MÉTODO PARTICLE SWARM OPTIMIZATION (PSO - ENJAMBRE DE PARTÍCULAS) APLICADO AL PROBLEMA DE MÚLTIPLES OBJETIVOS DEL JOB SHOP SCHEDULING (JSP) O SECUENCIAMIENTO DE MÁQUINAS.

CINDY JOHANNA SARMIENTO ARDILA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE ESTUDIOS INDUSTRIALES Y EMPRESARIALES
BUCARAMANGA
2012

MÉTODO PARTICLE SWARM OPTIMIZATION (PSO - ENJAMBRE DE PARTÍCULAS) APLICADO AL PROBLEMA DE MÚLTIPLES OBJETIVOS DEL JOB SHOP SCHEDULING (JSP) O SECUENCIAMIENTO DE MÁQUINAS.

CINDY JOHANNA SARMIENTO ARDILA

Trabajo de grado para optar al título de Ingeniera Industrial

Director:
Ph.D HENRY LAMOS DÍAZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE ESTUDIOS INDUSTRIALES Y EMPRESARIALES
BUCARAMANGA
2012

DEDICATORIA

"The greatest enemy of knowledge is not ignorance; it is the illusion of knowledge."

"One, remember to look up at the stars and not down at your feet. Two, never give up work. Work gives you meaning and purpose and life is empty without it. Three, if you are lucky enough to find love, remember it is there and don't throw it away."

Stephen Hawking.

AGRADECIMIENTOS

To the ones I love and care …

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

RESÚMEN

TÍTULO: MÉTODO PARTICLE SWARM OPTIMIZATION (PSO) APLICADO AL PROBLEMA DE MÚLTIPLES OBJETIVOS DEL JOB SHOP SCHEDULING (JSP) O SECUENCIAMIENTO DE MÁQUINAS[*]

AUTOR: CINDY JOHANNA SARMIENTO ARDILA[**]

PALABRAS CLAVE: Programación de tareas, taller, combinatorios, metaheurística, enjambre de partículas, múltiples objetivos, problemas de comparación.

DESCRIPCIÓN:
El problema de la programación de tareas en empresas tipo taller (Job Shop Scheduling-JSP) es una rama de la programación productiva que pertenece a la clase de problemas combinatorios de complejidad *NP-Hard*. Se aplica, usualmente, a la optimización de un único objetivo dejando a un lado la optimización de más ellos que, en la mayoría de las veces, suelen contradecirse. En problemas de tamaño medio y grande se presentan dos o más criterios en conflicto, es por esto, que se hace importante considerar múltiples objetivos a la hora de buscar soluciones factibles y poder así, tomar las respectivas decisiones en cuanto a la mejor combinación que permita disminuir costos, tiempos de finalización y cumplir con los tiempos de entrega.

Gran variedad de métodos metaheurísticos han sido aplicados al problema obteniendo resultados aproximados al óptimo. Uno de estos métodos es la optimización por enjambre de partículas, el cual es utilizado usualmente para resolver problemas de optimización continua y en algunos casos para resolver problemas discretos como la programación de tareas.

Este trabajo estudia el problema de la programación de tareas y realiza una revisión bibliográfica de los últimos métodos utilizados en la resolución del problema y se enfoca, principalmente, en el método Particle Swarm Optimization (PSO). Además, presenta de manera clara y profunda el método Multi-objective Particle Swarm Optimization (MOPSO) el cual es una modificación del PSO para aplicarlo al problema de secuenciamiento de máquinas y resolverlo en problemas de comparación.

Se realiza una prueba de escritorio por medio de Excel para una instancia 3x3 del problema para entender el funcionamiento básico del algoritmo y finalmente, se presenta el pseudocódigo para resolver el problema de secuenciamiento de máquinas en empresas tipo taller con múltiples objetivos de optimización.

---

[*] Proyecto de Grado. Modalidad de Trabajo de Investigación.

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Estudios Industriales y Empresariales. Director: PhD. Henry Lamos Díaz.

ABSTRACT

TITLE: PARTICLE SWARM OPTIMIZATION (PSO) METHOD APPLIED TO MULTI-OBJECTIVE JOB SHOP SCHEDULING PROBLEM (JSP) [*]

AUTHOR: CINDY JOHANNA SARMIENTO ARDILA[**]

KEYWORDS: Scheduling, job shop, combinatorial, metaheuristic, particle swarm, multi-objective, benchmark problems.

DESCRIPTION:
Job Shop Scheduling problem is a branch of production scheduling that belongs to combinatorial problems with *NP-Hard* complexity. It is usually applied to single objective optimization leaving aside multi-objective optimization, objectives that in most cases are often contradictory. In medium and large size problems one can find conflicts between multiple criteria for which is so important to considerate them by the time one is seeking feasible solutions and thus take the respective decisions regarding to the best combinations that allows reduce costs, completion times and meet due dates.

Many different metaheuristics approaches have been applied to JSP getting close to optimal results. One of these methods is the Particle Swarm Optimization, generally used to solve continuous optimization problems and in some cases to solve discrete problems such as JSP.

This thesis studies the JSP and performs a literature review of recent methods used to solve JSP and focuses mainly on Particle Swarm Optimization. Also presents in a clear and profound way the Multi-objective Particle Swarm Optimization method which is a modification of PSO to apply into the JSP and to solve benchmark problems.

Test is performed using Excel for a 3x3 instance of JSP to understand the basic operation of the algorithm and finally to present a pseudocode to solve JSP with multiple objectives.

---

[*] Thesis Degree. Research Mode.

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Estudios Industriales y Empresariales. Thesis director: PhD. Henry Lamos Díaz.

INTRODUCCIÓN

La programación de tareas (*scheduling*) es un procedimiento que permite optimizar uno o más objetivos, de tal manera que sea más eficiente el proceso productivo de las empresas, así mismo, debe estar acompañado de la toma de decisiones, las cuales conllevan a la obtención de mejores resultados. Este es uno de los temas más importantes, a veces poco tratados, en la mayoría de sistemas de producción y manufactura que pueden convertirse en parte estratégica de la empresa y por consiguiente en una ventaja competitiva, por lo tanto contar con una buena programación que permita tomar decisiones respecto a cuál es la mejor combinación de máquinas y trabajos en el momento de producir, permitirá disminuir costos, tiempos de espera y tiempos de finalización.

Se da el caso de empresas que por su rigidez no cambian los esquemas y terminan imponiendo criterios sin tener en cuenta las exigencias y gustos de los consumidores. No obstante, las hay, que modifican sus estrategias en cuanto a la personalización de los productos y que han aumentado, tal como sucede en las empresas tipo *Job Shop* (Taller) en el mundo.

El *Job Shop* básicamente es una unidad de producción en la cual las cantidades a producir son pequeñas, los requerimientos del proceso varían según el pedido y éstos se realizan simultáneamente usando recursos compartidos. Posee proyectos que requieren varios trabajos con relaciones de precedencia y que requieren recursos de capacidad finita para realizarlo.

La programación de trabajos aplicada a empresas tipo *Job Shop* es compleja en la medida en que aumenta el número de trabajos y de máquinas por las cuales se procesa un producto.

En el *Job Shop*, un trabajo pasa por una secuencia de nodos denominados Centros de Trabajo a través de una ruta especificada y puede esperar por el recurso requerido mientras éste se libera. El tiempo de espera total, del trabajo en el proceso, constituye una gran parte del proceso de producción y que en la mayoría de los casos se pretende disminuir.

La gestión en el *Job Shop* implica determinar las fechas de inicio y de finalización para las órdenes de trabajo con diferentes rutas, planear el material para cada orden y la programación de la producción en general. Todas estas funciones son interdependientes y no se pueden realizar cada una por separado[1].

El *Job Shop Scheduling* - JSP (Programación de tareas-Secuenciamiento de máquinas) se usa para determinar la secuencia en la que se deben realizar las operaciones de un conjunto de trabajos en las máquinas correspondientes. Ha recibido gran atención debido al potencial de disminuir los costos e incrementar el rendimiento y por los beneficios que se derivan de su solución.

