2	2
rji	8	0	3
dji	10	2	5
Ci	10	12	14
Li	0	10	9

C1=2+8=10 C2=2+10=12 C3=2+12=14

L1=10-10=0 L2=12-2=10 L3=14-5=9

Lmax (2) =10

El Lmax es 10 porque es el mayor valor entre los Li encontrados en esta combinación de trabajos

Secuencia 3-1-2

Jobs	J3	J1	J2
pji	2	2	2
rji	0	3	8
dji	2	5	10
Ci	2	5	10
Li	0	0	0

C1=2+0=2 C2=2+3=5 C3=2+8=10

L1=2-2=0 L2=5-5=0 L3=10-10=0

Lmax (2) =0

Secuencia 3-2-1

Jobs	J3	J2	J1
pji	2	2	2
rji	0	8	3
dji	2	10	5
Ci	2	10	12
Li	0	0	7

C1=2+0=2 C2=2+8=10 C3=2+10=12

L1=2-2=0 L2=10-10=0 L3=12-5=7

Lmax (2) =7

El mínimo **Lmax (2) es 0**, la secuencia de los trabajos es 3-1-2.

- **Máquina 3**

r_{ji} es la ruta más larga desde el nodo S hasta el nodo Oji en la Figura 31.

r31=5 r32=3 r33=7

$$dji = Cmax - L((Oji), F) + pji$$

d31=10-5+5=10 d32=10-7+5=8 d33=10-3+3=10

Para determinar el tiempo Ci se usa $$C_i = \begin{cases} p_{ji} + C_{i-1} \text{ , si } C_{i-1} > r_{ji} \\ p_{ji} + r_{ji} \text{ , si } C_{i-1} < r_{ji} \end{cases}$$

El tiempo de tardanza es $Li=Ci-dji$

Secuencia 1-2-3

Jobs	J1	J2	J3
pji	5	5	3
rji	5	3	7
dji	10	8	10
Ci	10	15	18
Li	0	7	8

C1=5+5=10 C2=5+10=15 C3=3+15=18

L1=10-10=0 L2=15-8=7 L3=18-10=8

Lmax (3) =8

Secuencia 1-3-2

Jobs	J1	J3	J2
pji	5	3	5
rji	5	7	3
dji	10	10	8
Ci	10	13	18
Li	0	3	10

C1=5+5=10 C2=3+10=13 C3=5+13=18

L1=10-10=0 L2=13-10=3 L3=18-8=10

Lmax (3) =10

Secuencia 2-1-3

Jobs	J2	J1	J3
pji	5	5	3
rji	3	5	7
dji	8	10	10
Ci	8	13	16
Li	0	3	6

C1=5+3=8 C2=5+8=13 C3=3+13=16

L1=8-8=0 L2=13-10=3 L3=16-10=6

Lmax (3) =6

El Lmax es 6 porque es el mayor valor entre los Li encontrados en esta combinación de trabajos.

Secuencia 2-3-1

Jobs	J2	J3	J1
pji	5	3	5
rji	3	7	5
dji	8	10	10
Ci	8	11	16
Li	0	1	6

C1=5+3=8 C2=3+8=11 C3=5+11=16

L1=8-8=0 L2=11-10=1 L3=16-10=6

Lmax (3) =6

Secuencia 3-1-2

Jobs	J3	J1	J2
pji	3	5	5
rji	7	5	3
dji	10	10	8
Ci	10	15	20
Li	0	5	12

C1=3+7=10 C2=5+10=15 C3=5+15=20

L1=10-10=0 L2=15-10=5 L3=20-8=12

Lmax (3) =12

Secuencia 3-2-1

Jobs	J3	J2	J1
pji	3	5	5
rji	7	3	5
dji	10	8	10
Ci	10	15	20
Li	0	7	10

C1=3+7=10 C2=5+10=15 C3=5+15=20

L1=10-10=0 L2=15-8=7 L3=20-10=10

Lmax (3) =10

El Lmax es 10 porque es el mayor valor entre los Li encontrados en esta combinación de trabajos

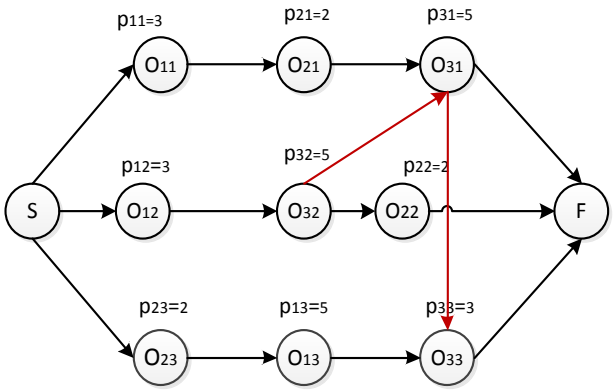
El mínimo **Lmax (3) es 6**, dos secuencias generan este valor por tanto se elige una de ellas arbitrariamente, 2-1-3 es la secuencia de los trabajos seleccionada.

La máquina 3 es el cuello de botella al tener al mayor Lmax y se incluye en el conjunto Mo. La secuencia 2-1-3 es insertada en el grafo disyuntivo de la Figura 31 generando un nuevo diagrama (Figura 32). El makespan actual es

Cmax ({3})=Cmax (∅) + Lmax (3)=10+6=16

Segunda iteración: A partir de la Figura 32 y el makespan actual se analizan las máquinas que aún no han sido programadas. Se evalúan todas las posibles secuencias de trabajos para determinar cuál es el nuevo cuello de botella repitiendo el procedimiento de la primera iteración.

Figura 32. Grafo disyuntivo con la primera máquina programada (Primera iteración).



- **Máquina 1**

r_{ji} es la ruta más larga desde el nodo S hasta el nodo Oji en la Figura 32.

r11=0 r12=0 r13=2

$$dji = Cmax - L((Oji), F) + pji$$

d11=16-13+3=6 d12=16-16+3=3 d13=16-8+5=13

Secuencia 1-2-3

Jobs	J1	J2	J3
pji	3	3	5
rji	0	0	2
dji	6	3	13
Ci	3	6	11
Li	-3	3	-2

C1=3+0=3 C2=3+3=6 C3=5+6=11

L1=3-6=-3 L2=6-3=3 L3=11-13=-2

Lmax (1) =3

Secuencia 1-3-2

Jobs	J1	J3	J2
pji	3	5	3
rji	0	2	0
dji	6	13	3
Ci	3	8	11
Li	-3	-5	8

C1=3+0=3 C2=5+3=8 C3=3+8=11

L1=3-6=-3 L2=8-13=-5 L3=11-3=8

Lmax (1) =8

Secuencia 2-1-3

Jobs	J2	J1	J3
pji	3	3	5
rji	0	0	2
dji	3	6	13
Ci	3	6	11
Li	0	0	-2

C1=3+0=3 C2=3+3=6 C3=5+6=11

L1=3-3=0 L2=6-6=0 L3=11-13=-2

Lmax (1) =0

El Lmax es 0 porque es el mayor valor entre los Li encontrados en esta secuencia.

Secuencia 2-3-1

Jobs	J2	J3	J1
pji	3	5	3
rji	0	2	0
dji	3	13	6
Ci	3	8	11
Li	0	-5	5

C1=3+0=3 C2=5+3=8 C3=3+8=11

L1=3-3=0 L2=8-13=-5 L3=11-6=5

Lmax (1) =5

Secuencia 3-1-2

Jobs	J3	J1	J2
pji	5	3	3
rji	2	2	0
dji	13	6	3
Ci	7	10	13
Li	-6	4	10

C1=5+2=7 C2=3+7=10 C3=3+10=13

L1=7-13=-6 L2=10-6=4 L3=13-3=10

Lmax (3) =10

Secuencia 3-2-1

Jobs	J3	J2	J1
pji	5	3	3
rji	2	0	0
dji	13	3	6
Ci	7	10	13
Li	-6	7	7

C1=5+2=7 C2=3+7=10 C3=3+10=13

L1=7-13=-6 L2=10-3=7 L3=13-6=7

Lmax (1) =7

El mínimo **Lmax (1) es 0**, la secuencia de los trabajos es 2-1-3.

- **Máquina 2**

r_{ji} es la ruta más larga desde el nodo S hasta el nodo $0ji$ en la Figura 32.

r21=3 r22=8 r23=0

d21=16-10+2=8 d22=16-2+2=16 d23=16-10+2=8

Secuencia 1-2-3

Jobs	J1	J2	J3
pji	2	2	2
rji	3	8	0
dji	8	16	8
Ci	5	10	12
Li	-3	-6	4

C1=2+3=5 C2=2+8=10 C3=2+10=12

L1=5-8=-3 L2=10-16=-6 L3=12-8=4

Lmax (2) =4

Secuencia 1-3-2

Jobs	J1	J3	J2
pji	2	2	2
rji	3	0	8
dji	8	8	16
Ci	5	7	10
Li	-3	-1	-6

C1=2+3=5 C2=2+5=7 C3=2+8=10

L1=5-8=-3 L2=7-8=-1 L3=10-16=-6

Lmax (2) =-1

Secuencia 2-1-3

Jobs	J2	J1	J3
pji	2	2	2
rji	8	3	0
dji	16	8	8
Ci	10	12	14
Li	-6	4	6

C1=2+8=10 C2=2+10=12 C3=2+12=14

L1=10-16=-6 L2=12-8=4 L3=14-8=6

Lmax (2) =6

Secuencia 2-3-1

Jobs	J2	J3	J1
pji	2	2	2
rji	8	0	3
dji	16	8	8
Ci	10	12	14
Li	-6	4	6

C1=2+8=10 C2=2+10=12 C3=2+12=14

L1=10-16=-6 L2=12-8=4 L3=14-8=6

Lmax (2) =6

Secuencia 3-1-2

Jobs	J3	J1	J2
pji	2	2	2
rji	0	3	8
dji	8	8	16
Ci	2	5	10
Li	-6	-3	-6

C1=2+0=2 C2=2+3=5 C3=2+8=10

L1=2-8=-6 L2=5-8=-3 L3=10-16=-6

Lmax (2) =-3

El Lmax (2) es -3 porque es el mayor valor entre los Li encontrados en esta secuencia.

Secuencia 3-2-1

Jobs	J3	J2	J1
pji	2	2	2
rji	0	8	3
dji	8	16	8
Ci	2	10	12
Li	-6	-6	4

C1=2+0=2 C2=2+8=10 C3=2+10=12

L1=2-8=-6 L2=10-16=4 L3=12-8=4

Lmax (2)=4

El mínimo **Lmax (2) es -3**, la secuencia de los trabajos es 3-1-2

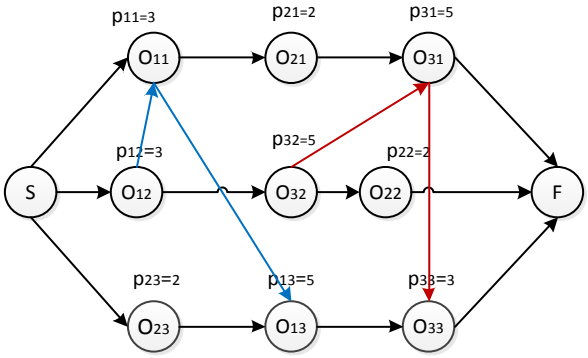
La máquina 1 es el cuello de botella al tener al mayor Lmax y se incluye en el conjunto Mo. La secuencia 2-1-3 es insertada en el grafo disyuntivo de la Figura 32 generando un nuevo diagrama (Figura 33). El makespan se mantiene.

Cmax ({3,1})=Cmax ({3}) + Lmax (1)=16-0=16

Para cada máquina ya secuenciada diferente de la última máquina programada, se eliminan los arcos disyuntivos correspondientes en el grafo y

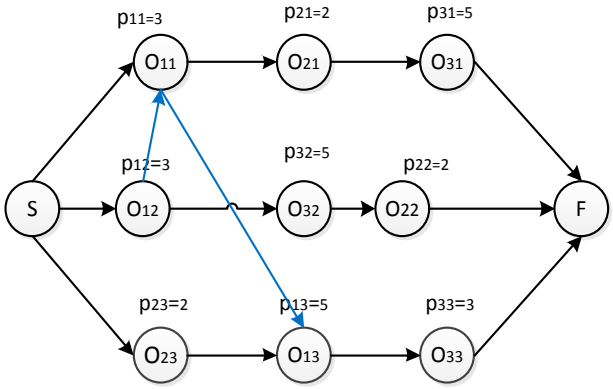
se reformula como un subproblema teniendo en cuenta la ruta más larga en el último grafo disyuntivo secuenciado.

Figura 33. Grafo disyuntivo segunda máquina programada (Segunda iteración).



En este caso se debe reprogramar la máquina 3, para ello en el grafo disyuntivo de la Figura 33 se eliminan los arcos correspondientes y se evalúan las secuencias.

Figura 34. Grafo disyuntivo sin los arcos de la máquina 3.



- **Máquina 3**

r_{ji} es la ruta más larga desde el nodo S hasta el nodo Oji en la figura 34. La ruta más larga actual en el grafo es Cmax=14

r21=3 r22=8 r23=0

d21=14-5+5=14 d22=14-7+5=12 d23=14-3+3=14

Secuencia 1-2-3

Jobs	J1	J2	J3
pji	5	5	3
rji	8	3	11
dji	14	12	14
Ci	13	18	21
Li	-1	6	7

C1=5+8=13 C2=5+13=18 C3=3+18=21

L1=13-14=-1 L2=18-12=6 L3=21-14=7

Lmax (3)=7

Secuencia 1-3-2

Jobs	J1	J3	J2
pji	5	3	5
rji	8	11	3
dji	14	14	12
Ci	13	16	21
Li	-1	2	9

C1=5+8=13 C2=3+13=16 C3=5+16=21

L1=13-14=-1 L2=16-14=2 L3=21-12=9

Lmax (3) = 9

Secuencia 2-1-3

Jobs	J2	J1	J3
pji	5	5	3
rji	3	8	11
dji	12	14	14
Ci	8	13	16
Li	-4	-1	2

C1=5+3=8 C2=5+8=13 C3=3+13=16

L1=8-12=-4 L2=13-14=-1 L3=16-14=2

Lmax (3)=2

El Lmax es 2 porque es el mayor valor entre los Li encontrados en esta secuencia.

Secuencia 2-3-1

Jobs	J2	J3	J1
pji	5	3	5
rji	3	11	8
dji	12	14	14
Ci	8	14	19
Li	-4	0	5

C1=5+3=8 C2=3+11=14 C3=5+14=19

L1=8-12=-4 L2=14-14=0 L3=19-14=5

Lmax (3) =5

Secuencia 3-1-2

Jobs	J3	J1	J2
pji	3	5	5
rji	11	8	3
dji	14	14	12
Ci	14	19	24
Li	0	5	12

C1=3+11=14 C2=14+5=19 C3=5+19=24

L1=14-14=0 L2=19-14=5 L3=24-12=12

Lmax (3)=12

Secuencia 3-2-1

Jobs	J3	J2	J1
pji	3	5	5
rji	11	3	8
dji	14	12	14
Ci	14	19	24
Li	0	7	10

C1=3+11=14 C2=14+5=19 C3=5+19=24

L1=14-14=0 L2=19-12=7 L3=24-14=10

Lmax (3)=10

El menor **Lmax (3) es 2**, la secuencia es 2-1-3, esta programación es la misma obtenida en la primera iteración por tanto el makespan se mantiene en 16, se dibujan de nuevo los arcos de la máquina 3 en el grafo disyuntivo.

Tercera iteración: Se han programado las máquinas 3 y 1; para determinar el orden en que se procesaran los trabajos en la máquina 2 se evalúan todas las secuencias a partir de la Figura 33 y Cmax=16.