El presente proyecto pretende dar una visión más clara sobre el JSP, sobre los diferentes métodos para resolver el JSP y específicamente, se profundiza en el método de Optimización por Enjambre de Partículas (*Particle Swarm Optimization – PSO*) aplicado al JSP con la optimización de dos objetivos.

El algoritmo PSO presentado por Sha y Lin (2010), llamado MOPSO, en el cual se basa este proyecto consiste en resolver el problema del JSP con múltiples objetivos, incluyendo la minimización del *makespan* (tiempo de salida del último trabajo del taller) y el *tiempo de tardanza total*[2]. Para que el PSO (de espacio continuo) sea adecuado para el JSP (de espacio discreto) los autores Sha y Lin modifican la representación, el movimiento y la velocidad de la partícula para

[1] VELAGA, Prasad, Ph.D. Optisol. College Station, TX. Disponible en: http://www.optisol.biz/job_shop_scheduling.html.
[2] Conceptos en los que se profundizará más adelante.

obtener así los mejores resultados para el JSP. En el método MOPSO se requiere de un amplio conocimiento en aspectos conceptuales relacionados con el problema específico al cual se vaya a aplicar.

El presente proyecto se encuentra estructurado de la siguiente manera: En el Capítulo 1 se encuentran las especificaciones generales de la investigación propuesta. En el Capítulo 2, se presenta una revisión de la literatura sobre los últimos métodos que han sido aplicados al problema del JSP. En el Capítulo 3, se presenta la descripción específica del JSP y los conceptos más importantes asociados al mismo.

En el Capítulo 5, se describe el PSO en su forma tradicional y en el Capítulo 6, se describe la aplicación del PSO en el JSP, se complementa la explicación del algoritmo con la resolución de una prueba de escritorio para una instancia 3x3 del JSP, el cual se encuentra en el Anexo A. En el Capítulo 7 se encuentra el pseudocódigo para el JSP, en el Capítulo 9 se encuentra información relacionada a los *benchmark problems*. Finalmente, en el Capítulo 10 y 11 se presentan las conclusiones y recomendaciones.

1. ESPECIFICACIONES GENERALES DEL PROYECTO

1.1 TITULO

MÉTODO PARTICLE SWARM OPTIMIZATION (PSO - OPTIMIZACIÓN POR ENJAMBRE DE PARTÍCULAS) APLICADO AL PROBLEMA DE MÚLTIPLES OBJETIVOS DEL *JOB SHOP SCHEDULING (JSP)* O SECUENCIAMIENTO DE MÁQUINAS)

1.2 MODALIDAD

Proyecto de Investigación.

1.3 RESPONSABLES

1.3.1 Autor del Proyecto.

CINDY JOHANNA SARMIENTO ARDILA
Estudiante de Ingeniería Industrial
Código: 2062574

1.3.3 Director del Proyecto.

Ph.D HENRY LAMOS DÍAZ
Docente Escuela de Estudios Industriales y Empresariales.

1.4 DESCRIPCIÓN DEL GRUPO DE INVESTIGACIÓN

1.4.1 Ópalo.

GRUPO DE OPTIMIZACIÓN Y ORGANIZACIÓN DE SISTEMAS PRODUCTIVOS, ADMINISTRATIVOS Y LOGÍSTICOS – OPALO de la Universidad Industrial de Santander.

El cual centra su investigación en los procesos de modelamiento para resolver problemas de Ingeniería Industrial. Creado en el año 2005 y liderado por el profesor Néstor Raúl Ortiz Pimiento[3].

Los objetivos del grupo son los siguientes:

- Fomentar y consolidar la labor investigativa de la Universidad Industrial de Santander, promoviendo la participación de docentes y estudiantes en el desarrollo de actividades de carácter científico y tecnológico.
- Contribuir al fortalecimiento de la imagen de responsabilidad, seriedad y compromiso académico y social de la Universidad Industrial de Santander.

El plan de trabajo es:

- Participar en las Convocatorias de Entidades que apoyen y financien proyectos de investigación.
- Realizar estudios basados en la experiencia empresarial, tendencias tecnológicas, benchmarking, entre otras, de tal forma que sea posible identificar temáticas de investigación que apoyen el mejoramiento de la productividad y competitividad de las empresas.

[3] Información disponible en el portal de Ingeniería Industrial: carpintero.uis.edu.co

OPALO maneja diferentes líneas de investigación, las cuales se presentan a continuación:

Nombre de la línea: Optimización de Sistemas Productivos.
Objetivo de la línea: Desarrollar proyectos de investigación orientados al estudio, análisis y aprovechamiento óptimo de los recursos productivos de empresa generadoras de bienes o servicios.

Nombre de la línea: Optimización de procesos Administrativos.
Objetivo de la línea: Desarrollar proyectos de investigación que contribuyan a elevar la productividad y competitividad de la empresa en el área administrativa y financiera.

El grupo OPALO tiene como su visión que para el año 2014 será reconocido como uno de los mejores grupos de investigación de la Universidad Industrial de Santander por su excelente producción científica y su constante labor de apoyo al sector empresarial de la región santandereana.

1.5 PLANTEAMIENTO DEL PROBLEMA

Las investigaciones sobre el problema del JSP se basan, usualmente, en optimizar un solo criterio (p. ej. *Makespan*). Sin embargo, en el mundo productivo actual existe la necesidad de optimizar más de 2 objetivos simultáneamente (p. ej. minimizar el *makespan* y la demora total) que en muchos casos son contradictorios.

El problema del JSP consiste en determinar la programación más eficiente para los trabajos que son procesados en varias máquinas, cada trabajo tiene un conjunto de operaciones que son procesadas en éstas en un orden específico.

Para m máquinas y n trabajos, el tamaño del espacio solución es igual a la fórmula $(n!)^m$. Para un problema de tamaño $20x10$ se pueden tener al menos $7.2651x10^{183}$ posibles soluciones pero se considera poco práctico el hecho de tomar en cuenta todas estas posibilidades para identificar las programaciones factibles y finalmente la solución óptima. Debido a esta explosión factorial el JSP es considerado un miembro de una gran clase de problemas numéricos insolubles conocidos como *NP-Hard* (*Non-deterministic polynomial*)[4]. Donde el espacio solución y el tiempo computacional crecen exponencialmente con el tamaño del problema. Si x es el tamaño de entrada y y es una constante entonces, los problemas con un requerimiento de tiempo de $O(x^y)$ tienen una complejidad polinómica denotada por P. Pero si la complejidad del problema es $O(y^x)$ éste es considerado como *NP-Hard* (si $P \neq NP$). Entonces para obtener un algoritmo óptimo se requieren unos pasos computacionales que crezcan exponencialmente con las entradas. Esto concluye que el JSP es uno de los problemas más complejos de la optimización combinatoria.

Debido a que los métodos exactos tienen limitaciones para la solución del JSP se hace necesario el uso de nuevos enfoques. El PSO es una de las técnicas evolutivas más recientes que se usa para el problema del JSP. Ha sido usado en diferentes campos, fundamentalmente, para la solución de problemas de optimización continua, recientemente se usa para la solución de problemas combinatorios.

La investigación se centra en entender el funcionamiento del método PSO en una versión modificada para aplicarlo al problema del JSP con dos objetivos en un tamaño de instancias pequeño.

[4] JAIN, A.S y MEERAN, S. Deterministic Job Shop Scheduling: Past, Present and Future. En: European Journal of Operational Research, 1999, Vol. 113, pp. 390-434.

1.6 ALCANCE DEL TRABAJO

El propósito de la investigación es profundizar en la metaheurística PSO adaptada al JSP y presentar un pseudocódigo en un contexto multiobjetivo en aras de ampliar el conocimiento sobre este tipo de problemas y métodos de solución en la Escuela de Estudios Industriales y Empresariales de la UIS.

Los resultados que se plantea entregar mediante la realización de esta investigación son:

- Un pseudocódigo del método PSO aplicado al problema de múltiples objetivos del JSP con un tamaño de instancias pequeño.
- Un artículo académico publicable sobre el tema expuesto.
- Una biblioteca con la recopilación de problemas de *benchmark.*

1.7 OBJETIVOS

1.7.1 Objetivo General.

Desarrollar un enfoque basado en el método PSO para la solución del JSP con dos objetivos.