- **Máquina 2**

r_{ji} es la ruta más larga desde el nodo S hasta el nodo $0ji$ en la figura 33. La ruta más larga actual en el grafo es Cmax=16

r21=6 r22=8 r23=0

d21=16-10+2=8 d22=16-2+2=16 d23=16-10+2=16

Secuencia 1-2-3

Jobs	J1	J2	J3
pji	2	2	2
rji	6	8	0
dji	8	16	8
Ci	8	10	12
Li	0	-6	4

C1=2+6=8 C2=2+8=10 C3=2+10=12

L1=8-8=0 L2=10-16=-6 L3=12-8=4

Lmax (2) =4

Secuencia 1-3-2

Jobs	J1	J3	J1
pji	2	2	2
rji	6	0	8
dji	8	8	16
Ci	8	10	12
Li	0	2	-4

C1=2+6=8 C2=2+8=10 C3=2+10=12

L1=8-8=0 L2=10-8=2 L3=12-16=-4

Lmax (2) =2

Secuencia 2-1-3

Jobs	J2	J1	J3
pji	2	2	2
rji	8	6	0
dji	16	8	8
Ci	10	12	14
Li	-6	4	6

C1=2+8=10 C2=2+10=12 C3=2+12=14

L1=10-16=-6 L2=12-8=4 L3=14-8=6

Lmax (2) =6

Secuencia 2-3-1

Jobs	J2	J3	J1
pji	2	2	2
rji	8	0	6
dji	16	8	8
Ci	10	12	14
Li	-6	4	6

C1=2+8=10 C2=2+10=12 C3=2+12=14

L1=10-16=-6 L2=12-8=4 L3=14-8=6

Lmax (2)=6

Secuencia 3-1-2

Jobs	J3	J1	J2
pji	2	2	2
rji	0	6	8
dji	8	8	16
Ci	2	8	10
Li	-6	0	-6

C1=2+0=2 C2=2+6=8 C3=2+8=10

L1=2-8=-6 L2=8-8=4 L3=10-16=-6

Lmax (2)=0

Secuencia 3-2-1

Jobs	J3	J2	J1
pji	2	2	2
rji	0	8	6
dji	8	16	8
Ci	2	10	12
Li	-6	-6	4

C1=2+0=2 C2=2+8=10 C3=2+10=12

L1=2-8=-6 L2=10-16=-6 L3=12-8=4

Lmax (2)=4

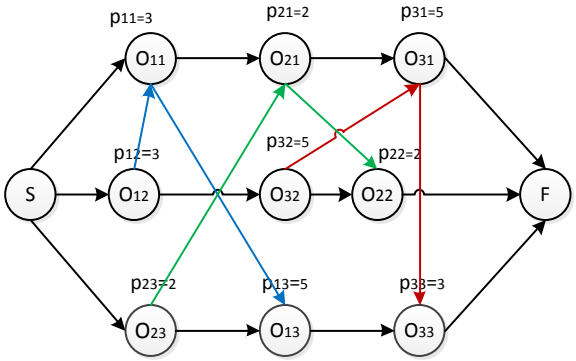
El mínimo **Lmax (2) es 0**, la secuencia de los trabajos es 3-1-2

Los arcos de la secuencia de trabajos en la máquina 2 se insertan en la Figura 33 y se completa la programación de la maquinas (Figura 35)

El makespan actual es

Cmax ({3, 1,2})=Cmax ({3,1}) + Lmax (2)=16-0=16

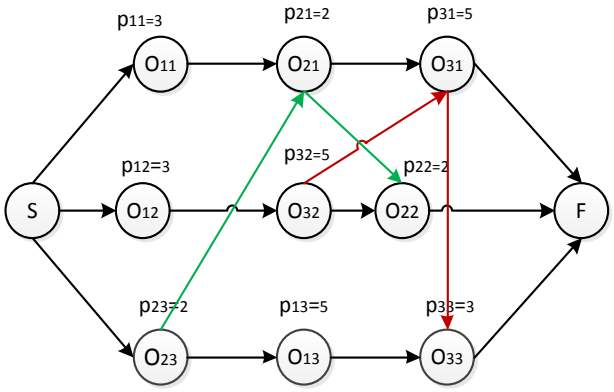
Figura 35. Grafo disyuntivo con todas las máquinas programadas para el ejemplo 3 máquinas y 3 trabajos (tercera iteración)



Para cada máquina ya secuenciada exceptuando la última máquina programada, se eliminan en el grafo los arcos disyuntivos correspondientes y se reformula como un subproblema.

En este caso se debe reprogramar la máquina 3 y la máquina 1.

Figura 36. Grafo disyuntivo sin los arcos de la maquina 1



- **Máquina 1**

r_{ji} es la ruta más larga desde el nodo S hasta el nodo Oji en la Figura 36. La ruta más larga en el grafo al eliminar los arcos de la máquina 1 es Cmax=16

r11=0 r12=0 r13=2

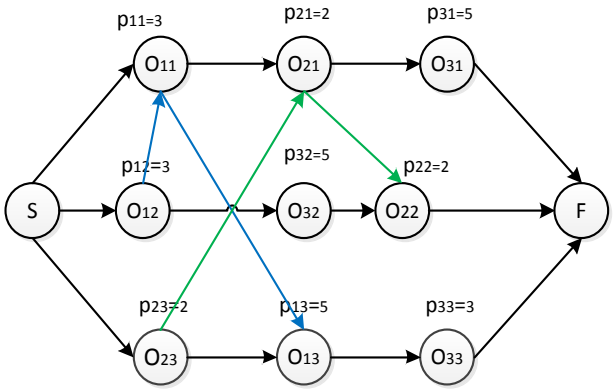
d11=16-13+3=6 d12=16-16+3=3 d13=16-8+5=13

Se observa que los r_{ji} y dji son iguales a los hallados en la segunda iteración cuando se evaluaron las combinaciones de trabajos por tanto el **Lmax (1) es 0** y la secuencia de los trabajos es 2-1-3, al ser el mismo orden no se realiza ningún cambio en el grafo y makespan se mantienen en 16

- **Máquina 3**

r_{ji} es la ruta más larga desde el nodo S hasta el nodo Oji en la Figura 37. La ruta más larga en el grafo al eliminar los arcos de la máquina 3 es Cmax=14

Figura 37. Grafo disyuntivo sin los arcos de la máquina 3



r31=5 r32=3 r33=11

d31=14-5+5=14 d32=14-7+5=12 d23=14-3+3=16

Se observa que los r_{ji} y dji son los mismo hallados en la segunda iteración cuando se reprogramó esta máquina, por tanto el **Lmax (3) es 2** y la secuencia de los trabajos es 2-1-3, al ser el mismo orden no se realiza ningún cambio en el grafo.

Al ser programadas todas las máquinas la heurística finaliza y se encuentra un Schedule factible para el problema de 3 máquinas y 3 trabajos.

La asignación de trabajos a las máquinas es:

Máquina 1: 2-1-3 **Máquina 2**: 3-1-2 **Máquina 3:** 2-1-3

El makespan final es **Cmax=16**

La programación final de todas las máquinas es la misma de la Figura 35.

ANEXO C. TIPOS DE REPRESENTACIÓN PARA EL PROBLEMA DEL JOB SHOP SCHEDULING

En la literatura se encuentran distintos tipos o formas de representación para el problema del Job Shop Scheduling (JSP). En esta sección se discuten diferentes representaciones y la selección de una de ellas de acuerdo a los objetivos de la presente investigación.

REPRESENTACIÓN BASADA EN EL GRAFO DISYUNTIVO

El problema de Job Shop Scheduling puede representarse mediante un grafo disyuntivo D=(N, A, E) donde,

N, es el conjunto de nodos que representan las operaciones.

A, es el conjunto de arcos conjuntivos (arcos dirigidos) que representan el orden de precedencia de las operaciones.

E, es el conjunto de arcos disyuntivos (arcos no dirigidos) que representa la secuencia de procesamiento de las operaciones en cada máquina.

La construcción de un programa se establece con la orientación de todos los arcos disyuntivos. Cuando se ha determinado la orientación de los arcos, estos se convierten en conjuntivos o dirigidos y se establece la secuencia de procesamiento en cada máquina.

La representación basada en el grafo disyuntivo, al igual que otras representaciones para el Algoritmo Genético, consta de un cromosoma que consiste en una cadena binaria correspondiente a una lista ordenada de los arcos disyuntivos en E. Determinar el grafo disyuntivo del problema es el primer paso para iniciar esta representación.

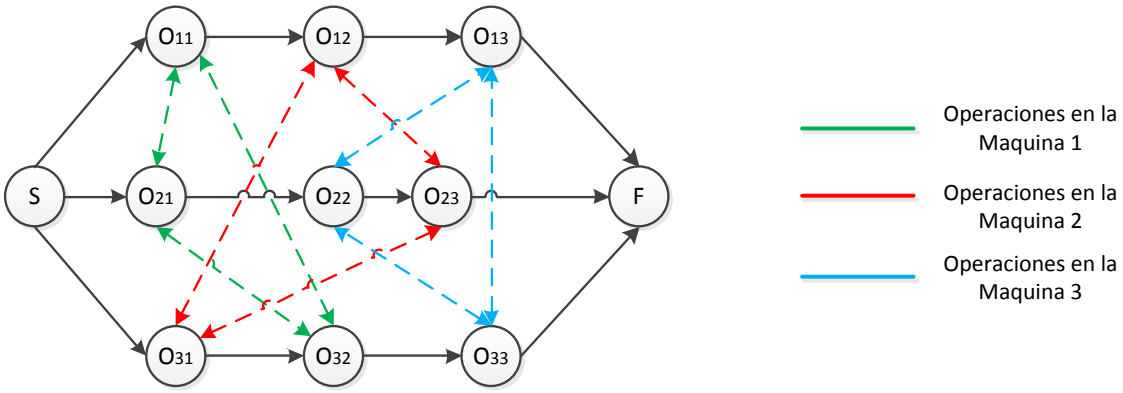
El siguiente ejemplo permite esclarecer la representación discutida. Se considera un problema de 3 máquinas y 3 trabajos como aparece en la Tabla 35. Inicialmente en el grafo existen arcos a los que no se les ha definido la dirección, estos son los arcos disyuntivos (Figura 38).Para codificar el

cromosoma a partir de esta información se construye el grafo disyuntivo del problema definiendo la dirección de los arcos disyuntivos y determinando la secuencia de procesamiento de los trabajos en cada máquina, los nodos del grafo corresponde a o_{ij}, la operación j del trabajo i.

Tabla 35. Ejemplo para JSP de 3 trabajos y 3 máquinas.

Trabajos	Secuencia de máquinas	Tiempo de Procesamiento
J_1	m_1 - m_2 - m_3	3 -2- 5
J_2	m_1 - m_3 - m_2	3 -5 - 2
J_3	m_2 - m_1 - m_3	2 -5 - 3

Figura 38 Grafo disyuntivo para el ejemplo de la Tabla 35.



En la Figura 38, las operaciones que deben procesarse en la misma máquina están unidas por arcos disyuntivos del mismo color. Con este grafo se explicará la representación basada en el grafo disyuntivo para el GA

Una lista ordenada de los arcos disyuntivos se realiza con el fin de establecer el cromosoma, la lista se basa en la notación

e_{ij} : representa la dirección del arco disyuntivo entre los nodos i y j.

La lista se ordena teniendo en cuenta los arcos disyuntivos de cada máquina es decir primero se codifica el grupo e_{ij} para la máquina1, luego para la máquina 2 y por último para la máquina 3.

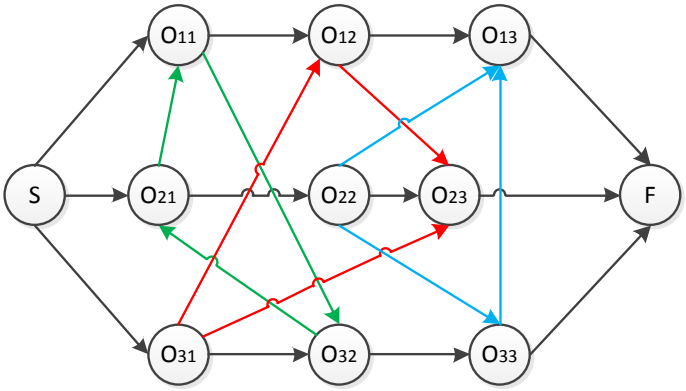
La lista ordenada de arcos disyuntivos del ejemplo es:

$$\overbrace{e_{21}e_{31}e_{32}}^{\text{Maq 1}}\ \overbrace{e_{13}e_{12}e_{23}}^{\text{Maq 2}}\ \overbrace{e_{12}e_{13}e_{32}}^{\text{Maq 3}}$$

El valor que toma i y j se basa en la relación que muestran los arcos entre los trabajos que pasan por la misma máquina. Por ejemplo para la máquina 1, e_{21} es el arco disyuntivo entre el trabajo 2 y el trabajo 1 que deben procesarse en esa máquina; para la máquina 2, e_{12} es el arco disyuntivo entre el trabajo 1 y el trabajo 2.

Para establecer el cromosoma, se define la orientación de los arcos disyuntivos como se muestra en la Figura 39, esta orientación es arbitraria.

Figura 39. Grafo disyuntivo con todos los arcos dirigidos.



Una vez todos los arcos han sido dirigidos, el cromosoma se representa teniendo en cuenta la lista ordenada de arcos y la siguiente notación:

$$e_{ij}:\begin{cases}1, \text{ si la orientacion del arco disyuntivo es desde nodo j al nodo i}\\ 0, \text{ si la orientacion del arco disyuntivo es desde el nodo i al nodo j}\end{cases}$$

Para escribir el cromosoma se busca en el grafo disyuntivo los arcos ya dirigidos para cada máquina (Figura 39) y teniendo en cuenta esa orientación se define el valor que toma el gen en el cromosoma. Si en la máquina 1 el arco disyuntivos se orientó desde el nodo del trabajo 2 al nodo del trabajo 1, el valor

del gen es 0 porque el arco disyuntivo va desde el nodo i a j según e_{21} donde i= {2} y j= {1}.

Lista de los arcos disyuntivos: $e_{21}e_{31}e_{32}e_{13}e_{12}e_{23}e_{12}e_{13}e_{32}$

Cromosoma: $[0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1]$

En el cromosoma el primer gen es 0, significa que el arco disyuntivo está dirigido desde el trabajo 2 al trabajo 1, es decir desde el nodo O_{21} al nodo O_{11} en la máquina 1; el cuarto gen es 1, significa que el arco está orientado desde el trabajo 3 al trabajo 1, es decir desde el nodo O_{31} al nodo O_{12} en la máquina 2; así cada una de los genes del cromosoma es determinado.

Este cromosoma no se usa para representar una programación, su objetivo es utilizarlo como una decisión de preferencia, establece la orientación de los arcos disyuntivos para cada máquina.

A partir del cromosoma y la lista de arcos disyuntivos es posible realizar el proceso inverso y establecer el grafo disyuntivo.

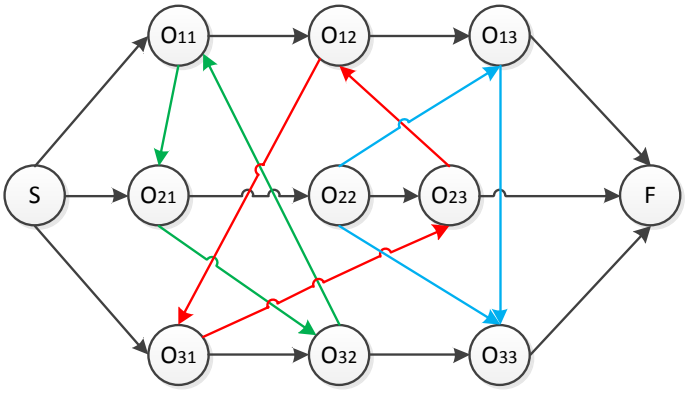
Se tiene la misma lista de arcos disyuntivos

$e_{21}e_{31}e_{32}e_{13}e_{12}e_{23}e_{12}e_{13}e_{32}$

Y el cromosoma [1 0 1 0 1 1 1 0 1]

Teniendo en cuenta los valores que puede tomar e_{ij} , se interpreta cada gen del cromosoma y se establece la orientación de los arcos disyuntivos, se convierten en dirigidos y se obtiene el grafo. Por ejemplo el arco disyuntivo e_{21} está dirigido desde el trabajo 1 al trabajo 2 en la máquina 1, porque el valor del gen es 1 para este arco en el cromosoma. Al revisar cada gen se obtiene la dirección de los arcos y el grafo es construido como se ve en la Figura 40.

Figura 40. Grafo disyuntivo decodificado.

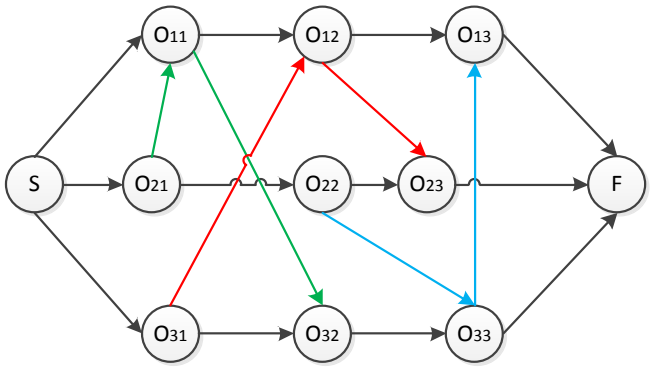


REPRESENTACIÓN BASADA EN OPERACIONES

Para el problema de Job Shop Scheduling para n trabajos y m máquinas la representación del cromosoma se hace mediante un vector que contiene nxm posiciones, cada posición es un gen que corresponde a un entero que toma valores desde 1 hasta n y representa las operaciones de un trabajo, todas las operaciones de un mismo trabajo se representan con el mismo valor para el gen y se interpretan de acuerdo al orden de aparición en el cromosoma, determinando la secuencia de procesamiento. Cada trabajo aparece en el cromosoma m veces y cada vez que se repite un gen dentro del cromosoma indica una operación de ese trabajo. Este tipo de representación siempre genera programas factibles ya que no viola las restricciones de precedencia al depender de la lectura y programación de las operaciones.

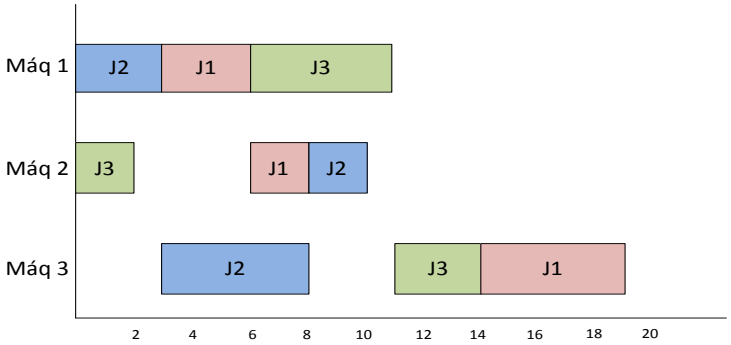
Mediante el ejemplo (Tabla 35) se explica la representación discutida usando el grafo disyuntivo de la Figura 41.

Figura 41 Grafo disyuntivo con secuencias definidas para la Tabla 35.



Teniendo en cuenta las restricciones de precedencia, la secuencia de procesamiento para cada máquina establecida en el grafo disyuntivo y los tiempos de procesamiento de la Tabla 35, se grafica el Diagrama de Gantt, el tiempo de finalización de la última operación programada en el diagrama de Gantt determina el valor del Makespan.

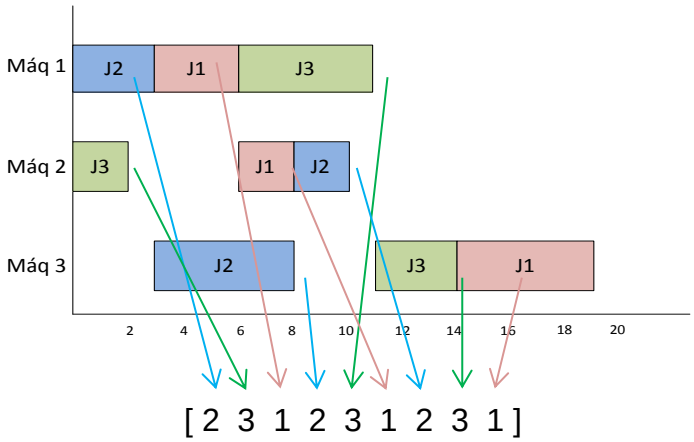
Figura 42 Diagrama de Gantt para la Tabla 35.



Para codificar el cromosoma se leen las operaciones desde la izquierda según la secuencia de procesamiento en el diagrama de Gantt como se muestra en la Figura 43, teniendo en cuenta que los genes toman los valores 1, 2, 3 que corresponden a los trabajos (J1, J2, J3) cada uno se repite tres veces porque es número de operaciones que tiene cada trabajo (también se puede relacionar con el número de máquinas ya que una máquina solo puede realizar una operación).

Cromosoma: [2 3 1 2 3 1 2 3 1]

Figura 43 Cromosoma a partir del Diagrama de Gantt



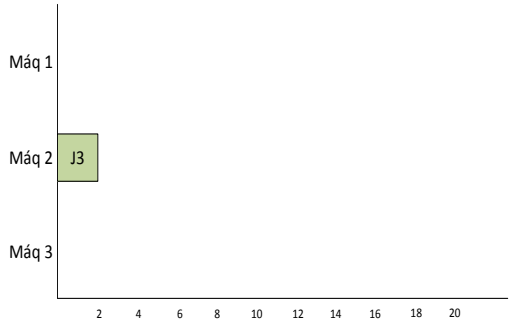
Un diagrama de Gantt puede generar diferentes cromosomas a partir de la lectura de las operaciones, pero todos ellos tendrán el mismo Makespan (Tiempo total de procesamiento). En el cromosoma el primer gen es el 2 que representa la primera operación del trabajo 2 y debe ser la primera en procesarse en la máquina 1, el segundo gen es el 3, indica la primera operación del trabajo 3 y es la primera en pasar por la máquina 2, por tanto el cromosoma se puede interpretar como una lista de operaciones $[O_{21}\ O_{31}\ O_{11}\ O_{22}\ O_{32}\ O_{12}\ O_{23}\ O_{33}\ O_{13}]$ donde O_{ij} es la operación j del trabajo i.

La decodificación a un diagrama de Gantt se realiza por medio del cromosoma y la información del problema (tiempos de procesamiento y secuencia de las maquinas). El cromosoma se lee de izquierda a derecha y se traducen los genes en operaciones que son programadas en el diagrama teniendo en cuenta los tiempos de procesamiento y la máquina correspondiente, cada operación se asigna en el menor tiempo posible.

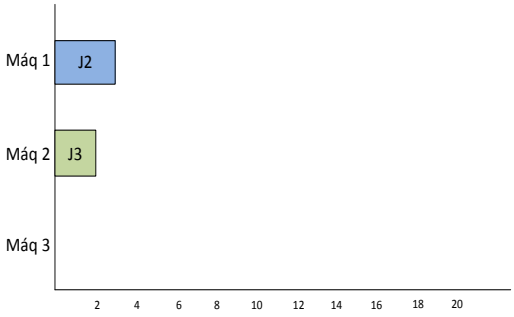
Para el ejemplo de la Tabla 35, se considera el siguiente cromosoma [3 2 1 2 3 1 2 1 3], el primer gen es 3, se debe programa la primera operación del trabajo 3, el diagrama inicia con esta operación que debe realizarse en la máquina 2 con tiempo de procesamiento de 2 unidades (Figura 44a); el siguiente gen es 2 y corresponde a la primera operación del trabajo 2 que se

procesa en la máquina 1 y toma 3 unidades de tiempo (Figura 44b), el tercer gen es 1, significa que se programa la primera operación del trabajo 1 en la máquina 1 (Figura 44c), la siguiente operación en programarse corresponde al trabajo 2 y se realiza en la máquina 3. (Figura 44d), de esta forma se traducen los genes en el diagrama de Gantt hasta que todas las operaciones han sido programadas como se muestra en la Figura 44e, el tiempo en que finaliza la última operación del diagrama determina el makespan del cromosoma en este caso ese valor es de 16 unidades de tiempo.

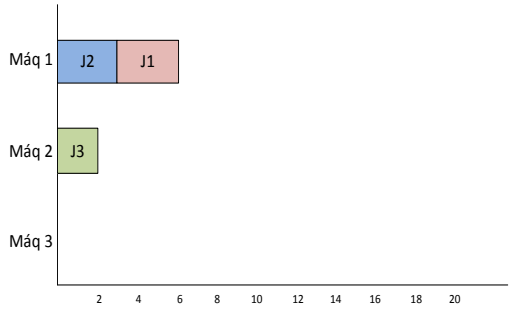
Figura 44 Construcción del diagrama de Gantt a partir de cromosoma: (a) asignación del primer gen; (b) asignación del segundo gen; (c) asignación del tercer gen; (d) Asignación del cuarto gen, (e) Diagrama de Gantt terminado



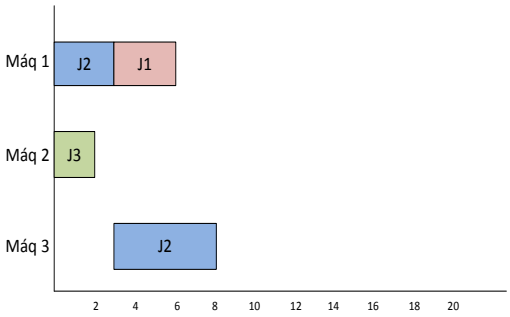
(a)



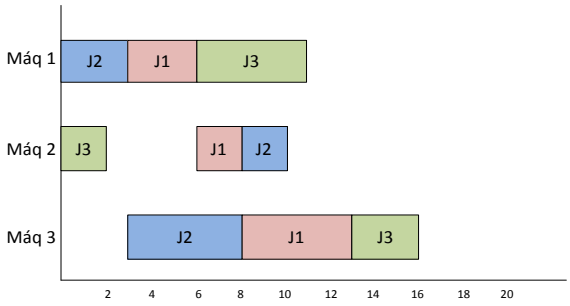
(b)



(c)



(d)



(e)

REPRESENTACIÓN BASADA EN LOS TIEMPOS DE TERMINACIÓN

La representación se basa en una lista ordenada de los tiempos de terminación de las operaciones. En ella el cromosoma no es una cadena binaria, se denota como:

C_{jir} , el tiempo de terminación de la operación i del trabajo j en la máquina r.

Siguiendo con el ejemplo (Tabla 35), se explica esta representación. Para construir el cromosoma es necesario codificar cada operación de cada trabajo con la notación dada, como se muestra en la Tabla 36. Por ejemplo C_{223} indica el tiempo de terminación de la operación 2 del trabajo 2 en la máquina 3.

Tabla 36. Codificación del ejemplo 3 máquinas 3 trabajos para la representación basada en tiempos de terminación.

Trabajo	Operaciones 1-2-3 Secuencia de Maq.	Tiempos de Oper.	Codificación
J1	1-2-3	3-2-5	C_{111} C_{122} C_{133}
J2	1-3-2	3-5-2	C_{211} C_{223} C_{232}
J3	2-1-3	2-5-3	C_{312} C_{321} C_{333}

Una vez que se han codificado todas las operaciones, el cromosoma se construye organizando los C_{jir} por trabajo, es decir, en el cromosoma los primeros tres genes $(C_{111}\ C_{122}\ C_{133})$ pertenecen al trabajo J1, los siguientes tres genes $(C_{211}\ C_{223}\ C_{232})$ al trabajo J2 y los últimos tres genes $(C_{312}\ C_{321}\ C_{333})$ al trabajo 3.

Cromosoma $\left[C_{111}C_{122}C_{133}C_{211}C_{223}C_{232}C_{312}C_{321}C_{331}\right]$

Esta representación no es apropiada para la mayoría de los operadores genéticos y puede generar programas no factibles, sus autores propusieron un operador de cruce especial para resolver la inconsistencia entre el cromosoma y los operadores.

REPRESENTACIÓN BASADA EN CLAVES ALEATORIAS

En esta representación se usan números aleatorios como claves ordenadas para decodificar la solución.

Para programar n trabajos en m máquinas, cada gen del cromosoma es un trabajo que debe ser programado y consta de 2 partes, una parte entera que pertenece al conjunto {1, 2, 3,…, m} que representa la asignación de la máquina a ese trabajo y una parte decimal generada de forma aleatoria entre 0 y 1.

Para codificar el cromosoma, se requiere generar un número aleatorio para cada una de las operaciones de cada trabajo. La Tabla 37 muestra los números aleatorios generados para el ejemplo de 3 máquinas y 3 trabajos que se presentó al inicio. La parte entera del cromosoma resulta de la máquina que está asignada para cada trabajo en el problema y va desde 1 hasta 3, con estos dos valores la clave queda completa para los trabajos de cada máquina como se muestra en la Tabla 38.

Tabla 37. Números aleatorios generados para las operaciones de cada trabajo.

Trabajo	Operaciones	Máquina	N° Aleatorio
1	1	1	0,67
1	2	2	0,93
1	3	3	0,76
2	1	1	0,30
2	2	3	0,11
2	3	2	0,58
3	1	2	0,56
3	2	1	0,39
3	3	3	0,96

Como la operación 1 del trabajo 1 se realiza en la máquina 1, la clave para este trabajo será 1,67 donde la parte entera (1) es la máquina y la parte decimal el número aleatorio generado para esta operación, la operación 1 del trabajo 3 se procesa en la máquina 2 por tanto la clave será 2,56 donde la parte entera (2) representa la máquina y la parte decimal el número aleatorio generado, de esta forma es configurado cada uno de los trabajos.

Tabla 38. Clave aleatoria para los trabajos de cada máquina.

Máquina	Trabajo		
	1	2	3
Máquina1	1,67	1,30	1,39
Máquina2	2,93	2,58	2,56
Máquina3	3,76	3,11	3,96

En el cromosoma los genes se ordenan de acuerdo a la máquina, primero las claves aleatorias que corresponde a la máquina 1, luego las claves de la máquina 2 y por ultimo las claves de la máquina 3.

Cromosoma 1 $\overbrace{[1{,}67\ \ 1{,}30\ \ 1{,}39}^{\text{maq 1}}\ \overbrace{2{,}93\ \ 2{,}58\ \ 2{,}56}^{\text{maq 2}}\ \overbrace{3{,}76\ \ 3{,}11\ \ 3{,}96]}^{\text{maq3}}$

Con el cromosoma anterior se construye la representación basada en claves aleatorias, para cada máquina las claves se ordena de forma ascendente de

acuerdo a la parte decimal de estas, este orden indicará la secuencia en que los trabajos deben pasar por cada máquina.

Tabla 39. Orden Ascendente de las claves.

	Orden Ascendente		
Máquina 1	1,30	1,39	1,67
Máquina 2	2,56	2,58	2,93
Máquina 3	3,11	3,76	3,96

El cromosoma final del problema se construye usando la notación O_{jm} el trabajo j en la máquina m. En el nuevo cromosoma los genes se establecen por máquina teniendo en cuenta el orden de la Tabla 39, es decir que el primer gen que debe aparecer es O_{21} ya que es el menor valor en la máquina 1 e indica que el trabajo 2 es el primero en procesarse en esta máquina, así cada gen es codificado hasta completar todos los trabajos.

Cromosoma final $\left[O_{21}O_{31}O_{21}O_{32}O_{22}O_{12}O_{23}O_{13}O_{33}\right]$

La representación puede violar las restricciones de precedencia, por consiguiente la codificación se acompaña con un pseudocódigo que maneja estas restricciones.

REPRESENTACIÓN BASADA EN TRABAJOS

Se tiene un problema que contiene n trabajos y m máquinas. La representación consiste en programar esta cantidad de trabajos en un diagrama de Gantt, de acuerdo a la secuencia en la que aparecen estos trabajos en el cromosoma.

La cadena del cromosoma, contendrá tantos genes como trabajos existan en el problema, y cada trabajo será representado con un número entero, es decir, si se tiene 3 trabajos a realizar, el largo de la cadena contiene 3 genes o números enteros. Para una secuencia de trabajos, todas las operaciones del primer

trabajo que aparece de izquierda a derecha en el cromosoma son las primeras en ser programadas en el diagrama, después se consideran las operaciones del segundo trabajo que aparece en la cadena del cromosoma para su programación y así sucesivamente.

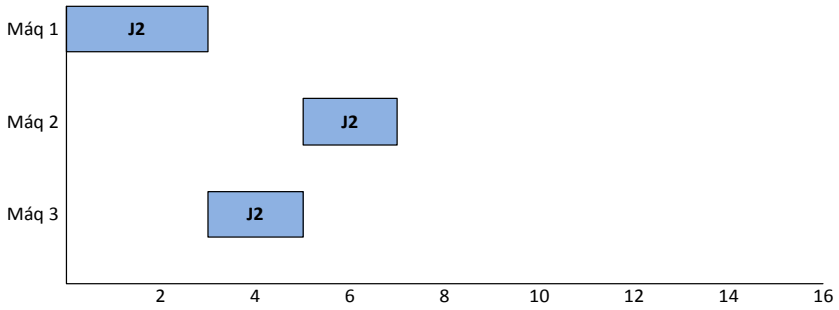
La 1ª operación del trabajo elegido para ser programado es asignada en el menor tiempo de procesamiento disponible en la máquina correspondiente a la operación en la que se requiere ser procesada, después la segunda operación y así sucesivamente hasta que se han programado todas las operaciones del trabajo.

Se repite el proceso con cada uno de los trabajos representados en el cromosoma en la secuencia apropiada.

Para el ejemplo de la Tabla 35, se tiene el siguiente cromosoma: $[2 \quad 3 \quad 1]$

Se programan las operaciones pertenecientes al trabajo número 2, de acuerdo a las restricciones de precedencia presentada en la información del ejemplo sería $(m_1\ m_3\ m_2)$ y los tiempos de procesamiento correspondientes para las operaciones en cada máquina (1 5 3) respectivamente, como lo muestra la Figura 45.

Figura 45. Programación de las operaciones del trabajo 2



Posteriormente son programadas las operaciones del segundo trabajo de izquierda a derecha del cromosoma, para este caso el siguiente trabajo a ser programado en el diagrama es el 3. Se programa de la misma manera como fue programado el trabajo anterior y cada operación es programada en el mejor

tiempo disponible en cada máquina tomando en cuenta las restricciones de precedencia de cada trabajo (Figura 46). Se repite el mismo procedimiento para el último trabajo que corresponde al 1(Figura 47).