1.7.2 Objetivos Específicos.

- Aplicar el algoritmo MOPSO para la solución del problema JSP con múltiples objetivos.
- Resolver el JSP con dos objetivos en un tamaño de instancias pequeño.
- Diseñar en pseudocódigo el algoritmo PSO para su posterior implementación.

- Documentar problemas de *benchmark* para el JSP para la creación de una biblioteca disponible para futuras investigaciones.

1.8 METODOLOGÍA

El proyecto se va realizar siguiendo las siguientes etapas que garantizan el logro de los objetivos propuestos.

Etapa Inicial: Revisión bibliográfica sobre los métodos de solución del JSP.
Está dedicada al estudio y clasificación de la literatura existente sobre algoritmos para la solución del problema JSP.

Etapa Intermedia: Comprensión de las bases teóricas del algoritmo PSO para la solución de problemas de optimización combinatoria.
Estudio del algoritmo para comprender cómo se adapta del espacio solución continuo al espacio discreto.
Clasificación de los problemas de *benchmark* encontrados para el JSP e iniciar la biblioteca.

Etapa Final: Construcción del algoritmo en pseudocódigo del PSO aplicado al JSP con múltiples objetivos.
Creación de la biblioteca con la respectiva clasificación de los problemas de *Benchmark* para el JSP.

1.9 ENTIDADES INTERESADAS

El trabajo es punto de partida para futuras investigaciones en el grupo OPALO, debido a que el tema que se va a analizar pretende resolver uno de los problemas

más complejos de las industrias. También es de fuerte interés para la autora del proyecto ya que es requisito primordial de grado, así mismo, para el Director del proyecto para complementar y ampliar su desarrollo profesional y académico dentro de la Universidad. Además, para los investigadores que estén interesados en ésta área y que deseen expandir sus conocimientos.

2. ESTADO DEL ARTE

En el presente capítulo se hace un recuento de los métodos más utilizados para resolver el problema del JSP. Cada uno con diferentes variaciones, criterios de optimización y mejoras que permiten obtener resultados aceptados como óptimos para el JSP.

Según Jain (1999) se puede declarar a Johnson (1954) como el que inició la investigación del JSP con el primer estudio dentro de la Teoría del *Scheduling*, un algoritmo óptimo para dos máquinas en un ambiente *flow-shop*. Jain indica que no se confirma con seguridad quién propuso el problema del JSP en su forma actual pero entre las primeras investigaciones se encuentran: Jackson (1956) quien generaliza el algoritmo de Johnson para el *Job Shop*, Roy y Sussman (1964) presentan la representación mediante el Grafo Disyuntivo, Balas (1969) aplica un enfoque enumerativo basado en el Grafo, y Giffler y Thompson (Algoritmo G&T - 1969) proponen un algoritmo voraz que genera programaciones activas en nxm iteraciones,

Inicialmente, el enfoque para resolver los problemas estaba en el uso de metodologías heurísticas un poco gruesas pero efectivas, que fueron la base para la evolución de la Teoría del *Scheduling* clásico. Durante los años 60, hubo un cambio en el enfoque al usar algoritmos enumerativos con formulaciones

matemáticas más sofisticadas y elaboradas. Durante los años 70 y hasta mediados de los 80, se trató de justificar la complejidad del problema. A mediados de los 70's, Garey (1976) demostró que el problema del JSP es de tipo *NP-Hard* cuando *m>2*. Al final de los 80, los métodos se enfocan en aplicar técnicas de aproximación. En 1989, los métodos evolutivos se usan para resolver problemas difíciles, los cuales están inspirados por fenómenos naturales e inteligentes. En 1990, aparecen los métodos metaheurísticos. Actualmente, se hace uso de métodos híbridos que resaltan las capacidades de dos o más metaheurísticas para resolver eficientemente los problemas de tipo combinatorio. Hasta el momento, el JSP sigue causando gran interés entre los investigadores de ingeniería industrial y la OR (*Operational Research* - Investigación de Operaciones) por sus ventajas y complejidad.

Se presenta en forma resumida algunas de las investigaciones que intentan resolver el JSP desde el 2005 a la fecha:

Investigaciones interesadas en resolver el JSP con un solo objetivo:

- Mattfeld y Bierwirth (2004) proponen un Algoritmo Genético (GA) para el JSP con fechas de liberación y vencimiento y con la minimización de la tardanza como objetivo. Muestran que una reducción al espacio de búsqueda puede ayudar al algoritmo a encontrar mejores soluciones en un tiempo computacional menor.
- Huang y Liao (2008) presentan un híbrido entre la Optimización por Colonia de Hormigas (*Ant Colony Optimization-ACO*) y la Búsqueda Tabú (TS) para el JSP llamado ACOFT. El algoritmo emplea un método novedoso de descomposición inspirado por el SBP (*Shifting Bottleneck Procedure*) y un mecanismo de re-optimizaciones ocasionales de los *schedules* parciales. Se usa TS para mejorar la calidad de la solución generada.
- Q. Niu *et al* (2008), redefinen y modifican el PSO al introducir operadores genéticos, como el *crossover* y el operador de mutación, para actualizar las partículas llamado GPSO (*Particle Swarm Optimization Combined With Genetic*

Operators) y con tiempos de procesamientos *fuzzy*. Los autores demostraron que el método provee mejores resultados que GA.

- Zhang *et al* (2008), proponen usar un enfoque llamado (TSSA); la idea principal de este es usar el Recocido Simulado (SA) para encontrar soluciones élite dentro del Gran Valle (BV-Big Valley) para que TS pueda re-intensificar la búsqueda de las soluciones más prometedoras. Tiene como objetivo la minimización del makespan. Puede obtener la solución en un tiempo computacional aceptable y es un algoritmo robusto y eficiente para el JSP.
- Zhang y Wu (2010) proponen un híbrido de Recocido Simulado (SA) basado en un mecanismo novedoso inmune con el objetivo de minimizar la tardanza ponderada total. Diseñan un sistema de inferencia difuso (fuzzy) para evaluar el nivel del cuello de botella. Es un método eficiente y efectivo para resolver el JSP. Además, puede considerar problemas con fechas de vencimiento (due dates).
- Asadzadeh y Zamanifar (2010) proponen un enfoque Paralelo Basado en Agentes para el problema del JSP donde la población inicial y la paralelización del GA se hacen a través del Enfoque Basado en Agentes. Los resultados indican que el método paralelo mejora la calidad de las soluciones obtenidas.
- Surekha y Sumathi (2011) presentan el *Fuzzy Genetic Swarm Optimization*, un híbrido entre *Genetic Algorithms* (Gas) y *Particle Swarm Optimization* (PSO) el cual, en cada iteración, cambia algunos individuos por unos nuevos a través del GA. El conjunto de soluciones resultantes se mueve al espacio solución por el PSO. Según sus autores presenta una buena solución al problema del JSP.
- Gao *et al* (2011), proponen un Algoritmo Memético (MA) eficiente con una novedosa mejora a la búsqueda local para resolver el JSP con la minimización del *makespan*. En la búsqueda local realizan un cambio sistemático del vecindario para evitar quedar atrapado en un óptimo local mejorando así la calidad de la solución. Diseñan dos estructuras de vecindario al realizar intercambios e inserciones basadas en el camino crítico. Demostraron que la eficiencia del método es superior a otros reportados en la literatura.

- Sels *et al* (2011), presentan un híbrido entre un Algoritmo Genético y la Búsqueda Dispersa (SS). Extienden la población de una sola a una doble, al tomar en cuenta las características específicas del problema, para agregar diversidad en el proceso de búsqueda. Sus resultados computacionales comprueban los beneficios de la diversidad en la efectividad del proceso de búsqueda.
- Mati *et al* (2011), presentan un método de búsqueda local que usa un modelo de un grafo disyuntivo y vecindarios generados al cambiar los arcos críticos. Proponen un método eficiente para evaluar los movimientos que está basado en una nueva función de evaluación -que permite estimar el valor generado al hacer un intercambio sin tener que hacerlo realmente-. El método puede ser usado para cualquier criterio de optimización y sólo necesita modificar el rango de un parámetro. Según sus autores es el primer método de búsqueda local que optimiza cualquier criterio.
- Bülbül (2011) propone un híbrido entre el *Shifting Bottleneck* (SB) y TS al reemplazar el paso de re-optimización del SB por el TS con el objetivo de minimizar la tardanza ponderada total.
- Li y Chen (2011) introducen un algoritmo computacionalmente efectivo del Proceso en Equipo (*Team Process Algorithm-TPA*) para resolver el JSP con minimización del *makespan*. Provee buenas soluciones en tiempos razonables.
- Qing-dao-er-ji y Wang (2012) proponen un híbrido de Algoritmo Genético con base a un nuevo operador *crossover* basado en la máquina y un operador de mutación basado en el camino crítico. Presentan un nuevo algoritmo para encontrar el camino crítico de un *schedule*. Además, introducen un operador de búsqueda local para mejorar la habilidad de búsqueda local de GA. Es un método efectivo y se desempeña mejor que los métodos con los que se comparó.