Figura 46 Programación de las operaciones del trabajo 3

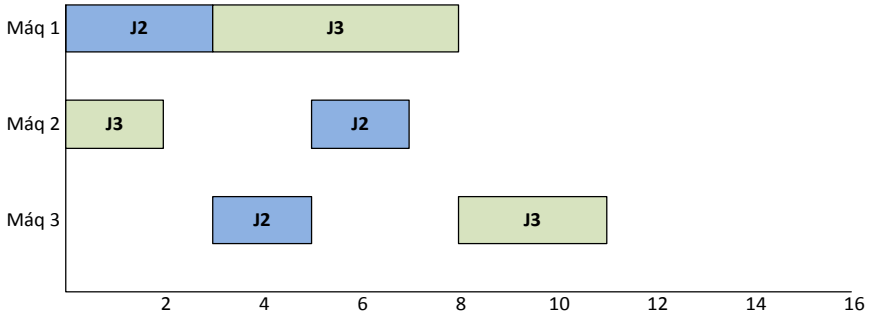
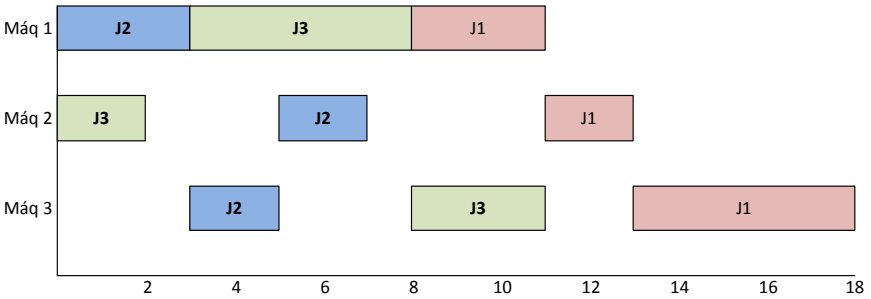


Figura 47 Programación de las operaciones del trabajo 1



El tiempo total de procesamiento de todos los trabajos con sus respectivas operaciones (makespan) es de 18 unidades de tiempo.

REPRESENTACIÓN BASADA EN LISTAS DE PREFERENCIAS

Para un problema de Job Shop Scheduling que contiene n trabajos y m máquinas, un cromosoma es formado por m subcromosomas, es decir, uno por cada máquina.

Cada subcromosoma es una cadena de números enteros con longitud n, también denominado gen y cada entero identifica una operación de un trabajo que debe ser procesado en la máquina correspondiente, por ejemplo, el

número 2 representa una operación del trabajo 2 en la máquina que corresponde la lista de preferencia.

Los subcromosomas no describen la secuencia de operación en la máquina, sino que ellos son *listas de preferencia* y cada máquina tiene su propia lista de preferencia, es decir, la operación que aparece primero en la lista de preferencia será elegida para ser programada.

A partir de un cromosoma dado, se procede a deducir el Schedule actual. Considerando el ejemplo de la Tabla 35 se tiene el siguiente cromosoma: $\left[\left(2 \quad 3 \quad 1\right)\left(1 \quad 3 \quad 2\right)\left(2 \quad 1 \quad 3\right)\right]$.

Cada subcromosoma representa la lista de preferencia para cada una de las máquinas. El primer gen (2 3 1) del cromosoma es la lista de preferencia para la máquina 1, (1 3 2) para la máquina 2 y (2 1 3) para la máquina 3.

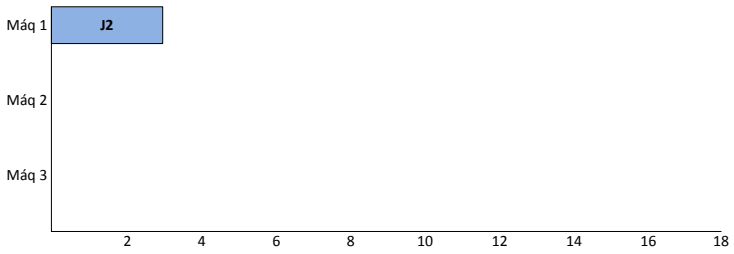
$$\left[\left(\underbrace{2 \quad 3 \quad 1}_{\substack{\textit{Lista de preferencia} \\ \textit{máq 1}}}\right)\left(\underbrace{1 \quad 3 \quad 2}_{\substack{\textit{Lista de preferencia} \\ \textit{máq 2}}}\right)\left(\underbrace{2 \quad 1 \quad 3}_{\substack{\textit{Lista de preferencia} \\ \textit{máq 3}}}\right)\right]$$

Para iniciar la representación se puede deducir que las operaciones de preferencia son el trabajo 2 en la máquina 1, trabajo 1 en la máquina 2 y el trabajo 2 en la máquina 3.

$$\left[\left(\underbrace{2 \quad 3 \quad 1}_{\substack{\textit{Lista de preferencia} \\ \textit{máq 1}}}\right)\left(\underbrace{1 \quad 3 \quad 2}_{\substack{\textit{Lista de preferencia} \\ \textit{máq 2}}}\right)\left(\underbrace{2 \quad 1 \quad 3}_{\substack{\textit{Lista de preferencia} \\ \textit{máq 3}}}\right)\right]$$

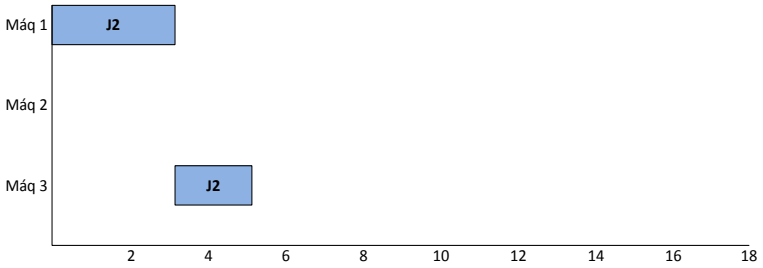
Tomando en cuenta las restricciones de precedencia únicamente el trabajo J_2 es programable en la máquina 1, así como lo muestra la Figura 48

Figura 48. Programación de la primera operación del trabajo 2 en la máquina 1

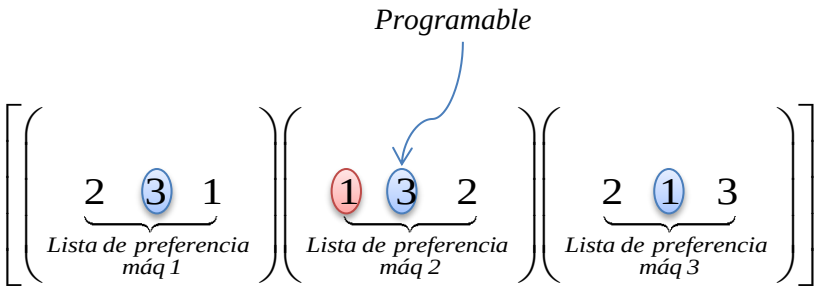


Una vez programada la primera operación, se procede a programar la siguiente de acuerdo a la lista de preferencia, en este caso el trabajo 2 puede ser programado en la máquina 3, recordando que debe esperar a que finalice la primera operación en la máquina respectiva para que pueda iniciar la segunda operación correspondiente (Figura 49).

Figura 49. Programación del trabajo 2 en máquina 2.

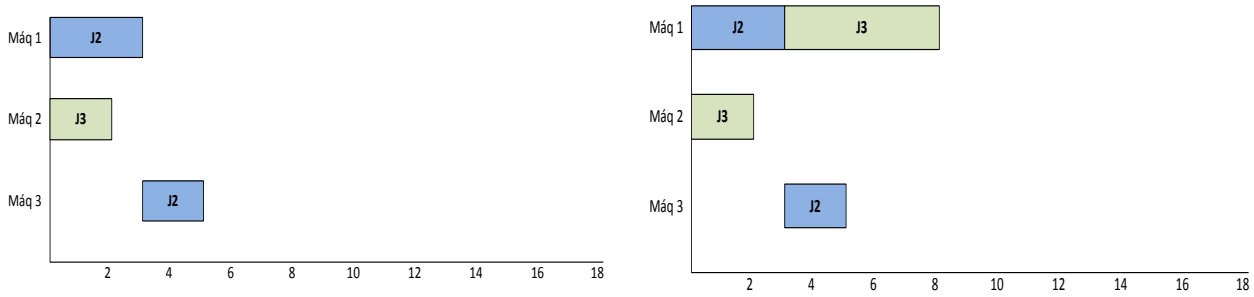


Los próximos trabajos de la lista de preferencia son el trabajo 3 en la máquina 1, trabajo 3 en la máquina 2 y el trabajo 1 en la máquina 3. De los anteriores trabajos, únicamente es posible programar el trabajo 3 en la máquina 2. El trabajo 1 de la máquina 2 se programa una vez las restricciones de precedencia lo permita.



El trabajo 3 es programado en el menor tiempo de procesamiento disponible en la máquina 2, como lo muestra la Figura 50a. Debido que ya fue programada la operación 1 de este trabajo, se procede a ejecutar la operación 2 del trabajo 3 que corresponde a la máquina 1(Figura 50b).

Figura 50. Programación del trabajo 3 (J_3) en la máquina 1 (a) y máquina 2 (b).



(a) (b)

Aún no es posible programar el trabajo 1 en la máquina 3, debido a las restricciones de precedencia de las operaciones. Por lo tanto, se salta al tercer gen de cada subcromosoma que corresponde a las últimas operaciones a ser programadas de acuerdo a la lista de preferencia de cada máquina, se debe tener presente aquellas operaciones que aún no han sido programadas, pero que tienen preferencias en cada una de las máquinas. Las últimas operaciones en las listas de preferencia son: el trabajo 1 en la máquina 1, el trabajo 2 en la máquina 2 y el trabajo 3 en la máquina 3. Todas las operaciones son programables, sin embargo una vez programado el trabajo 1 en la máquina 1(Figura 51a), se da prioridad a las operaciones que se encuentran pendientes por realizar en las máquinas respectivas (Figura 51b) y posteriormente se programan las operaciones que siguen en la lista de preferencia, tomando en cuenta que deben ser ejecutadas en el menor tiempo disponible de cada máquina sin incumplir las restricciones de precedencia (Figura 51c y 51d).

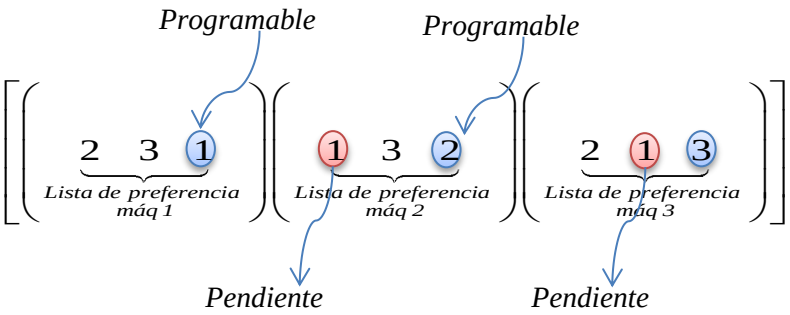
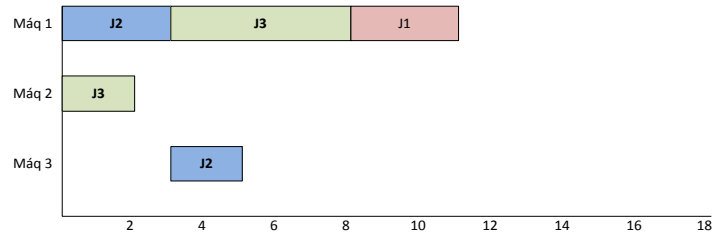
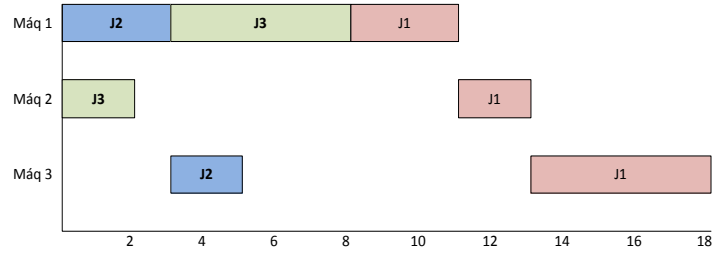


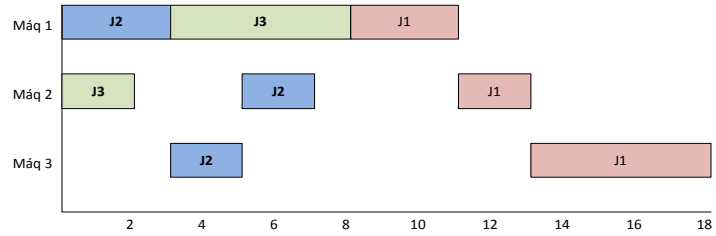
Figura 51. Programación de las operaciones de la lista de preferencia. (a) Operación 1 de J1, (b) operación 2 de J2, (c) operación 3 de J3, (d) operación 3 de J2 y operación 3 de J3.



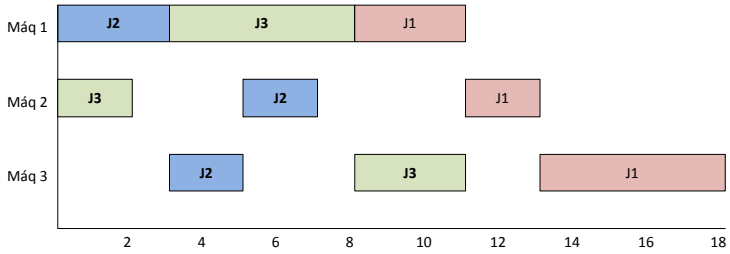
(a)



(b)



(c)



(d)

Una vez programadas todas las operaciones, se obtiene una solución factible, como todas las que puede arrojar las representaciones basadas en listas de

preferencias. En el ejemplo se obtuvo un tiempo total de procesamiento de los trabajos (makespan) de 18 unidades de tiempo.

ANEXO D. CÓDIGOS DE PROGRAMACIÓN ALGORITMO MEMÉTICO

El presente anexo contiene los códigos de programación del algoritmo memético propuesto con las funciones utilizadas para su desarrollo en el lenguaje de programación Matlab® para resolver el problema del Job Shop Scheduling.

Función Población

Crea la matriz de individuos que representa la población inicial de manera aleatoria, el número de columnas de la población depende del tamaño de la instancia (nxm).

Entradas:

- *trabajo*: Cantidad de trabajos de la instancia del benchmark.
- *maquina*: Cantidad de máquinas de la instancia del benchmark.
- *TamPob*: Tamaño de la población inicial.

Salida:

- *población*: Matriz de población inicial

```
function poblacion=nuevapoblacion(trabajos,maquinas,tamPob)

tamIndv=trabajos*maquinas;
%crear matriz de individuos en ceros
poblacion=zeros(tamPob,tamIndv);

 for k=1:tamPob
    control=1;
% generar un vector permutado aleatorio
    gen=randperm(tamIndv);

% Para todas las máquinas y cada trabajo
     for i=1:maquinas
        for j=1:trabajos

%asignar en la fila k de la matriz de individuos el numero de un
trabajo en la columna que indica gen(control)
             poblacion(k,gen(control))=j;
             control=control+1;
        end
    end

end
```

Función DecodificacionBL

Realiza la decodificación de cada cromosoma de la matriz de población para hallar el makespan y el Gantt correspondiente. Posteriormente se hallan las rutas críticas y respectivos bloques para ejecutar el procedimiento de búsqueda local.

Entradas:

- *poblacion*: matriz de población
- *mmaq*: Matriz de secuencia de operaciones en las máquinas para cada trabajo.
- *mtiempo*: Matriz que contiene los tiempos de procesamiento para cada operación.

Salidas:

- *gantt*: Diagrama de Gantt de cada individuo de la población.
- *makespan*: makespan de cada uno de los cromosomas decodificados.
- *poblaciónTotal*: matriz de la población inicial y los vecinos generados en la búsqueda local.

```
function[makespan,gantt,poblacionTotal]=DecodificacionBL(trabajos,maqu
inas,poblacion, mtiempo, mmaq)

%%  inicialización de variables

totalIndividuos=size(poblacion,1);
makespan=zeros(1,totalIndividuos);
matrizvecinos=zeros(totalIndividuos,(trabajos*maquinas));
makespanvecinos=zeros(1,totalIndividuos);

%% ciclo para realizar el proceso en todas los individuos

for k=1:totalIndividuos

%% se limpian las matrices ya utilizadas

clear gantt tcontrol vcontrol

%% inicialización de vectores de control de operación y tiempo inicial
tcontrol=ones(1,trabajos);
vcontrol=ones(1,trabajos);
suma=0;
%tiempo máximo de realizar todo los trabajos
for i=1:trabajos
    suma=suma+sum(mtiempo(i,:));
end

%Crear matriz que representa el diagrama de Gantt
```

```
    gantt=zeros(maquinas,suma);

%Para cada gen del cromosoma
for i=1:trabajos*maquinas
%Determinar qué trabajo se va a programar
    trabajo=poblacion(k,i);
%Determinar qué operación corresponde a ese trabajo
    op=vcontrol(trabajo);
%Determinar en qué máquina se procesa esa operación
    maq=mmaq(trabajo,op);
%Determinar el tiempo de procesamiento de la operación
    tiempo=mtiempo(trabajo,op);
    tlocal=tcontrol(trabajo);

%Asignar la operación a la máquina correspondiente en la matriz gantt
en el menor tiempo posible.

    while gantt(maq,tlocal)
        tcontrol(trabajo)=tcontrol(trabajo)+1;
        tlocal=tcontrol(trabajo);
    end
    mComprobacion=find(gantt(maq,tlocal:suma));
    if isempty(mComprobacion)
       Disponible=suma;
    else
            tDisponible=mComprobacion(1)-1;
    end

    while sum(gantt(maq,tlocal:suma)) && tDisponible<tiempo
        tcontrol(trabajo)=tcontrol(trabajo)+1;
        tlocal=tcontrol(trabajo);
        mComprobacion=find(gantt(maq,tlocal:suma));

        if isempty(mComprobacion)
        tDisponible=suma;
        else
            tDisponible=mComprobacion(1)-1;
        end
    end
    gantt(maq,tlocal:tlocal+tiempo-1)=trabajo;
    tcontrol(trabajo)=tcontrol(trabajo)+tiempo;
    vcontrol(trabajo)=vcontrol(trabajo)+1;
end
makespan(k)=max(tcontrol)-1;

%% generación de rutas y bloques críticos de cada individuo

%Inicializar variables
ms=1;
ctrl=1;
ctrlgral=1;
clear vruta vbloque vrutatemp vbloquetemp
vruta=zeros(1,trabajos*maquinas);
vmaqtemp=0;
vbloque=vruta;
vbloquetemp=vruta;
vrutatemp=vruta;
```

```
%Ejecutar la función Rutas
[vruta, vbloque, ctrlgral]= Rutas(gantt, mmaq, mtiempo, ctrl,
ctrlgral, vrutatemp, vruta, vmaqtemp, vbloque, ms, trabajos, maquinas,
makespan(k), vbloquetemp);

%% Aplicar la búsqueda local para generar vecinos
[matrizbusqueda]=BusquedaLocal(poblacion(k,:),vruta,vbloque,trabajos,m
maq);

%% decodificación de los vecinos y selección de los mejores vecinos
[makespanbl, ganttmb]= DecodificacionHijos(trabajos, maquinas,
matrizbusqueda, mtiempo, mmaq);
[makespanmejor,indice]=min(makespanbl);
makespanvecinos(1,k)=makespanmejor;
matrizvecinos(k,:)=matrizbusqueda(indice,:);
end

%Obtener vector de makespan y matriz de población
makespan=[makespan,makespanvecinos];
poblacionTotal=[poblacion;matrizvecinos];
```

Función Rutas

Construye todas las rutas críticas de cada uno de los Schedules decodificados en el diagrama de Gantt.

Entrada:

- *gantt*: Diagrama de Gantt de cada cromosoma de la población

Salidas:

- *vruta*: Matriz de rutas críticas de cada cromosoma (Schedule).
- *vbloque*: Matriz que corresponde a los bloques críticos de cada una de las rutas críticas guardadas en vruta.

```
function [vruta, vbloque,ctrlgral]=Rutas(gantt, mmaq, mtiempo, ctrl,
ctrlgral, vrutatemp, vruta, vmaqtemp, vbloque, ms, trabajos, maquinas,
makespan, vbloquetemp)

%Vector que contiene las máquinas donde puede iniciar cada ruta
temporal
fms=find(gantt(:,ms)~=0
if ms~=1
% Define el recorrido de la ruta temporal
    indfms=find(gantt(fms,ms)~=gantt(fms,ms-1));
else
    indfms=find(gantt(fms,ms)==gantt(fms,ms));
end
saltomaq=size(indfms,1);
if saltomaq~=0
    for i=1:saltomaq
```

```
%Identificar la operación y el trabajo que pertenece a la ruta
        job=gantt(fms(indfms(i)),ms);
        op=find(mmaq(job,:)==fms(indfms(i)));
        tiempo=mtiempo(job,op);
        if ctrl~=1
%Asignar la máquina que corresponde a la operación en la ruta
            if vmaqtemp==mmaq(job,op)
%Formar los bloques temporales de cada ruta temporal
               vbloquetemp(ctrl)=vbloquetemp(ctrl-1);
            else
                 vbloquetemp(ctrl)=vbloquetemp(ctrl-1)+1;
            end
        else
            vbloquetemp(ctrl)=1;
            vmaqtemp=mmaq(job,op);
        end
        vrutatemp(ctrl)=sub2ind([trabajos,maquinas],job,op);
        actrl=ctrl+1;
%Verificar que la última operación finalice en el valor del makespan
        if (ms+tiempo)==makespan+1
%Se determina el vector de cada bloque
            vbloque(ctrlgral,1:size(vbloquetemp,2))=vbloquetemp;
%Se establece el vector de la ruta crítica
            vruta(ctrlgral,1:size(vrutatemp,2))=vrutatemp;
            ctrlgral=ctrlgral+1;
        else
            ams=ms+tiempo;
%Realizar el mismo procedimiento para encontrar todas las rutas
críticas posibles
[vruta,vbloque,ctrlgral]=Rutas(gantt,mmaq,mtiempo,actrl,ctrlgral,vruta
temp,vruta,mmaq(job,op),vbloque,ams,trabajos,maquinas,makespan,vbloque
temp;

        end
    end
else

    if (ms)==makespan+1
        vbloque(ctrlgral,1:size(vbloquetemp,2))=vbloquetemp;
        vruta(ctrlgral,1:size(vrutatemp,2))=vrutatemp;
        ctrlgral=ctrlgral+1;
    end
end
```

Función BusquedaLocal

Realiza el procedimiento de la búsqueda local, generando los vecinos por cada Schedule.

Entradas:

- *cromosoma*: Codificación del Schedule de cada individuo de la población.

- *vruta*: Matriz de rutas críticas de cada cromosoma (Schedule).
- *vbloque*: Matriz que corresponde a los bloques críticos de cada una de las rutas críticas guardadas en vruta.

Salida:

- *matrizbusqueda*: Matriz con los mejores vecinos obtenidos en el procedimiento de BL.

```
Function[matrizbusqueda]=BusquedaLocal(cromosoma,vruta,vbloque,trabajo
s,mmaq)

% Hallar la ruta crítica más larga
mruta=double(vruta~=0);
sruta=sum(mruta');
operaciones=size(mmaq,2);
maxruta=max(sruta);
mmaxruta=find(sruta==maxruta);

%% Determinar bloque de la ruta más larga

%Determinar si hay más de una ruta crítica larga
if size(mmaxruta,2)>1
    control=1;
    for i=1:size(mmaxruta,2)
%Encontrar el número de bloques que contiene la ruta
        maximo=max(vbloque(mmaxruta(i),:));
        if i==1
            bmaximo=maximo;
            vposmax(control)=mmaxruta(1,i);
        else
            if maximo<bmaximo
                vposmax=(1);
                control=1;
                bmaximo=maximo;
                vposmax(control)=mmaxruta(i);
            elseif maximo==bmaximo
                    control=control+1;
                    bmaximo=maximo;
                    vposmax(control)=mmaxruta(i);
            end
        end
    end
    if control>1
        controlruta=randi(control,1);
        controlruta=find(mmaxruta==vposmax(controlruta));
        mmaxruta=mmaxruta(controlruta);

    else
        controlruta=find(mmaxruta==vposmax(control));
        mmaxruta=mmaxruta(1,controlruta);
    end
    tamano=find(vruta(mmaxruta,:)==0);
```

```
    tamano=tamano(1)-1;
    rutalarga=vruta(mmaxruta,1:tamano);
    brutalarga=vbloque(mmaxruta,1:tamano);
else
    tamano=find(vruta(mmaxruta,:)==0);
    tamano=tamano(1)-1;
    rutalarga=vruta(mmaxruta,1:tamano);
    brutalarga=vbloque(mmaxruta,1:tamano);
end

%% Separación de los bloques

maxbloques=max(brutalarga);
control=1;
for i=1:maxbloques
    bloquetemp=find(brutalarga==i);
    mbloques(i)=size(bloquetemp,2);
       if mbloques(i)>1 && control<2
        bloques(control)=i;
        control=control+1;
       end
end

for i=maxbloques:-1:1
    if mbloques(i)>1 && control<3 && bloques(control-1)<i
        bloques(control)=i;
        control=control+1;
    end
end

%% Generar vecinos

if control>2
%Encontrar el primer bloque de la ruta crítica
    mv1=find(brutalarga==bloques(1));
%Encontrar el último bloque de la ruta crítica
    mv2=find(brutalarga==bloques(2));
    v(1,:)=rutalarga;
    v(2,:)=rutalarga;

%% Se genera el vecino 1

%Intercambio de las dos últimas operaciones del primer bloque de la RC
    [in1a,in1b]=ind2sub([trabajos,operaciones],v(1,mv1(size(mv1,2))));
    [in2a,in2b]=ind2sub([trabajos,operaciones],v(1,mv1(size(mv1,2)-1)));
    cromosoma1=cromosoma;
    pos1=find(cromosoma==in1a);
    pos2=find(cromosoma==in2a);
    cromosoma1(pos1(in1b))=cromosoma(pos2(in2b));
    cromosoma1(pos2(in2b))=cromosoma(pos1(in1b));

%% Se genera el vecino 2

%Intercambio de las dos primeras operaciones del último bloque de la
RC
    [in1a,in1b]=ind2sub([trabajos,operaciones],v(2,mv2(1)));
```

```
    [in2a,in2b]=ind2sub([trabajos,operaciones],v(2,mv2(2)));
    cromosoma2=cromosoma;
    pos1=find(cromosoma==in1a);
    pos2=find(cromosoma==in2a);
    cromosoma2(pos1(in1b))=cromosoma(pos2(in2b));
    cromosoma2(pos2(in2b))=cromosoma(pos1(in1b));

%% Se genera el vecino 3

%Intercambio simultáneo de las operaciones en el primer y último
bloque
    [in1a,in1b]=ind2sub([trabajos,operaciones],v(1,mv1(size(mv1,2))));
    [in2a,in2b]=ind2sub([trabajos,operaciones],v(1,mv1(size(mv1,2)-1)));
    cromosoma3=cromosoma;
    pos1=find(cromosoma==in1a);
    pos2=find(cromosoma==in2a);
    cromosoma3(pos1(in1b))=cromosoma(pos2(in2b));
    cromosoma3(pos2(in2b))=cromosoma(pos1(in1b));
    [in1a,in1b]=ind2sub([trabajos,operaciones],v(2,mv2(1)));
    [in2a,in2b]=ind2sub([trabajos,operaciones],v(2,mv2(2)));
    pos1=find(cromosoma==in1a);
    pos2=find(cromosoma==in2a);
    cromosoma3(pos1(in1b))=cromosoma(pos2(in2b));
    cromosoma3(pos2(in2b))=cromosoma(pos1(in1b));
    matrizbusqueda=[cromosoma1;cromosoma2;cromosoma3];
else

%Si existe un solo bloque, se realiza el intercambio de las dos
últimas operaciones de ese bloque
    mv1=find(brutalarga==bloques);
    v(1,:)=rutalarga;
    [in1a,in1b]=ind2sub([trabajos,operaciones],v(1,mv1(size(mv1,2))));
    [in2a,in2b]=ind2sub([trabajos,operaciones],v(1,mv1(size(mv1,2)-1)));
    cromosoma1=cromosoma;
    pos1=find(cromosoma==in1a);
    pos2=find(cromosoma==in2a);
    cromosoma1(pos1(in1b))=cromosoma(pos2(in2b));
    cromosoma1(pos2(in2b))=cromosoma(pos1(in1b));
    matrizbusqueda=cromosoma1;
end
```

Función Seleccion

Construye una matriz con los individuos seleccionados a partir de la matriz de poblacionTotal, de acuerdo al porcentaje de selección establecido.

Entrada:

- *porcentajeSeleccion*: Porcentaje de mejores individuos de acuerdo al makespan que se requiere que pase al siguiente operador.
- *poblacionTotal*: matriz de la población inicial y los vecinos generados en la búsqueda local.

- *makespan*: Vector que arroja el valor de la función objetivo de cada cromosoma decodificado.

Salida:

- *mPadres*: matriz de los cromosomas seleccionados que tomarán el rol de padres en el operador de cruce.

```
function [mPadres,
indices]=Seleccion(porcentajeSeleccion,poblacionTotal, makespan)

%Ordenar el vector makespan de menor a mayor
[makespanOrd, indices]=sort(makespan);

tamPob=size(makespan,2)/2;

%Cantidad de mejores individuos que serán seleccionados
porseleccion=round(porcentajeSeleccion*tamPob);

%Cantidad de individuos no tan buenos a seleccionar
porseleccion1=round((1-porcentajeSeleccion)*tamPob);

%Seleccionar aleatoriamente los individuos buenos
posicion=randperm(floor(tamPob),porseleccion);

%Seleccionar aleatoriamente los individuos no tan buenos
posicion1=floor(tamPob)+randperm(floor(tamPob),porseleccion1);
posiciones=[posicion,posicion1];

%La cantidad total de individuos a seleccionar debe ser un número par
if mod(size(posiciones,2),2)~=0
    eliminarposicion=randi((size(posiciones,2)),1);
    posiciones(:,eliminarposicion)=[];
end

%Se construye la matriz de cromosomas seleccionados
mPadres=poblacionTotal(indices(posiciones),:);
```

Funcion Cruce

Construye una matriz con los cromosomas (hijos) aplicando el operador de cruce (JOX).

Entrada:

- *mPadres:* matriz de los cromosomas seleccionados que tomarán el rol de padres en el operador de cruce.

Salida:

- *mHijos:* matriz de cromosomas descendientes, a los cuales se aplicó el operador de cruce JOX.

```
function mHijos=Cruce(trabajos,maquinas,mPadres)

tamIndv=trabajos*maquinas;

%Crear matriz de hijos en ceros
mHijos=zeros(size(mPadres,1),tamIndv);
trabajoelegido=randi(trabajos,1);
controlhijos=1;

%Generar un vector de números aleatorios para seleccionar las parejas
de padres a los que se realizará el cruce
padrealeatorio=randperm(size(mPadres,1));
for i=1:2:size(padrealeatorio,2)-1
      j=i+1;

%%procedimiento para el hijo A

%En el hijo A se deja estático el trabajo seleccionado en la misma
posición del padre 1.

hA=double(mPadres(padrealeatorio(1,i),:)==trabajoelegido)*trabajoelegi
do;
      control=1;
      for k=1:size(hA,2)
      %Llenar el cromosoma con las posiciones del segundo padre
            if hA(k)~=trabajoelegido
            while
mPadres(padrealeatorio(1,j),control)==trabajoelegido)
                  control=control+1;
            end
            hA(k)=mPadres(padrealeatorio(1,j),control);
            control=control+1;
            end

      end

%%procedimiento para el hijo B

%En el hijo B se deja estático el trabajo seleccionado en la misma
posición del padre 2.

      hB=double(mPadres(padrealeatorio(1,j),:)==trabajoelegido)*trabaj
      oelegido;
        control=1;
        for k=1:size(hB,2)
      %Llenar el cromosoma con las posiciones del primer padre
            if hB(k)~=trabajoelegido
                while
Padres(padrealeatorio(1,i),control)==trabajoelegido
                      control=control+1;
                end
               hB(k)=mPadres(padrealeatorio(1,i),control);
               control=control+1;
            end
        end
```

```
%Se actualiza la matriz de hijos generados
    mHijos(controlhijos,:)=hA;
    controlhijos=controlhijos+1;
    mHijos(controlhijos,:)=hB;
    controlhijos=controlhijos+1;

 end
```

Función Mutacion

Actualiza la matriz de hijos generada en el procedimiento de cruce, con los cromosomas (hijos) que han sido mutados.

Entrada:

- *Pormutacion:* Probabilidad que un cromosoma sea mutado, se asigna en decimales.
- *mHijos:* matriz de cromosomas descendientes, a los cuales se aplicó el operador de cruce JOX.