Investigaciones interesadas en resolver el JSP con múltiples objetivos:

- Tavakkoli-Moghaddam *et al* (2011), presentan un nuevo modelo matemático para un problema con dos objetivos del JSP con tiempos de preparación y de inicio (*ready-times*) que dependen de la secuencia y que

minimiza el tiempo ponderado de flujo promedio y la suma de los costos ponderados de demora y adelanto. Proponen un nuevo algoritmo PSO combinado con operadores genéticos como la búsqueda de vecindario variable (*Variable Neighborhood Search*-VNS). Usan un elemento de la Búsqueda Dispersa (SS) para seleccionar un nuevo enjambre, en cada iteración, para encontrar soluciones de Pareto para el JSP. Supera los resultados encontrados individualmente por los métodos al mejorar la calidad de las soluciones encontradas y es apto para problemas de tamaño real.

- Nawaz (2007) propone un enfoque para JSP con múltiples objetivos llamado el Algoritmo Genético con Genes "Saltarines" (*Jumping Genes Genetic Algorithm*-JGGA). Este enfoque es capaz de mantener su consistencia y converger a las soluciones no-dominadas. Extienden el híbrido para aliviar la diversidad adicional generada por las operaciones saltarinas que se introducen en el JGGA. Este método puede buscar las soluciones no-dominadas con una mejor convergencia y optimizar múltiples criterios simultáneamente.
- Suresh y Mohanasundaram (2006) estudian el JSP con la minimización del makespan y el tiempo promedio de flujo de los trabajos. Proponen la metaheurística llamada *Pareto Archived Simulated Annealing* (PASA) para descubrir los conjuntos de soluciones no-dominadas para el JSP. Usan un nuevo mecanismo de perturbación llamado el *Segment-Random Insertion* (SRI) para generar un conjunto de soluciones vecinas a la solución actual.

Para el enfoque de la presente investigación se hace un recuento de los artículos donde se aplica el PSO al JSP. Dado a que el inconveniente principal del PSO para aplicarlo a problemas combinatorios es su naturaleza continua -comparada con la naturaleza discreta del JSP[5]- se encuentran investigaciones que presentan distintas modificaciones de las cuales se destacan:

[5] YEN, G. G y IVERS, B. Job Shop Scheduling Optimization through Multiple Independent Particle Swarms. En: International Journal of Intelligent Computing and Cybernetics, 2009, Vol. 2, pp. 5-33.

- Z. Lian *et al* (2006), aplican un PSO similar al JSP para minimizar el makespan y proponen nuevos operadores demostrando una buena calidad en un tiempo razonable para problemas de instancias pequeñas.
- Lei (2008) presenta el PSO aplicado al problema de objetivos múltiples en JSP, propone un método para convertir el JSP a naturaleza continua. Genera soluciones de Pareto con una mejor calidad que indican que este método es más simple y fácil de implementar que manejar el PSO en forma discreta. El objetivo es minimizar simultáneamente el *makespan* y la tardanza total convirtiendo el JSP en naturaleza continua al usar el método de representación basado en reglas de prioridad.
- Yen e Ivers (2009) examinan la optimización del JSP a través de un esquema de División Del Espacio De Búsqueda al asignar a cada partícula una población independiente de partículas. El uso de múltiples enjambres se basa en la idea de "divide y vencerás" para reducir la complejidad computacional. El método necesita pocas evaluaciones de la función objetivo para encontrar el Schedule óptimo y soluciona efectivamente los problemas de JSP para instancias de tamaño pequeño.
- Sha y Lin (2010) varían el PSO al modificar la representación de la posición, el movimiento y la velocidad de la partícula y lo aplican al JSP con múltiples objetivos. Los resultados demuestran que presenta un mejor desempeño en la calidad de la búsqueda y eficiencia que la mayoría de las técnicas evolutivas, motivo por el cual se escoge para el desarrollo de la presente investigación -el cual se extenderá más adelante-.
- Lin *et al* (2010), proponen un nuevo híbrido de la Inteligencia de Enjambre (SI) que consiste en el PSO, SA y un esquema de mejora individual multi-tipo para resolver el JSP, llamado MPSO. MPSO adopta un espacio real como el espacio de búsqueda llamado Espacio *Random-Key* (RK). En este espacio la posición de la partícula se compone de nxm números reales que representan la permutación de todas las operaciones, de todos los trabajos, por el esquema de codificación. Es un algoritmo más robusto y eficiente que los algoritmos existentes.

Se puede observar que la mayoría de las técnicas utilizadas en los últimos años para el JSP es la implementación de algoritmos *híbridos*. Es decir, la combinación de dos o más técnicas para lograr mejores resultados que los alcanzados al usar sólo un método de solución.

Además, de las investigaciones nombradas anteriormente se puede deducir que el JSP está evolucionando, cada vez más, debido al desarrollo de métodos novedosos que están generando muy buenos resultados. En la actualidad, la mayoría de las investigaciones tratan de resolver instancias de tipo pequeño y mediano. El rumbo de la investigación del JSP se dirige hacia la solución de problemas de tamaño real (un gran tamaño) ya que estos son los problemas que se encuentran normalmente en las industrias del mundo actual.

3. OPTIMIZACIÓN COMBINATORIA

La optimización combinatoria es una rama de la matemática aplicada y de la ciencia de la computación cuyo objetivo es encontrar el mínimo (máximo) de una función dada sobre un conjunto finito de soluciones. El dominio de la optimización combinatoria se compone de problemas de optimización donde las posibles soluciones son discretas o se pueden reducir a un conjunto discreto. La investigación de operaciones, la teoría de los algoritmos y la teoría de la complejidad computacional son campos que están estrechamente relacionados con la optimización combinatoria.

Los problemas de optimización combinatoria se pueden modelar mediante modelos matemáticos. Un modelo de optimización consta de una función objetivo y una serie de restricciones que dependen de un conjunto de variables de decisión.

"Un problema de optimización combinatoria consta de la pareja $P = (S, f)$ donde:

Para el conjunto de variables $X = \{x_1, ..., x_n\}$ cuyo dominio pertenece a $D_1, ...D_n$; D_i conjuntos finitos y que pertenecen a Z^+.

Se define una función objetivo f a minimizar, esto es donde, $f : D_1 \times ... \times D_n \rightarrow R^+$.

y el problema consiste en la minimización de $f(x)$.

El conjunto de todas las posibles asignaciones factibles es:

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

S es el espacio de búsqueda ya que cada elemento del conjunto puede ser un candidato solución.

El problema consiste en encontrar una solución $s^* \in S$ que dé el valor mínimo de la función objetivo, es decir, $f(s^*) \leq f(s) \quad \forall\ s \in S$. Donde, el conjunto $S^* \subseteq S$ se llama el conjunto de soluciones óptimas globales[6]".

Para encontrar una solución que satisfaga las características del problema construye un algoritmo que paso a paso halla una solución mejor que la anterior. Un *algoritmo* es una secuencia de instrucciones que representan un modelo solución para un problema. Una *instancia* del problema es un caso particular de éste con valores definidos. Por ejemplo, para el JSP la dimensión de una instancia del problema es $n \times m$ donde, n es el número de trabajos y m el número de máquinas. Los algoritmos de optimización combinatoria resuelven instancias de problemas *NP-hard* al explorar el espacio de soluciones y reducir el tamaño efectivo de éste y explorarlo de nuevo de manera eficiente.