Salida:

- *mHijos:* matriz de cromosomas (hijos) que han sido mutados.

```
function mHijos=Mutacion(pormutacion,mHijos,poblacion,tamPob, indices)

%Para cada hijo generado en cruce
 for i=1:size(mHijos,1)
%Generar el número aleatorio que determina si se realiza la mutación
    aleatorio=rand(1);
     if  aleatorio<=pormutacion
%Se elige aleatoriamente las dos posiciones para hacer el intercambio
        posicion=randperm(size(mHijos,2),2);
        m1=mHijos(i,posicion(1,1));
        mHijos(i,posicion(1,1))=mHijos(i,posicion(1,2));
        mHijos(i,posicion(1,2))=m1;
    end
end
```

Función DecodificacionHijos

Se realiza el diagrama de Gantt por medio de una matriz para cada cromosoma generado en los procesos de búsqueda local, cruce y mutación para hallar el makespan y elegir el mejor valor (mínimo).

Entrada:

- *mHijos:* matriz de cromosomas descendientes, a los cuales se aplicó el operador de cruce JOX.

- *mHijos:* matriz actualizada de cromosomas (hijos) que han sido mutados.

Salida:

- *makespan:* Vector que arroja el valor de la función objetivo de cada cromosoma decodificado.

```
function [makespan,gantt]=DecodificacionHijos(trabajos, maquinas,
poblacion, mtiempo, mmaq)

%%  inicialización de variables
totalIndividuos=size(poblacion,1);
makespan=zeros(1,totalIndividuos);

%% ciclo para realizar el proceso en todas los individuos
 for k=1:totalIndividuos

     clear gantt tcontrol vcontrol

%% inicialización de vectores de control de operación y tiempo inicial

tcontrol=ones(1,trabajos);
vcontrol=ones(1,trabajos);
suma=0;

%tiempo máximo de realizar todo los trabajos
for i=1:trabajos
    suma=suma+sum(mtiempo(i,:));
end

%Crear matriz que representa el diagrama de Gantt
    gantt=zeros(maquinas,suma);
%Para cada gen del cromosoma
for i=1:trabajos*maquinas
%Determinar que trabajo se va a programar
    trabajo=poblacion(k,i);
%Determinar qué operación corresponde a ese trabajo
    op=vcontrol(trabajo);
%Determinar en qué maquina se procesa esa operación
    maq=mmaq(trabajo,op);
%Determinar el tiempo de procesamiento de la operación

    tiempo=mtiempo(trabajo,op);
    tlocal=tcontrol(trabajo);

%Asignar la operación a su máquina correspondiente en la matriz Gantt
en el menor tiempo posible.

    while gantt(maq,tlocal)
        tcontrol(trabajo)=tcontrol(trabajo)+1;
        tlocal=tcontrol(trabajo);
    end

    mComprobacion=find(gantt(maq,tlocal:suma));
```

```
        if isempty(mComprobacion)
            tDisponible=suma;
        else
            tDisponible=mComprobacion(1)-1;
        end

    while sum(gantt(maq,tlocal:suma)) && tDisponible<tiempo
        tcontrol(trabajo)=tcontrol(trabajo)+1;
        tlocal=tcontrol(trabajo);
        mComprobacion=find(gantt(maq,tlocal:suma));
        if isempty(mComprobacion)
        tDisponible=suma;
        else
            tDisponible=mComprobacion(1)-1;
        end
    end
    gantt(maq,tlocal:tlocal+tiempo-1)=trabajo;
    tcontrol(trabajo)=tcontrol(trabajo)+tiempo;
    vcontrol(trabajo)=vcontrol(trabajo)+1;
end

%Vector de makespan
makespan(k)=max(tcontrol)-1;
end
```

Función AlgoritmoGenetico

Esta función muestra el procedimiento de ejecución de las funciones mencionadas anteriormente para llevar a cabo el algoritmo memético final. Además realiza la comparación de todos los makespan arrojados durante el proceso para elegir el mejor de todos (mínimo makespan).

```
function [makespanminfinal,cromosomafinal]=AlgoritmoGenetico(trabajos,
maquinas,tamPob,porcentajeSeleccion,pormutacion,mtiempo,mmaq,
generaciones)

cromosomafinal=zeros(1,trabajos*maquinas);
makespanminfinal=0;

%Número de generaciones
for i=1:generaciones

%Población inicial para la primera iteración
    if i==1
        poblacion=nuevapoblacion(trabajos, maquinas, tamPob);
    else

%Población inicial proveniente del proceso de mutación de la segunda
iteración en adelante
        poblacion=mHijos;
    end
```

```
%Procedimiento de búsqueda local y decodificación
[makespan,  gantt,poblacionTotal]=DecodificacionBL(trabajos,  maquinas,
poblacion, mtiempo, mmaq);

%Aplicar operador de selección
[mPadres,indices]=Seleccion(porcentajeSeleccion,poblacionTotal,makespa
n);

%Aplicar operador de cruce y guardar mejor valor del makespan
mHijos=Cruce(trabajos,maquinas,mPadres);
[makespanmb,ganttmb]=DecodificacionHijos(trabajos,maquinas,mHijos,mtie
mpo,mmaq);
    control=0;
    if i==1
        comparacion=double(makespanmb<min(makespan));
        if sum(comparacion)>0
            control=1;
            [makespanmin,indic]=min(makespanmb);
            cromosomafinal=mHijos(indic,:);
            makespanminfinal=makespanmin;
        end
    else
      [makespanmin,indic]=min(makespanmb);
      if makespanmin<makespanminfinal
          cromosomafinal=mHijos(indic,:);
         makespanminfinal=makespanmin;
      end
    end

%Aplicar operador de mutación
    mHijos=Mutacion(pormutacion,mHijos,     poblacionTotal,     tamPob,
indices);

%Calcular el makespan mínimo
    [makespanmin,indices]=min(makespan);

    if i==1 && control==0
        cromosomafinal=poblacionTotal(indices,:);
        makespanminfinal=makespanmin;
    else
        if makespanmin<makespanminfinal
            cromosomafinal=poblacionTotal(indices,:);
            makespanminfinal=makespanmin;
        end
    end

end
[makespanmin,indices]=min(makespan);
if makespanmin<makespanminfinal
    cromosomafinal=poblacionTotal(indices,:);
    makespanminfinal=makespanmin;
end
```

AlgoritmoMemetico

Se ingresan los parámetros del problema: Datos de las instancias (matriz de secuencia de máquinas y tiempos de procesamiento), especificar cantidad de trabajos y máquinas, tamaño de población, generaciones, porcentaje de selección y porcentaje de mutación.

Se ejecuta la función de algoritmo genético para obtener el mínimo makespan obtenido en todo el procedimiento y su respectivo Schedule.

```
clear all
clc
tic
load('mmaqft6.mat');
load('mtiempoft6.mat');
trabajos=6; %Cantidad de trabajos de la instancia
maquinas=6; %Cantidad de máquinas de la instancia
tamPob=10; %Tamaño que se va a mantener fijo durante las generaciones
porcentajeSeleccion=0.7; %Probabilidad dada en decimales
pormutacion=0.1; %Probabilidad dada en decimales
generaciones=5; %Numero de generaciones

[makespanfinal,cromosomafinal]=AlgoritmoGenetico(trabajos,
maquinas,tamPob,                                    porcentajeSeleccion,
pormutacion,mtiempo,mmaq,generaciones);

disp(['el makespan minimo final es ', num2str(makespanfinal)]);
disp(['el    cromosoma    del    makespan    minimo    final    es:
',num2str(cromosomafinal)]);
toc
```

ANEXO G. INSTANCIAS DEL BENCHMARKING

La información de las instancias del benchmarking aportadas por la OR-Library que fueron validadas mediante el algoritmo memético propuesto se relacionan a continuación.

En cada instancia se describe sus autores y su tamaño, es decir, la cantidad de trabajos y máquinas que contiene. En la matriz, cada fila corresponde a la secuencia de máquinas para cada trabajo con su respectivo tiempo de procesamiento (ti). Cabe aclarar que las máquinas son enumeradas iniciando desde cero.

Instancia Ft 06

Fisher y Thomspon (6x6): 6 trabajos y 6 máquinas

Tabla 40. Datos de la instancia Ft 06

		t1		t2		t3		t4		t5		t6
J1	2	1	0	3	1	6	3	7	5	3	4	6
J2	1	8	2	5	4	10	5	10	0	10	3	4
J3	2	5	3	4	5	8	0	9	1	1	4	7
J4	1	5	0	5	2	5	3	3	4	8	5	9
J5	2	9	1	3	4	5	5	4	0	3	3	1
J6	1	3	3	3	5	9	0	10	4	4	2	1

Instancia Ft 10

Fisher y Thompson (10x10): 10 trabajos y 10 máquinas

Tabla 41. Datos de la instancia Ft 10

		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10
J1	0	29	1	78	2	9	3	36	4	49	5	11	6	62	7	56	8	44	9	21
J2	0	43	2	90	4	75	9	11	3	69	1	28	6	46	5	46	7	72	8	30
J3	1	91	0	85	3	39	2	74	8	90	5	10	7	12	6	89	9	45	4	33
J4	1	81	2	95	0	71	4	99	6	9	8	52	7	85	3	98	9	22	5	43
J5	2	14	0	6	1	22	5	61	3	26	4	69	8	21	7	49	9	72	6	53
J6	2	84	1	2	5	52	3	95	8	48	9	72	0	47	6	65	4	6	7	25
J7	1	46	0	37	3	61	2	13	6	32	5	21	9	32	8	89	7	30	4	55
J8	2	31	0	86	1	46	5	74	4	32	6	88	8	19	9	48	7	36	3	79
J9	0	76	1	69	3	76	5	51	2	85	9	11	6	40	7	89	4	26	8	74
J10	1	85	0	13	2	61	6	7	8	64	9	76	5	47	3	52	4	90	7	45

Instancia Ft 20

Fisher y Thompson (20x5): 20 trabajos y 5 máquinas

Tabla 42. Datos de la instancia Ft 20

		t1		t2		t3		t4		t5
J1	0	29	1	9	2	49	3	62	4	44
J2	0	43	1	75	3	69	2	46	4	72
J3	1	91	0	39	2	90	4	12	3	45
J4	1	81	0	71	4	9	2	85	3	22
J5	2	14	1	22	0	26	3	21	4	72
J6	2	84	1	52	4	48	0	47	3	6
J7	1	46	0	61	2	32	3	32	4	30
J8	2	31	1	46	0	32	3	19	4	36
J9	0	76	3	76	2	85	1	40	4	26
J10	1	85	2	61	0	64	3	47	4	90
J11	1	78	3	36	0	11	4	56	2	21
J12	2	90	0	11	1	28	3	46	4	30
J13	0	85	2	74	1	10	3	89	4	33
J14	2	95	0	99	1	52	3	98	4	43
J15	0	6	1	61	4	69	2	49	3	53
J16	1	2	0	95	3	72	4	65	2	25
J17	0	37	2	13	1	21	3	89	4	55
J18	0	86	1	74	4	88	2	48	3	79
J19	1	69	2	51	0	11	3	89	4	74
J20	0	13	1	7	2	76	3	52	4	45

Instancia La 01

Lawrence (10x5): 10 trabajos y 5 máquinas

Tabla 43. Datos de la instancia La 01

		t1		t2		t3		t4		t5
J1	1	21	0	53	4	95	3	55	2	34
J2	0	21	3	52	4	16	2	26	1	71
J3	3	39	4	98	1	42	2	31	0	12
J4	1	77	0	55	4	79	2	66	3	77
J5	0	83	3	34	2	64	1	19	4	37
J6	1	54	2	43	4	79	0	92	3	62
J7	3	69	4	77	1	87	2	87	0	93
J8	2	38	0	60	1	41	3	24	4	83
J9	3	17	1	49	4	25	0	44	2	98
J10	4	77	3	79	2	43	1	75	0	96

Instancia La 02

Lawrence (10x5): 10 trabajos y 5 máquinas

Tabla 44. Datos de la instancia La 02

		t1		t2		t3		t4		t5
J1	0	20	3	87	1	31	4	76	2	17
J2	4	25	2	32	0	24	1	18	3	81
J3	1	72	2	23	4	28	0	58	3	99
J4	2	86	1	76	4	97	0	45	3	90
J5	4	27	0	42	3	48	2	17	1	46
J6	1	67	0	98	4	48	3	27	2	62
J7	4	28	1	12	3	19	0	80	2	50
J8	1	63	0	94	2	98	3	50	4	80
J9	4	14	0	75	2	50	1	41	3	55
J10	4	72	2	18	1	37	3	79	0	61

Instancia La 03

Lawrence (10x5): 10 trabajos y 5 máquinas

Tabla 45. Datos de la instancia La 03

		t1		t2		t3		t4		t5
J1	1	23	2	45	0	82	4	84	3	38
J2	2	21	1	29	0	18	4	41	3	50
J3	2	38	3	54	4	16	0	52	1	52
J4	4	37	0	54	2	74	1	62	3	57
J5	4	57	0	81	1	61	3	68	2	30
J6	4	81	0	79	1	89	2	89	3	11
J7	3	33	2	20	0	91	4	20	1	66
J8	4	24	1	84	0	32	2	55	3	8
J9	4	56	0	7	3	54	2	64	1	39
J10	4	40	1	83	0	19	2	8	3	7

Instancia La 04

Lawrence (10x5): 10 trabajos y 5 máquinas

Tabla 46. Datos de la instancia La 04

		t1		t2		t3		t4		t5
J1	0	12	2	94	3	92	4	91	1	7
J2	1	19	3	11	4	66	2	21	0	87
J3	1	14	0	75	3	13	4	16	2	20
J4	2	95	4	66	0	7	3	7	1	77
J5	1	45	3	6	4	89	0	15	2	34
J6	3	77	2	20	0	76	4	88	1	53
J7	2	74	1	88	0	52	3	27	4	9
J8	1	88	3	69	0	62	4	98	2	52
J9	2	61	4	9	0	62	1	52	3	90
J10	2	54	4	5	3	59	1	15	0	88

Instancia La 05

Lawrence (10x5): 10 trabajos y 5 máquinas

Tabla 47. Datos de la instancia La 05

		t1		t2		t3		t4		t5
J1	1	72	0	87	4	95	2	66	3	60
J2	4	5	3	35	0	48	2	39	1	54
J3	1	46	3	20	2	21	0	97	4	55
J4	0	59	3	19	4	46	1	34	2	37
J5	4	23	2	73	3	25	1	24	0	28
J6	3	28	0	45	4	5	1	78	2	83
J7	0	53	3	71	1	37	4	29	2	12
J8	4	12	2	87	3	33	1	55	0	38
J9	2	49	3	83	1	40	0	48	4	7
J10	2	65	3	17	0	90	4	27	1	23

Instancia La 06

Lawrence (15x5): 15 trabajos y 5 máquinas

Tabla 48. Datos de la instancia La 06

		t1		t2		t3		t4		t5
J1	1	21	2	34	4	95	0	53	3	55
J2	3	52	4	16	1	71	2	26	0	21
J3	2	31	0	12	1	42	3	39	4	98
J4	3	77	1	77	4	79	0	55	2	66
J5	4	37	3	34	2	64	1	19	0	83
J6	2	43	1	54	0	92	3	62	4	79
J7	0	93	3	69	1	87	4	77	2	87
J8	0	60	1	41	2	38	4	83	3	24
J9	2	98	3	17	4	25	0	44	1	49
J10	0	96	4	77	3	79	1	75	2	43
J11	4	28	2	35	0	95	3	76	1	7
J12	0	61	4	10	2	95	1	9	3	35
J13	4	59	3	16	1	91	2	59	0	46
J14	4	43	1	52	0	28	2	27	3	50
J15	0	87	1	45	2	39	4	9	3	41

Instancia La 07

Lawrence (15x5): 15 trabajos y 5 máquinas

Tabla 49. Datos de la instancia La 07

		t1		t2		t3		t4		t5
J1	0	47	4	57	1	71	3	96	2	14
J2	0	75	1	60	4	22	3	79	2	65
J3	3	32	0	33	2	69	1	31	4	58
J4	0	44	1	34	4	51	3	58	2	47
J5	3	29	1	44	0	62	2	17	4	8
J6	1	15	2	40	0	97	4	38	3	66
J7	2	58	1	39	0	57	4	20	3	50

J8	2	57	3	32	4	87	0	63	1	21
J9	4	56	0	84	2	90	1	85	3	61
J10	4	15	0	20	1	67	3	30	2	70
J11	4	84	0	82	1	23	2	45	3	38
J12	3	50	2	21	0	18	4	41	1	29
J13	4	16	1	52	0	52	2	38	3	54
J14	4	37	0	54	3	57	2	74	1	62
J15	4	57	1	61	0	81	2	30	3	68

Instancia La 08

Lawrence (15x5): 15 trabajos y 5 máquinas

Tabla 50. Datos de la instancia La 08

		t1		**t2**		**t3**		**t4**		**t5**
J1	3	92	2	94	0	12	4	91	1	7
J2	2	21	1	19	0	87	3	11	4	66
J3	1	14	3	13	0	75	4	16	2	20
J4	2	95	4	66	0	7	1	77	3	7
J5	2	34	4	89	3	6	1	45	0	15
J6	4	88	3	77	2	20	1	53	0	76
J7	4	9	3	27	0	52	1	88	2	74
J8	3	69	2	52	0	62	1	88	4	98
J9	3	90	0	62	4	9	2	61	1	52
J10	4	5	2	54	3	59	0	88	1	15
J11	0	41	1	50	4	78	3	53	2	23
J12	0	38	4	72	2	91	3	68	1	71
J13	0	45	3	95	4	52	2	25	1	6
J14	3	30	1	66	0	23	4	36	2	17
J15	2	95	0	71	3	76	1	8	4	88

Instancia La 09

Lawrence (15x5): 15 trabajos y 5 máquinas

Tabla 51. Datos de la instancia La 09

		t1		**t2**		**t3**		**t4**		**t5**
J1	1	66	3	85	2	84	0	62	4	19
J2	3	59	1	64	2	46	4	13	0	25
J3	4	88	3	80	1	73	2	53	0	41
J4	0	14	1	67	2	57	3	74	4	47
J5	0	84	4	64	2	41	3	84	1	78
J6	0	63	3	28	1	46	2	26	4	52
J7	3	10	2	17	4	73	1	11	0	64
J8	2	67	1	97	3	95	4	38	0	85
J9	2	95	4	46	0	59	1	65	3	93
J10	2	43	4	85	3	32	1	85	0	60
J11	4	49	3	41	2	61	0	66	1	90
J12	1	17	0	23	3	70	4	99	2	49
J13	4	40	3	73	0	73	1	98	2	68
J14	3	57	1	9	2	7	0	13	4	98
J15	0	37	1	85	2	17	4	79	3	41

Instancia La 10

Lawrence (15x5): 15 trabajos y 5 máquinas

Tabla 52. Datos de la instancia La 10

		t1		t2		t3		t4		t5
J1	1	58	2	44	3	5	0	9	4	58
J2	1	89	0	97	4	96	3	77	2	84
J3	0	77	1	87	2	81	4	39	3	85
J4	3	57	1	21	2	31	0	15	4	73
J5	2	48	0	40	1	49	3	70	4	71
J6	3	34	4	82	2	80	0	10	1	22
J7	1	91	4	75	0	55	2	17	3	7
J8	2	62	3	47	1	72	4	35	0	11
J9	0	64	3	75	4	50	1	90	2	94
J10	2	67	4	20	3	15	0	12	1	71
J11	0	52	4	93	3	68	2	29	1	57
J12	2	70	0	58	1	93	4	7	3	77
J13	3	27	2	82	1	63	4	6	0	95
J14	1	87	2	56	4	36	0	26	3	48
J15	3	76	2	36	0	36	4	15	1	8

Instancia La 11

Lawrence (20x5): 20 trabajos y 5 máquinas

Tabla 53. Datos de la instancia La 11

		t1		t2		t3		t4		t5
J1	2	34	1	21	0	53	3	55	4	95
J2	0	21	3	52	1	71	4	16	2	26
J3	0	12	1	42	2	31	4	98	3	39
J4	2	66	3	77	4	79	0	55	1	77
J5	0	83	4	37	3	34	1	19	2	64
J6	4	79	2	43	0	92	3	62	1	54
J7	0	93	4	77	2	87	1	87	3	69
J8	4	83	3	24	1	41	2	38	0	60
J9	4	25	1	49	0	44	2	98	3	17
J10	0	96	1	75	2	43	4	77	3	79
J11	0	95	3	76	1	7	4	28	2	35
J12	4	10	2	95	0	61	1	9	3	35
J13	1	91	2	59	4	59	0	46	3	16
J14	2	27	1	52	4	43	0	28	3	50
J15	4	9	0	87	3	41	2	39	1	45
J16	1	54	0	20	4	43	3	14	2	71
J17	4	33	1	28	3	26	0	78	2	37
J18	1	89	0	33	2	8	3	66	4	42
J19	4	84	0	69	2	94	1	74	3	27
J20	4	81	2	45	1	78	3	69	0	96

Instancia La 12

Lawrence (20x5): 20 trabajos y 5 máquinas

Tabla 54. Datos de la instancia La 12

		t1		t2		t3		t4		t5
J1	1	23	0	82	4	84	2	45	3	38
J2	3	50	4	41	1	29	0	18	2	21
J3	4	16	3	54	1	52	2	38	0	52
J4	1	62	3	57	4	37	2	74	0	54
J5	3	68	1	61	2	30	0	81	4	57
J6	1	89	2	89	3	11	0	79	4	81
J7	1	66	0	91	3	33	4	20	2	20
J8	3	8	4	24	2	55	0	32	1	84
J9	0	7	2	64	1	39	4	56	3	54
J10	0	19	4	40	3	7	2	8	1	83
J11	0	63	2	64	3	91	4	40	1	6
J12	1	42	3	61	4	15	2	98	0	74
J13	1	80	0	26	3	75	4	6	2	87
J14	2	39	4	22	0	75	3	24	1	44
J15	1	15	3	79	4	8	0	12	2	20
J16	3	26	2	43	0	80	4	22	1	61
J17	2	62	1	36	0	63	3	96	4	40
J18	1	33	3	18	0	22	4	5	2	10
J19	2	64	4	64	0	89	1	96	3	95
J20	2	18	4	23	3	15	1	38	0	8

Instancia La 13

Lawrence (20x5): 20 trabajos y 5 máquinas

Tabla 55. Datos de la instancia La 13

		t1		t2		t3		t4		t5
J1	3	60	0	87	1	72	4	95	2	66
J2	1	54	0	48	2	39	3	35	4	5
J3	3	20	1	46	0	97	2	21	4	55
J4	2	37	0	59	3	19	1	34	4	46
J5	2	73	3	25	1	24	0	28	4	23
J6	1	78	3	28	2	83	0	45	4	5
J7	3	71	1	37	2	12	4	29	0	53
J8	4	12	3	33	1	55	2	87	0	38
J9	0	48	1	40	2	49	3	83	4	7
J10	0	90	4	27	2	65	3	17	1	23
J11	0	62	3	85	1	66	2	84	4	19
J12	3	59	2	46	4	13	1	64	0	25
J13	2	53	1	73	3	80	4	88	0	41
J14	2	57	4	47	0	14	1	67	3	74
J15	2	41	4	64	3	84	1	78	0	84
J16	4	52	3	28	2	26	0	63	1	46
J17	1	11	0	64	3	10	4	73	2	17
J18	4	38	3	95	0	85	1	97	2	67
J19	3	93	1	65	2	95	0	59	4	46
J20	0	60	1	85	2	43	4	85	3	32

Instancia La 14

Lawrence (20x5): 20 trabajos y 5 máquinas

Tabla 56. Datos de la instancia La 14

		t1		t2		t3		t4		t5
J1	3	5	4	58	2	44	0	9	1	58
J2	1	89	4	96	0	97	2	84	3	77
J3	2	81	3	85	1	87	4	39	0	77
J4	0	15	3	57	4	73	1	21	2	31
J5	2	48	4	71	3	70	0	40	1	49
J6	0	10	4	82	3	34	2	80	1	22
J7	2	17	0	55	1	91	4	75	3	7
J8	3	47	2	62	1	72	4	35	0	11
J9	1	90	2	94	4	50	0	64	3	75
J10	3	15	2	67	0	12	4	20	1	71
J11	4	93	2	29	0	52	1	57	3	68
J12	3	77	1	93	0	58	2	70	4	7
J13	1	63	3	27	0	95	4	6	2	82
J14	4	36	0	26	3	48	2	56	1	87
J15	2	36	1	8	4	15	3	76	0	36
J16	4	78	1	84	3	41	0	30	2	76
J17	1	78	0	75	4	88	3	13	2	81
J18	0	54	4	40	2	13	1	82	3	29
J19	1	26	4	82	0	52	3	6	2	6
J20	3	54	1	64	0	54	2	32	4	88

Instancia La 15

Lawrence (20x5): 20 trabajos y 5 máquinas

Tabla 57. Datos de la instancia La 15

		t1		t2		t3		t4		t5
J1	0	6	2	40	1	81	3	37	4	19
J2	2	40	3	32	0	55	4	81	1	9
J3	1	46	4	65	2	70	3	55	0	77
J4	2	21	4	65	0	64	3	25	1	15
J5	2	85	0	40	1	44	3	24	4	37
J6	0	89	4	29	1	83	3	31	2	84
J7	4	59	3	38	1	80	2	30	0	8
J8	0	80	2	56	1	77	4	41	3	97
J9	4	56	0	91	3	50	2	71	1	17
J10	1	40	0	88	4	59	2	7	3	80
J11	0	45	1	29	2	8	4	77	3	58
J12	2	36	0	54	3	96	1	9	4	10
J13	0	28	2	73	1	98	3	92	4	87
J14	0	70	3	86	2	27	1	99	4	96
J15	1	95	0	59	4	56	3	85	2	41
J16	1	81	2	92	4	32	0	52	3	39
J17	1	7	4	22	2	12	0	88	3	60
J18	3	45	0	93	2	69	4	49	1	27
J19	0	21	1	84	2	61	3	68	4	26
J20	1	82	2	33	4	71	0	99	3	44

Instancia La 16

Lawrence (10x10): 10 trabajos y 10 máquinas

Tabla 58. Datos de la instancia La 16

		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10
J1	1	21	6	71	9	16	8	52	7	26	2	34	0	53	4	21	3	55	5	95
J2	4	55	2	31	5	98	9	79	0	12	7	66	1	42	8	77	6	77	3	39
J3	3	34	2	64	8	62	1	19	4	92	9	79	7	43	6	54	0	83	5	37
J4	1	87	3	69	2	87	7	38	8	24	9	83	6	41	0	93	5	77	4	60
J5	2	98	0	44	5	25	6	75	7	43	1	49	4	96	9	77	3	17	8	79
J6	2	35	3	76	5	28	9	10	4	61	6	9	0	95	8	35	1	7	7	95
J7	3	16	2	59	0	46	1	91	9	43	8	50	6	52	5	59	4	28	7	27
J8	1	45	0	87	3	41	4	20	6	54	9	43	8	14	5	9	2	39	7	71
J9	4	33	2	37	8	66	5	33	3	26	7	8	1	28	6	89	9	42	0	78
J10	8	69	9	81	2	94	4	96	3	27	0	69	7	45	6	78	1	74	5	84

Instancia La 17

Lawrence (10x10): 10 trabajos y 10 máquinas

Tabla 59. Datos de la instancia La 17

		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10
J1	4	18	7	21	9	41	2	45	3	38	8	50	5	84	6	29	1	23	0	82
J2	8	57	5	16	1	52	7	74	2	38	3	54	6	62	9	37	4	54	0	52
J3	2	30	4	79	3	68	1	61	8	11	6	89	7	89	0	81	9	81	5	57
J4	0	91	8	8	3	33	7	55	5	20	2	20	4	32	6	84	1	66	9	24
J5	9	40	0	7	4	19	8	7	6	83	2	64	5	56	3	54	7	8	1	39
J6	3	91	2	64	5	40	0	63	7	98	4	74	8	61	1	6	6	42	9	15
J7	1	80	7	39	8	24	3	75	4	75	5	6	6	44	0	26	2	87	9	22
J8	1	15	7	43	2	20	0	12	8	26	6	61	3	79	9	22	5	8	4	80
J9	2	62	3	96	4	22	9	5	0	63	6	33	7	10	8	18	1	36	5	40
J10	1	96	0	89	5	64	3	95	9	23	7	18	8	15	2	64	6	38	4	8

Instancia La 18

Lawrence (10x10): 10 trabajos y 10 máquinas

Tabla 60. Datos de la instancia La 18

		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10
J1	6	54	0	87	4	48	3	60	7	39	8	35	1	72	5	95	2	66	9	5
J2	3	20	9	46	6	34	5	55	0	97	8	19	4	59	2	21	7	37	1	46
J3	4	45	1	24	8	28	0	28	7	83	6	78	5	23	3	25	9	5	2	73
J4	9	12	1	37	4	38	3	71	8	33	2	12	6	55	0	53	7	87	5	29
J5	3	83	2	49	6	23	9	27	7	65	0	48	4	90	5	7	1	40	8	17
J6	1	66	4	25	0	62	2	84	9	13	6	64	7	46	8	59	5	19	3	85
J7	1	73	3	80	0	41	2	53	9	47	7	57	8	74	4	14	6	67	5	88
J8	5	64	3	84	6	46	1	78	0	84	7	26	8	28	9	52	2	41	4	63
J9	1	11	0	64	7	67	4	85	3	10	5	73	9	38	8	95	6	97	2	17
J10	4	60	8	32	2	95	3	93	1	65	6	85	7	43	9	85	5	46	0	59

Instancia La 21

Lawrence (15x10): 15 trabajos y 10 máquinas

Tabla 61. Datos de la instancia La 21

		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10
J1	2	34	3	55	5	95	9	16	4	21	6	71	0	53	8	52	1	21	7	26
J2	3	39	2	31	0	12	1	42	9	79	8	77	6	77	5	98	4	55	7	66
J3	1	19	0	83	3	34	4	92	6	54	9	79	8	62	5	37	2	64	7	43
J4	4	60	2	87	8	24	5	77	3	69	7	38	1	87	6	41	9	83	0	93
J5	8	79	9	77	2	98	4	96	3	17	0	44	7	43	6	75	1	49	5	25
J6	8	35	7	95	6	9	9	10	2	35	1	7	5	28	4	61	0	95	3	76
J7	4	28	5	59	3	16	9	43	0	46	8	50	6	52	7	27	2	59	1	91
J8	5	9	4	20	2	39	6	54	1	45	7	71	0	87	3	41	9	43	8	14
J9	1	28	5	33	0	78	3	26	2	37	7	8	8	66	6	89	9	42	4	33
J10	2	94	5	84	6	78	9	81	1	74	3	27	8	69	0	69	7	45	4	96
J11	1	31	4	24	0	20	2	17	9	25	8	81	5	76	3	87	7	32	6	18
J12	5	28	9	97	0	58	4	45	6	76	3	99	2	23	1	72	8	90	7	86
J13	5	27	9	48	8	27	7	62	4	98	6	67	3	48	0	42	1	46	2	17
J14	1	12	8	50	0	80	2	50	9	80	3	19	5	28	6	63	4	94	7	98
J15	4	61	3	55	6	37	5	14	2	50	8	79	1	41	9	72	7	18	0	75

Instancia La 22

Lawrence (15x10): 15 trabajos y 10 máquinas

Tabla 62. Datos de la instancia La 22

		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10
J1	9	66	5	91	4	87	2	94	7	21	3	92	1	7	0	12	8	11	6	19
J2	3	13	2	20	4	7	1	14	9	66	0	75	6	77	5	16	7	95	8	7
J3	8	77	7	20	2	34	0	15	9	88	5	89	6	53	3	6	1	45	4	76
J4	3	27	2	74	6	88	4	62	7	52	8	69	5	9	9	98	0	52	1	88
J5	4	88	6	15	1	52	2	61	7	54	0	62	8	59	5	9	3	90	9	5
J6	6	71	0	41	4	38	3	53	7	91	8	68	1	50	5	78	2	23	9	72
J7	3	95	9	36	6	66	5	52	0	45	8	30	4	23	2	25	7	17	1	6
J8	4	65	1	8	8	85	0	71	7	65	6	28	5	88	3	76	9	27	2	95
J9	9	37	1	37	4	28	3	51	8	86	2	9	6	55	0	73	7	51	5	90
J10	3	39	2	15	6	83	9	44	7	53	0	16	4	46	5	24	1	25	8	82
J11	1	72	4	48	0	87	2	66	9	5	6	54	7	39	8	35	5	95	3	60
J12	1	46	3	20	0	97	2	21	9	46	7	37	8	19	4	59	6	34	5	55
J13	5	23	3	25	6	78	1	24	0	28	7	83	8	28	9	5	2	73	4	45
J14	1	37	0	53	7	87	4	38	3	71	5	29	9	12	8	33	6	55	2	12
J15	4	90	8	17	2	49	3	83	1	40	6	23	7	65	9	27	5	7	0	48