La complejidad de un algoritmo viene dada por la cantidad de operaciones que se efectúan en el peor caso y está dada en función del tamaño de los datos de entrada. La complejidad del problema viene dada por la complejidad del algoritmo

[6] BLUM, Christian y ROLI, Andrea. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. En: ACM Computing Surveys, September, 2003, Vol. 35 No. 3, pp. 268–308.

más eficiente que lo resuelva. Evaluar la eficiencia de un algoritmo se relaciona con evaluar la complejidad del mismo.

Complejidad Computacional: La teoría de la complejidad estudia los recursos requeridos durante el cálculo para resolver un problema. Si un cálculo es difícil de realizar tiende a ser complejo y se necesitan más recursos. Los recursos son el tiempo -que se define como el número de pasos que necesita el algoritmo para resolver un problema- y el espacio -que se define como la cantidad de memoria utilizada para resolverlo-.

La teoría de la complejidad divide el conjunto de problemas en clases según el límite superior del número de operaciones en el peor caso:

- **Clase** P (Polinomial): Conjunto de problemas de decisión que pueden ser resueltos en tiempo polinomial respecto al tamaño de la instancia, por una maquina secuencial determinista (Turing[7]).

- **Clase** NP (No-determinista en tiempo polinomial): Conjunto de problemas de decisión donde puede verificarse la veracidad de una solución en tiempo polinomial por una maquina secuencial no-determinista[8] (algoritmo no-determinista). El algoritmo consiste en dos fases: “adivinar y verificar”. En tiempo polinomial, la fase “adivinar” genera una posible solución al problema llamado Certificado. La fase “verificar” prueba si el certificado se ajusta a los

[7] La máquina de Turing es una entidad matemática abstracta con la que se demostró que existen problemas que no se pueden solucionar con ninguna máquina de turing. La máquina puede resolver cualquier problema enumerable que equivale a un problema que se puede solucionar por un ordenador digital. Aplicado en la teoría de la complejidad, puede comparar la dificultad de los diferentes métodos para resolver un problema. Se llama Máquina de Turing Determinista cuando existe sólo una posibilidad de ejecución.

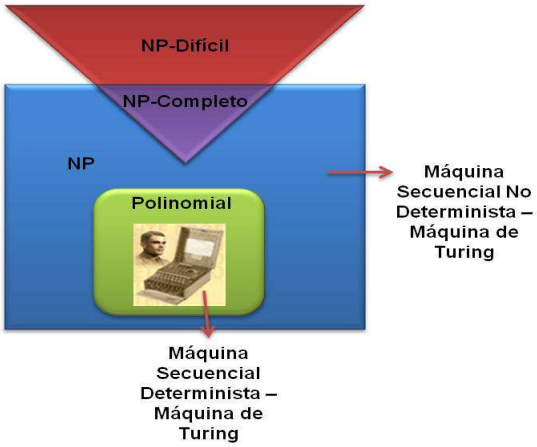
[8] En el caso de que exista más de una posible combinación de ejecuciones se trata de una Máquina de Turing No-determinista.

requerimientos de una solución correcta y entrega un SI o un NO también en un tiempo polinomial. El algoritmo no-determinista corre hasta que entregue un Sí[9].

NP-Complete: Un problema de decisión *Q* en *NP* es *NP-Completo*, si cualquier $P \in NP$ entonces[10] $P \rightarrow Q$. Es decir, problemas que son muy difíciles de resolver.

NP-Hard (al menos tan complejo como NP): La idea intuitiva de un problema "difícil de resolver" queda reflejada en el término científico NP-hard utilizado en el contexto de la complejidad algorítmica. En términos coloquiales, se puede decir que un problema de optimización difícil es aquel para el que no se puede garantizar que se encuentre la mejor solución posible en un tiempo razonable[11].

Figura 1. Diagrama de las clases P, NP, NP-Completo y NP-Difícil



Fuente: Autor

El diagrama que se presenta en la Figura 1 representa cada conjunto mediante una figura. En este diagrama el conjunto P está contenido dentro de los problemas

[9] GOWER, M. WILKERSON, R. R-by-C Crozzle: An NP-Hard Problem. Tesis Maestría, Department of Computer Science, University of Missouri – Rolla.
[10] BRUCKER, Peter. Scheduling Algorithms. Fifth Edition, Germany, Springer, 2007, p. 367.
[11] MARTÍ, Rafael. Procedimientos Heurísticos en Optimización Combinatoria. (Tesis Pregrado) España, Universidad de Valencia, Departamento de Estadística e Investigación Operativa, p. 60.

NP. La intersección resultante entre el conjunto NP y NP-difícil se llama NP-Completo.

Los algoritmos de optimización combinatoria se relacionan con problemas tipo *NP-Hard*. Entre los problemas de optimización combinatoria se encuentra el JSP.

3.1 JOB SHOP SCHEDULING (JSP)

El *scheduling* (*) es la asignación de recursos en un tiempo determinado para realizar un grupo predeterminado de trabajos. El problema de *scheduling* es un problema de toma de decisiones que constituye uno de los problemas más importantes en gestión de la producción tanto desde el punto de vista teórico como práctico[12].

A continuación se presenta una serie de conceptos necesarios para el desarrollo de la presente investigación sobre el JSP:

a) Trabajo: Conjunto de operaciones que se realizan en un conjunto de máquinas. Se identifica como el producto a entregar.

b) Operación: Es el procesamiento de un trabajo en una máquina. Cada trabajo está compuesto por una o más operaciones. Si las operaciones pertenecen al mismo trabajo tienen una secuencia en la que deben ejecutarse (restricción de precedencia).

c) Máquina: Medio donde se realiza una operación. Una máquina no puede realizar varias operaciones a la vez (restricciones de capacidad/recurso).

d) Objetivos: Los criterios más comunes con los que se valoran los objetivos son, el *makespan* $C_{\max}$ (el máximo valor de los tiempos de finalización C_j de todos los

(*) A partir de aquí se usará la palabra en inglés ya que su traducción tiene varios significados (programación, secuenciamiento).

[12] Ibid.

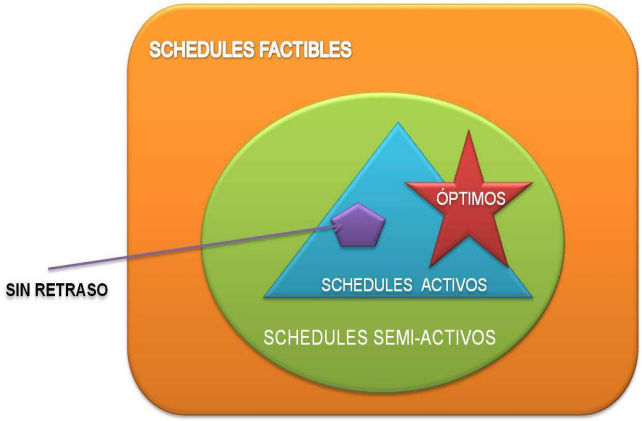
n trabajos), el *tiempo de tardanza máxima de los trabajos* (el máximo valor, entre todos los trabajos, la diferencia entre $C_j - d_j$, d_j "*due date"* es la fecha de entrega), el *tiempo de retraso* $T_{\max}$(es el máximo valor positivo de $C_j - d_j$ de todos los trabajos) y el *tiempo total de terminación* $\sum_1^n C_j$ (suma de los tiempos de terminación de los trabajos), entre otros.

e) Clases de *schedule*: Define la secuencia en que las operaciones utilizan las máquinas. Se clasifican así:

- *Schedule* activo: Un Schedule factible es activo si al alterar la secuencia de procesamiento no se puede adelantar la finalización de una operación sin afectar el tiempo de inicio de la otra operación.
- *Schedule* Semi-activo: En este tipo de programación ninguna operación puede iniciar más temprano sin cambiar la secuencia operativa de cualquier máquina. La siguiente operación se asigna en el momento más temprano posible. Esta programación en general no es óptima ya que los *schedules* óptimos se encuentran, usualmente, en el espacio de la programación activa. Por esto, se recurre al procedimiento "Permissible Left Shifting" (Cambios hacia la Izquierda). Para modificar la programación semi-activa en una activa se cambia la secuencia de dos (2) operaciones adyacentes, siempre y cuando no viole ninguna restricción de precedencia o cause una demora en alguna de las secuencias de la máquina. Cuando se hacen todos los cambios permitidos se obtiene una programación activa.
- *Schedule* sin retraso (non-delay): En esta programación, las máquinas no pueden mantenerse ociosas si hay alguna operación sin procesarse.

En la Figura 2 se muestra la relación entre los espacios de programación y se observa que los *schedules* óptimos se encuentran en los *schedules* activos, principalmente, y pueden estar en los *schedules* semi-activos.

Figura 2. Relación entre los espacios de programación



Fuente: SIERRA, María Rita.

A continuación, se presenta un ejemplo para identificar los *schedules* anteriormente enunciados.

Ejemplo 1. Se tiene una instancia de tamaño 3x3 - 3 trabajos y 3 máquinas -. Cada trabajo tiene una secuencia correspondiente y unos tiempos de procesamiento que se definen de la siguiente forma:

- El trabajo 1 (J1-Job 1) debe pasar por la máquina 1, con un tiempo de procesamiento de 3 unidades de tiempo, luego por la máquina 2 con un tiempo de procesamiento de 5 y por último, por la máquina 3 con un tiempo de procesamiento de 2 unidades de tiempo.
- El trabajo 2 (J2) debe pasar por las máquinas 1, 3 y 2 con procesamiento de 5, 1 y 4 unidades de tiempo respectivamente.
- El trabajo 3 (J3) debe pasar por las máquinas 2,1 y 3 con procesamiento de 4, 2 y 1 unidades de tiempo respectivamente.

La información anterior se organiza en la Tabla 1. En la primera columna se enuncia el trabajo. Las filas corresponden a la secuencia en que se realiza el trabajo y el correspondiente tiempo de procesamiento se colocan entre paréntesis.

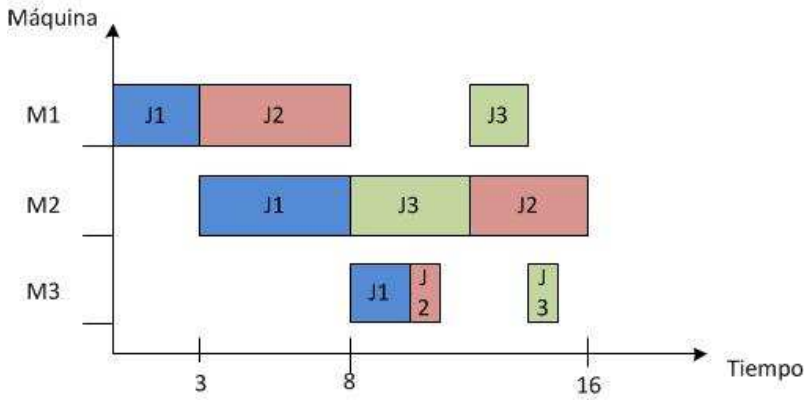
Tabla 1. Ejemplo del JSP para 3 trabajos y 3 máquinas

Trabajo	**Secuencia de máquinas (tiempo de procesamiento)**		
1	1 (3)	2 (5)	3 (2)
2	1 (5)	3 (1)	2 (4)
3	2 (4)	1 (2)	3 (1)

Fuente: YEN, Gary G. y IVERS, Brian.

Entonces, un *schedule* semi-activo para este Ejemplo se muestra en la Figura 3. Las operaciones con restricciones de precedencia se programan lo más rápido posible.

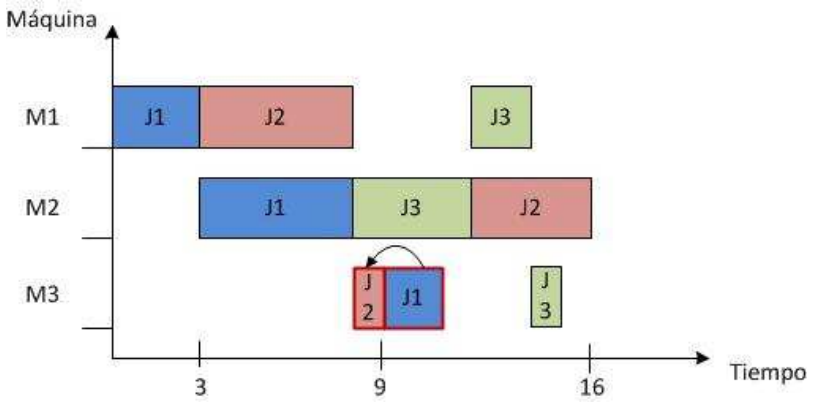
Figura 3. Schedule semi-activo para el Ejemplo 1



Fuente: YEN, Gary. y IVERS, Brian.

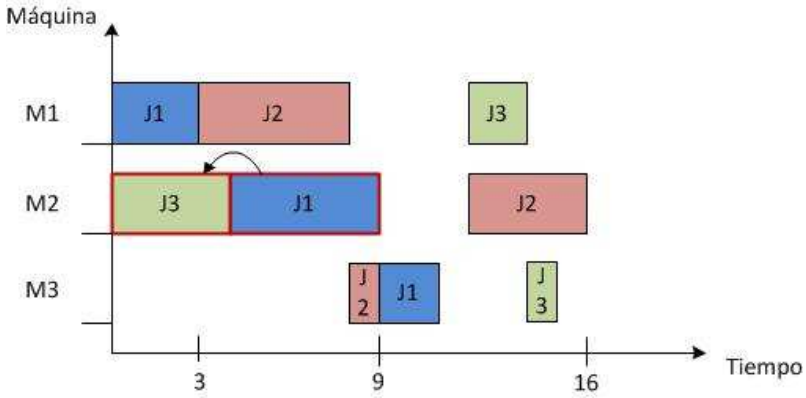
En aras de convertir el diagrama anterior en un schedule activo se realizan cambios permitidos hacia la izquierda, como se muestra en la Figura 4 y en la Figura 5. Los cambios hacia la izquierda consisten en intercambiar 2 operaciones adyacentes que no rompan alguna restricción o causen alguna demora.

Figura 4. Cambio hacia la izquierda del J2 por el J1 en la máquina 3 para el Ejemplo 1



Fuente: YEN, Gary. y IVERS, Brian.

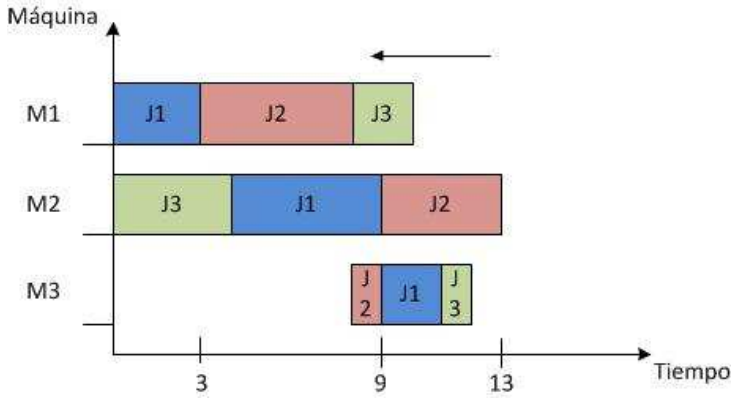
Figura 5. Cambio hacia la izquierda del J3 por el J1 en la máquina 2 para el Ejemplo 1



Fuente: YEN, Gary. y IVERS, Brian.

La Figura 6., representa un *schedule* activo después de realizar todos los cambios permitidos hacia la izquierda.

Figura 6. Schedule Activo después de realizar cambios a la izquierda



Fuente: YEN, Gary. y IVERS, Brian.

La figura anterior también representa un *schedule* sin retraso pues las operaciones se asignan lo más rápido posible a las máquinas sin demora alguna. Se recuerda que los *schedules* óptimos se encuentran, usualmente, en el conjunto de los *schedules* activos. Los *schedules* sin retraso también son activos pero no necesariamente es cierto lo inverso.