Instancia La 23

Lawrence (15x10): 15 trabajos y 10 máquinas

Tabla 63. Datos de la instancia La 23

		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10
J1	7	84	5	58	8	77	2	44	4	97	6	89	3	5	1	58	9	96	0	9
J2	6	21	1	87	4	15	5	39	2	81	3	85	7	31	8	57	9	73	0	77
J3	0	40	5	71	8	34	9	82	3	70	6	22	4	10	7	80	2	48	1	49
J4	5	75	2	17	3	7	6	72	4	11	7	62	8	47	9	35	1	91	0	55
J5	9	20	4	12	6	71	7	67	0	64	2	94	8	15	5	50	3	75	1	90
J6	6	93	5	93	1	57	7	70	8	77	4	58	0	52	2	29	9	7	3	68
J7	7	56	0	95	8	48	4	26	2	82	1	63	9	36	3	27	6	87	5	6
J8	3	76	5	15	9	78	1	8	8	41	2	36	4	30	6	84	0	36	7	76
J9	0	75	7	13	2	81	8	29	4	54	6	82	5	88	1	78	9	40	3	13
J10	2	6	1	26	7	32	6	64	4	54	0	52	5	82	3	6	9	88	8	54
J11	8	62	2	67	5	32	0	62	7	69	3	61	1	35	4	72	9	5	6	93
J12	2	78	9	90	0	85	1	72	8	64	6	63	3	11	7	82	5	88	4	7
J13	4	28	9	11	7	50	6	88	0	44	5	31	2	27	1	66	8	49	3	35
J14	2	14	5	39	6	56	4	62	3	97	9	66	7	69	1	7	8	47	0	76
J15	1	18	8	93	7	58	6	47	3	69	9	57	2	41	5	53	4	79	0	64

Instancia La 24

Lawrence (15x10): 15 trabajos y 10 máquinas

Tabla 64. Datos de la instancia La 24

		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10
J1	7	8	9	75	0	72	6	74	4	30	8	43	2	38	5	98	1	26	3	19
J2	6	19	8	73	3	43	0	23	1	85	4	39	5	13	9	26	2	67	7	9
J3	1	50	3	93	5	80	4	7	0	55	2	61	6	57	8	72	9	42	7	46
J4	1	68	7	43	4	99	6	60	5	68	0	91	8	11	3	96	9	11	2	72
J5	7	84	2	34	8	40	5	7	1	70	6	74	3	12	0	43	9	69	4	30
J6	8	60	0	49	4	59	5	72	9	63	1	69	7	99	6	45	3	27	2	9
J7	6	71	2	91	8	65	1	90	9	98	4	8	7	50	0	75	5	37	3	17
J8	8	62	7	90	5	98	3	31	2	91	4	38	9	72	1	9	0	72	6	49
J9	4	35	0	39	9	74	5	25	7	47	3	52	2	63	8	21	6	35	1	80
J10	9	58	0	5	3	50	8	52	1	88	6	20	2	68	5	24	4	53	7	57
J11	7	99	3	91	4	33	5	19	2	18	6	38	0	24	9	35	1	49	8	9
J12	0	68	3	60	2	77	7	10	8	60	5	15	9	72	1	18	6	90	4	18
J13	9	79	1	60	3	56	6	91	2	40	8	86	7	72	0	80	5	89	4	51
J14	4	10	2	92	5	23	6	46	8	40	7	72	3	6	1	23	0	95	9	34
J15	2	24	5	29	9	49	8	55	0	47	6	77	3	77	7	8	1	28	4	48

Instancia La 25

Lawrence (15x10): 15 trabajos y 10 máquinas

Tabla 65. Datos de la instancia La 25

		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10
J1	8	14	4	75	3	12	2	38	0	76	5	97	9	12	1	29	7	44	6	66
J2	5	38	3	82	2	85	4	58	6	87	9	89	0	43	1	80	7	69	8	92
J3	9	5	1	84	0	43	6	48	4	8	7	7	3	41	5	61	8	66	2	14
J4	2	42	1	8	0	96	5	19	4	59	7	97	9	73	8	43	3	74	6	41
J5	6	55	2	70	3	75	8	42	4	37	7	23	1	48	5	5	9	38	0	7
J6	8	9	2	72	7	31	0	79	5	73	3	95	4	25	6	43	9	60	1	56
J7	0	97	2	64	3	78	5	21	4	94	9	31	8	53	6	16	7	86	1	7
J8	3	86	7	85	9	63	0	61	2	65	4	30	5	32	1	33	8	44	6	59
J9	2	44	3	16	4	11	6	45	1	30	9	84	8	93	0	60	5	61	7	90
J10	7	36	8	31	4	47	6	52	0	32	5	11	2	28	9	35	3	20	1	49
J11	8	20	6	49	7	74	4	10	5	17	3	34	0	85	2	77	9	68	1	84
J12	1	85	5	7	8	71	6	59	4	76	0	17	3	29	2	17	7	48	9	13
J13	2	15	6	87	7	11	1	39	4	39	8	43	0	19	3	32	9	16	5	64
J14	6	32	2	92	5	33	8	82	1	83	7	57	9	99	4	91	3	99	0	8
J15	4	88	7	7	8	27	1	38	3	91	2	69	6	21	9	62	5	39	0	48

Instancia La 31

Lawrence (30x10): 30 trabajos y 10 máquinas

Tabla 66. Datos de la instancia La 31

		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10
J1	4	21	7	26	9	16	2	34	3	55	8	52	5	95	6	71	1	21	0	53
J2	8	77	5	98	1	42	7	66	2	31	3	39	6	77	9	79	4	55	0	12
J3	2	64	4	92	3	34	1	19	8	62	6	54	7	43	0	83	9	79	5	37
J4	0	93	8	24	3	69	7	38	5	77	2	87	4	60	6	41	1	87	9	83
J5	9	77	0	44	4	96	8	79	6	75	2	98	5	25	3	17	7	43	1	49
J6	3	76	2	35	5	28	0	95	7	95	4	61	8	35	1	7	6	9	9	10
J7	1	91	7	27	8	50	3	16	4	28	5	59	6	52	0	46	2	59	9	43
J8	1	45	7	71	2	39	0	87	8	14	6	54	3	41	9	43	5	9	4	20
J9	2	37	3	26	4	33	9	42	0	78	6	89	7	8	8	66	1	28	5	33
J10	1	74	0	69	5	84	3	27	9	81	7	45	8	69	2	94	6	78	4	96
J11	5	76	7	32	6	18	0	20	3	87	2	17	9	25	4	24	1	31	8	81
J12	9	97	8	90	5	28	7	86	0	58	1	72	2	23	6	76	3	99	4	45
J13	9	48	5	27	6	67	7	62	4	98	0	42	1	46	8	27	3	48	2	17
J14	9	80	3	19	5	28	1	12	4	94	6	63	7	98	8	50	0	80	2	50
J15	2	50	1	41	4	61	8	79	5	14	9	72	7	18	3	55	6	37	0	75
J16	9	22	5	57	4	75	2	14	7	65	3	96	1	71	0	47	8	79	6	60
J17	3	32	2	69	4	44	1	31	9	51	0	33	6	34	5	58	7	47	8	58
J18	8	66	7	40	2	17	0	62	9	38	5	8	6	15	3	29	1	44	4	97
J19	3	50	2	58	6	21	4	63	7	57	8	32	5	20	9	87	0	57	1	39
J20	4	20	6	67	1	85	2	90	7	70	0	84	8	30	5	56	3	61	9	15
J21	6	29	0	82	4	18	3	38	7	21	8	50	1	23	5	84	2	45	9	41
J22	3	54	9	37	6	62	5	16	0	52	8	57	4	54	2	38	7	74	1	52
J23	4	79	1	61	8	11	0	81	7	89	6	89	5	57	3	68	9	81	2	30
J24	9	24	1	66	4	32	3	33	8	8	2	20	6	84	0	91	7	55	5	20

J25	3	54	2	64	6	83	9	40	7	8	0	7	4	19	5	56	1	39	8	7
J26	1	6	4	74	0	63	2	64	9	15	6	42	7	98	8	61	5	40	3	91
J27	1	80	3	75	0	26	2	87	9	22	7	39	8	24	4	75	6	44	5	6
J28	5	8	3	79	6	61	1	15	0	12	7	43	8	26	9	22	2	20	4	80
J29	1	36	0	63	7	10	4	22	3	96	5	40	9	5	8	18	6	33	2	62
J30	4	8	8	15	2	64	3	95	1	96	6	38	7	18	9	23	5	64	0	89

Instancia La 32

Lawrence (30x10): 30 trabajos y 10 máquinas

Tabla 67. Datos de la instancia La 32

		t1		**t2**		**t3**		**t4**		**t5**		**t6**		**t7**		**t8**		**t9**		**t10**
J1	6	89	1	58	4	97	2	44	8	77	3	5	0	9	5	58	9	96	7	84
J2	7	31	2	81	9	73	4	15	1	87	5	39	8	57	0	77	3	85	6	21
J3	2	48	5	71	0	40	3	70	1	49	6	22	4	10	8	34	7	80	9	82
J4	4	11	6	72	7	62	0	55	2	17	5	75	3	7	1	91	9	35	8	47
J5	0	64	6	71	4	12	1	90	2	94	3	75	9	20	8	15	5	50	7	67
J6	2	29	6	93	3	68	5	93	1	57	8	77	0	52	9	7	4	58	7	70
J7	4	26	3	27	1	63	5	6	6	87	7	56	8	48	9	36	0	95	2	82
J8	1	8	7	76	3	76	4	30	6	84	9	78	8	41	0	36	2	36	5	15
J9	3	13	8	29	0	75	2	81	1	78	5	88	4	54	9	40	7	13	6	82
J10	0	52	2	6	3	6	5	82	6	64	9	88	8	54	4	54	7	32	1	26
J11	8	62	1	35	4	72	7	69	0	62	5	32	9	5	3	61	2	67	6	93
J12	2	78	3	11	7	82	4	7	1	72	8	64	9	90	0	85	5	88	6	63
J13	7	50	4	28	3	35	1	66	2	27	8	49	9	11	6	88	5	31	0	44
J14	4	62	5	39	0	76	2	14	6	56	3	97	1	7	7	69	9	66	8	47
J15	6	47	2	41	0	64	7	58	9	57	8	93	3	69	5	53	1	18	4	79
J16	7	76	9	81	0	76	6	61	4	77	8	26	2	74	5	22	1	58	3	78
J17	6	30	8	72	3	43	0	65	1	16	4	92	5	95	9	29	2	99	7	64
J18	1	35	3	74	5	16	4	85	0	7	2	81	6	86	8	61	9	35	7	34
J19	1	97	7	43	4	72	6	88	5	17	0	43	8	94	3	64	9	22	2	42
J20	7	99	2	84	8	99	5	98	1	20	6	31	3	74	0	92	9	23	4	89
J21	8	32	0	6	4	55	5	19	9	81	1	81	7	40	6	9	3	37	2	40
J22	6	15	2	70	8	25	1	46	9	65	4	64	7	21	0	77	5	65	3	55
J23	8	31	7	84	5	37	3	24	2	85	4	89	9	29	1	44	0	40	6	83
J24	4	80	0	8	9	41	5	59	7	56	3	38	2	30	8	97	6	77	1	80
J25	9	59	0	91	3	50	8	80	1	17	6	40	2	71	5	56	4	88	7	7
J26	7	36	3	58	4	54	5	77	2	8	6	9	0	45	9	10	1	29	8	96
J27	0	28	3	92	2	73	7	27	8	86	5	87	9	96	1	98	6	99	4	70
J28	9	32	1	95	3	85	6	81	2	41	8	39	7	92	0	59	5	56	4	52
J29	4	93	2	12	5	22	6	27	8	45	7	69	3	60	1	7	0	88	9	49
J30	2	61	5	26	9	71	8	44	0	21	6	82	3	68	7	33	1	84	4	99

Instancia La 35

Lawrence (30x10): 30 trabajos y 10 máquinas

Tabla 68. Datos de la instancia La 35

		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10
J1	0	66	2	84	3	26	7	29	9	94	6	98	8	7	5	98	1	45	4	43
J2	3	32	0	97	6	55	2	88	8	93	9	88	1	20	4	50	7	17	5	5
J3	4	43	3	68	8	47	9	68	1	57	6	20	5	81	2	60	7	94	0	62
J4	1	57	5	40	0	78	6	9	2	49	9	17	3	32	4	30	8	87	7	77
J5	0	52	4	30	3	48	5	48	1	26	9	17	6	93	8	97	7	49	2	89
J6	7	95	0	33	1	5	6	17	5	70	3	57	4	34	2	61	8	62	9	39
J7	7	97	5	92	1	31	8	5	2	79	4	5	3	67	0	5	9	78	6	60
J8	2	79	4	6	7	20	8	45	6	34	3	24	9	26	5	68	1	16	0	46
J9	7	58	9	50	2	19	8	93	6	49	3	25	5	85	4	50	0	93	1	26
J10	9	81	6	71	5	7	1	39	2	16	8	42	0	71	4	84	3	56	7	99
J11	8	9	0	86	9	6	3	71	6	97	5	85	4	16	2	42	7	81	1	81
J12	4	72	3	24	0	30	8	56	2	43	1	61	7	82	6	40	5	59	9	43
J13	9	43	1	13	6	70	7	93	0	95	8	12	4	15	2	78	5	97	3	14
J14	0	14	6	26	1	71	3	46	8	80	5	31	4	37	9	27	7	92	2	67
J15	2	12	0	43	5	96	6	7	3	45	7	20	1	13	9	29	4	60	8	33
J16	1	78	5	50	6	84	0	42	8	84	4	30	9	76	2	57	7	87	3	59
J17	4	49	7	50	1	15	8	13	0	93	6	50	9	32	5	59	3	10	2	35
J18	1	25	0	47	7	60	8	33	4	53	5	37	9	73	2	22	3	87	6	79
J19	0	84	6	83	1	71	5	68	9	89	8	11	3	60	4	50	2	33	7	97
J20	1	14	0	38	6	88	5	5	4	77	7	92	8	24	2	73	9	52	3	71
J21	7	62	9	19	6	38	3	15	8	64	2	64	4	8	1	61	0	19	5	33
J22	2	33	5	46	4	74	0	56	6	84	9	83	8	19	7	8	3	32	1	97
J23	4	50	3	71	6	50	2	97	9	8	0	17	7	19	8	92	5	54	1	52
J24	8	32	1	79	3	97	5	38	9	49	4	76	6	76	0	56	2	78	7	54
J25	5	13	3	5	2	25	0	86	1	95	9	28	6	78	8	24	7	10	4	39
J26	7	48	2	59	0	20	9	7	5	31	6	97	1	89	4	32	3	25	8	41
J27	5	87	0	18	9	48	2	43	1	30	6	97	7	47	8	65	3	69	4	27
J28	6	71	5	20	8	20	1	78	3	39	0	17	7	50	2	44	9	42	4	38
J29	0	50	9	42	3	72	5	7	1	77	7	58	4	78	2	89	6	70	8	36
J30	3	32	9	95	2	13	0	73	6	97	8	24	4	49	5	57	1	68	7	94

Instancia La 38

Lawrence (15x15): 15 trabajos y 15 máquinas

Tabla 69. Datos de la instancia La 38

		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10		t11		t12		t13		t14		t15
J1	1	26	12	67	0	72	6	74	14	13	8	43	4	30	3	19	10	23	11	85	5	98	13	43	2	38	7	8	9	75
J2	14	42	0	39	4	55	12	46	1	19	8	93	9	80	5	26	10	7	6	50	11	57	3	73	2	9	7	61	13	72
J3	3	96	4	99	12	34	6	60	7	43	14	7	13	12	8	11	11	70	10	43	0	91	1	68	9	11	5	68	2	72
J4	14	63	11	45	4	49	1	74	8	27	0	30	9	72	7	9	12	99	13	60	5	69	6	69	2	84	3	40	10	59
J5	2	91	0	75	9	98	3	17	10	72	13	31	11	9	14	98	7	50	5	37	4	8	8	65	1	90	12	91	6	71
J6	11	35	6	80	4	39	3	62	14	74	5	72	10	35	9	25	1	49	8	52	7	63	2	90	13	21	12	47	0	38
J7	14	19	7	57	10	24	13	91	3	50	0	5	11	49	12	18	9	58	5	24	8	52	1	88	2	68	6	20	4	53
J8	7	77	14	72	5	35	11	90	4	68	6	18	3	9	0	33	8	60	10	18	12	10	13	60	1	38	2	99	9	15
J9	13	6	8	86	2	40	9	79	12	92	11	23	5	89	10	95	6	91	7	72	0	80	1	60	3	56	4	51	14	23
J10	1	46	6	28	5	34	11	77	4	47	0	10	14	49	8	77	10	48	7	24	12	8	2	72	13	55	9	29	3	40
J11	10	22	4	89	12	79	0	7	9	15	1	6	11	30	6	38	5	11	8	52	3	20	7	5	14	9	2	20	13	28
J12	5	73	14	56	2	37	3	22	13	25	6	58	1	8	7	93	4	88	8	17	12	9	11	69	10	71	9	85	0	55
J13	9	85	14	58	3	46	8	64	2	49	6	37	1	33	4	30	5	26	0	20	13	74	10	77	12	99	11	56	7	21
J14	10	17	3	24	4	89	5	15	11	60	1	42	8	98	2	64	13	92	0	63	7	52	12	54	6	75	14	23	9	38
J15	3	8	5	17	11	56	7	93	14	26	9	62	6	7	10	88	0	97	1	7	2	43	8	29	13	35	12	87	4	57