Por otra parte, según Brucker (2007) el JSP hace parte de los problemas General Shop que se definen de la siguiente manera:

Para n trabajos ($(i = 1, 2, \ldots, n)$, existen m máquinas $M_1, \ldots, M_m$ donde se realizan los trabajos. Cada trabajo i consiste en un conjunto de operaciones O_{ij} $(j = 1, \ldots, \mathrm{m})$ con tiempos de procesamiento p_{ij}. Cada operación O_{ij} consume p_{ij} unidades de tiempo en una máquina $\mu_{ij} \in \{M_1, \ldots, M_m\}$. Puede haber relaciones de precedencia entre las operaciones de todos los trabajos. Cada trabajo puede ser procesado sólo por una máquina al tiempo y cada máquina sólo puede procesar un trabajo al tiempo. El objetivo deseado es encontrar una programación factible que minimice la función objetivo de los tiempos de finalización C_i de los trabajos $i = 1, \ldots, n$ o *makespan*.

3.1.1 Formulación del JSP en su forma tradicional.

En este numeral se hace la formulación clásica del JSP que ha sido ampliamente estudiada por diversos autores que se basa en la optimización del JSP con el objetivo de minimizar el makespan.

En el problema del JSP se tiene un conjunto de trabajos y otro de máquinas, cada trabajo posee una secuencia de operaciones las cuales se realizan en una de las máquinas durante un tiempo de procesamiento. Cada máquina puede procesar sólo un trabajo a la vez y se tienen restricciones de precedencia de la forma $O_{ij} \rightarrow O_{i,j+1}$ $(j = 1,...,m-1)$.

Las operaciones se realizan de acuerdo a las siguientes tres restricciones[13]:

- Restricción Conjuntiva o de Precedencia: La secuencia de los trabajos en las máquinas está definida.
- Restricción Disyuntiva o de Recursos: Cada máquina sólo puede realizar una operación a la vez.
- Restricción de no-expulsión: Expresa que deben ser procesadas en un intervalo de tiempo sin interrupción (*non-preemptive scheduling*).

La solución a este tipo de problemas se basa en determinar la asignación óptima de un número finito de recursos a un número finito de operaciones teniendo en cuenta las restricciones de precedencia. Es decir, definir la secuencia en la que cada máquina va a procesar cada trabajo de manera que satisfaga el criterio establecido. El cual, generalmente, está asociado con la minimización del *makespan* (tiempo de salida del último trabajo del taller).

[13]SIERRA, María Rita. Mejora de Algoritmos de Búsqueda Heurística Mediante Poda por Dominancia. Aplicación a Problemas de Scheduling. Tesis Doctoral, Universidad de Oviedo, Departamento de Informática, 2009.

Dados n trabajos $i = 1, 2, 3, \ldots n.$ y m máquinas $M_1, \ldots, M_m$. Cada trabajo i consiste en una secuencia de n_i operaciones $O_{i1}, O_{i2}, \ldots, O_{in_i}$ que deben ser procesadas en ese orden y que se relacionan con las restricciones de precedencia $O_{ij} \rightarrow O_{i,j+1}\ (j = 1, \ldots\ n_i - 1)$. Hay una máquina $M_{ij} \in \{M_1, \ldots, M_m\}$ y un tiempo de procesamiento p_{ij} asociado con cada operación O_{ij}. Es decir, la operación O_{ij} necesita p_{ij} unidades de tiempo en la máquina M_{ij}.

Se necesita encontrar una programación factible que minimice la función objetivo que depende de los tiempos C_i de las últimas operaciones O_{i,n_i} de los trabajos.

- s_{ij} : Tiempo de inicio de la operación O_{ij}
- C_{ij} : Tiempo de finalización de la operación O_{ij}, $C_{ij} = s_{ij} + p_{ij}$
- $C_{\max}$: *Makespan* de la programación

$$C_{max} = \max(C_{1n_1}, C_{2n_2}, \ldots, C_{in_i}), \quad i = 1, 2, \ldots, n$$

Para el JSP tradicional se aplican las siguientes reglas:

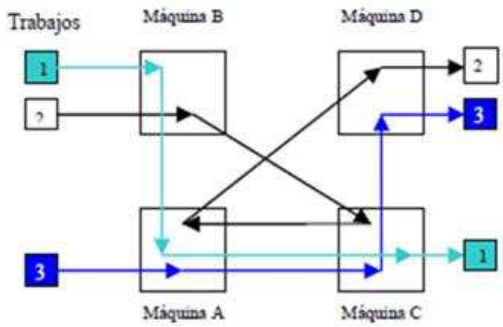
- Cada trabajo debe ser procesado por una máquina en un cierto orden.
- Cada máquina sólo puede procesar un trabajo a la vez.
- Cada trabajo debe ser procesado por cada máquina exactamente una vez y una vez que un trabajo ha iniciado no puede interrumpirse.

3.1.2 Configuración productiva del JSP.

La configuración productiva tipo Job Shop se caracteriza por la fabricación en pequeñas series de una gran variedad de productos en la que cada trabajo tiene una secuencia definida que es diferente y obligatoria. Los trabajos pueden compartir máquinas pero con un orden de visita diferente. A través de un modelo gráfico (Ver Figura 7.) se puede representar la ruta de cada trabajo. Se observa

que el trabajo 1 debe pasar por la máquina B, la máquina A y la máquina C. El trabajo 2 debe pasar por las máquinas B-C-A-D y el trabajo 3 por las máquinas A-C-D.

Figura 7. Configuración tipo Job Shop



Fuente: Sipper y Bulfin (1997).

Algunos ejemplos de empresas tipo *Job Shop* son los talleres metal-mecánicos, los hospitales, los fabricantes de aviones, de autopartes y de semiconductores, entre otros[14].

3.1.3 Representación del Problema.

A continuación se menciona un ejemplo de una instancia de tamaño 3×3 del JSP para mostrar la representación del JSP.

Ejemplo 2. Se tiene un problema de 3 trabajos y 3 máquinas donde, cada trabajo tiene una secuencia correspondiente y tiempos de procesamiento así:

- El trabajo 1 (J1) debe pasar por las máquinas 1, 2 y 3 con tiempos de procesamiento 3, 4 y 3 unidades de tiempo respectivamente.

[14] LONDOÑO, Pablo. MORA, Sebastián y XIBILLÉ, Juan Esteban. Módulo Job-Shop Scheduling para el Proyecto Arquímedes mediante el Heurístico Shifting Bottleneck. (Proyecto de Grado) Medellín, Universidad EAFIT, Departamento de Ingeniería de Producción, 2004, p. 100.

- El trabajo 2 (J2) debe pasar por las máquinas 2, 1 y 3 con procesamiento de 2, 4 y 3 unidades de tiempo respectivamente.
- El trabajo 3 (J3) debe pasar por las máquinas 1 y 3 con procesamiento de 3 y 2 unidades de tiempo respectivamente.

En la Tabla 2 se resume de manera más completa el enunciado anterior.

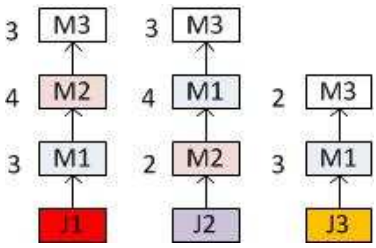
Tabla 2. Ejemplo de una instancia 3×3 del JSP

Trabajo	**Secuencia máquinas**	**Tiempos de procesamiento**
1	1,2,3	$p_{11}=3,\quad p_{12}=4\quad p_{13}=3$
2	2,1,3	$p_{22}=2,\quad p_{21}=4,\quad p_{23}=3$
3	1,3	$p_{31}=3,\quad p_{33}=2$

Fuente: Autor.

En la Figura 8 se muestra un modelo gráfico para el mismo problema tomando en cuenta la información anterior.

Figura 8. Modelo Gráfico del JSP con instancia 3x3.



Fuente: Autor

Entre las formas de representación, más comunes, de una solución para el JSP se encuentran:

a) Diagrama de Gantt: El diagrama de Gantt (Ver Figura 9.) es una forma conveniente de visualizar la representación de la solución del JSP. En el eje vertical se encuentra la máquina y los trabajos a realizar en ésta (dibujados como bloques). En el eje horizontal se muestra el tiempo de inicio y el procesamiento del trabajo en la máquina.

En la siguiente tabla se muestra la operación a realizar, su predecesor y el tiempo de procesamiento. Con esta tabla se puede realizar el diagrama de Gantt para el ejemplo. La operación O_{ij} hace referencia a la operación del trabajo i en la máquina j. Por ejemplo, la operación O_{11} es la operación del trabajo 1 que se realiza en la máquina 1.

Tabla 3. Operaciones, restricciones de precedencia y tiempos de procesamientos.

Operación	**Predecesor**	**Tiempo**
O_{11}	-	3
O_{12}	O_{11}	4
O_{13}	O_{12}	3
O_{22}	-	2
O_{21}	O_{22}	4
O_{23}	O_{21}	3
O_{31}	-	3
O_{33}	O_{31}	2

Fuente: Autor

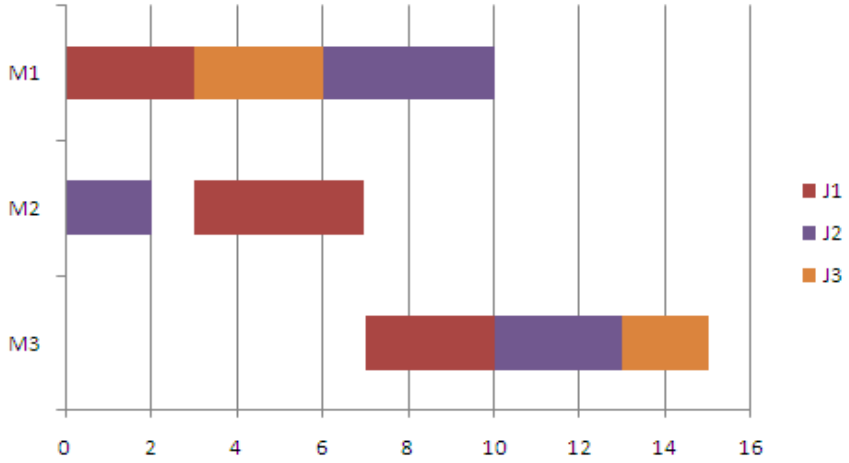
El diagrama de Gantt se construye de la siguiente manera:

- Se grafican los ejes principales.

- Se ubica en el diagrama las operaciones que NO tienen predecesores.
- Se ubican las operaciones que dependen de las operaciones del punto anterior y así, sucesivamente, hasta que todas las operaciones estén graficadas.

Para el Ejemplo 2, se inicia con las operaciones O_{11}, O_{22} y O_{31} continuando con la operación O_{12} y así, con las demás operaciones, hasta que se obtiene un diagrama de Gantt como el que se muestra en la Figura 9.

Figura 9. Representación en el diagrama de Gantt de una solución para el problema 3×3.



Fuente: Autor.

Se puede observar con el diagrama de Gantt anterior que con esta programación factible el *makespan* es igual a 15 unidades de tiempo. Es decir, el instante de salida del último trabajo (en este caso, el Trabajo 3 - J3) del taller es igual a 15 unidades de tiempo.

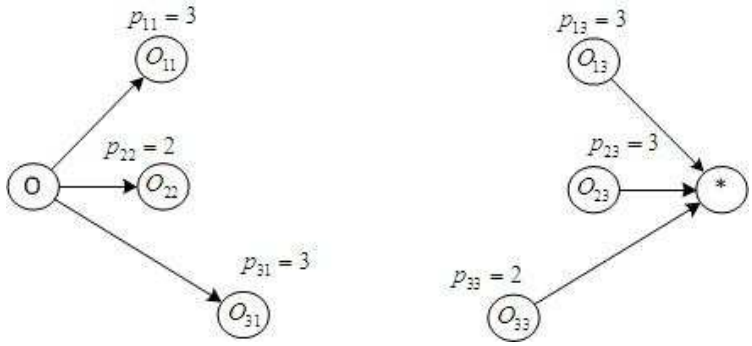
b) Grafo: Los problemas de *scheduling* también pueden representarse por medio de un *Grafo* $G=\{V,C\cup D\}$ (Roy and Sussman), donde:

- V es el conjunto de nodos que representan las operaciones O_{ij}

($i = 1, 2, ..., n \quad ; j = 1, 2, ..., m$) el problema. Contiene dos nodos ficticios (con valor igual a cero), el nodo "source" o fuente (O) y el nodo "sink" o sumidero ($\star$).

- C es el conjunto de arcos que representan las restricciones de precedencia entre las operaciones de cada trabajo. Contiene los arcos desde el nodo fuente hasta todas las operaciones sin un predecesor. Es decir, desde O hasta O_{11}, O_{22} y O_{31}. Además, contiene los arcos entre las operaciones sin un sucesor y el sumidero. Es decir, desde O_{13}, O_{23} y O_{33} hasta ($\star$). (Ver Figura 10.)

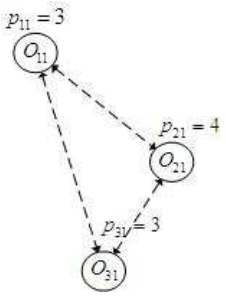
Figura 10. Arcos dirigidos, nodo inicio y nodo final.



Fuente: Autor.

- D es el conjunto de arcos no dirigidos, que representan las operaciones que deben ser realizadas en la misma máquina. Es decir, las restricciones de capacidad de las máquinas por ejemplo, O_{11}, O_{21} y O_{31} (Ver Figura 11.)

Figura 11. Arcos no dirigidos

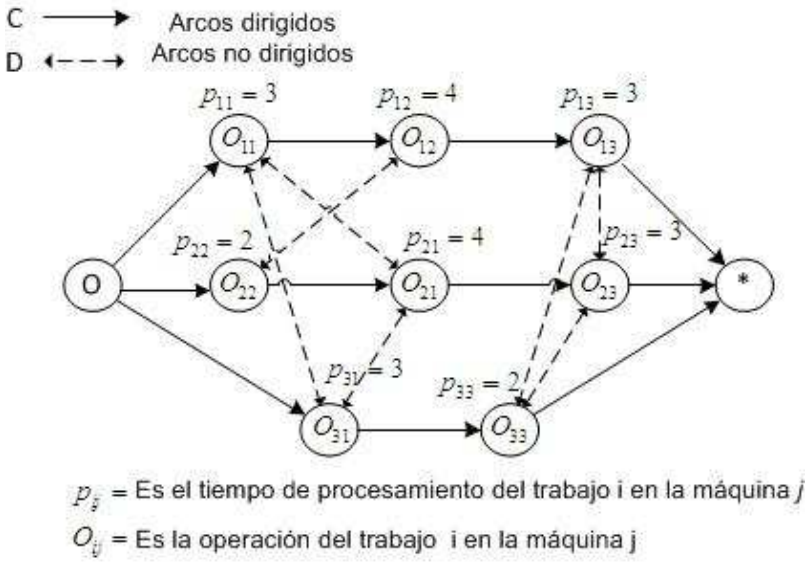


Fuente: Autor.

El tiempo de procesamiento p_{ij} para cada operación es un peso ponderado que se ubica junto a los nodos correspondientes. Cada nodo tiene una etiqueta O_{ij} que indica la máquina en la que la operación correspondiente debe ser procesada donde, i es el trabajo y j la máquina.

El grafo se construye de la siguiente manera: Primero se ubican los nodos correspondientes a las operaciones del problema. Se agregan los dos nodos ficticios, el nodo fuente (O) y el nodo sumidero ($\star$). Luego, se unen los nodos por medio de flechas dirigidas siguiendo las restricciones de precedencia. Finalmente, se agregan flechas dobles (no dirigidas) entre las operaciones que se realizan en la misma máquina, es decir, las restricciones de capacidad de la máquina.

Figura 12. Grafo para el Ejemplo 2



Fuente: Autor.

En este grafo se observa que entre cada operación consecutiva del mismo trabajo hay un arco dirigido, el cual representa las restricciones de precedencia. Es decir, el orden que deben seguir las operaciones para completar el trabajo. En el grafo

se muestra que la operación O_{11} debe realizarse primero que la operación O_{12}, así $O_{11} \rightarrow O_{12}$.

Entre cada operación de la misma máquina se tiene una arista no dirigida (disyuntiva) que representa las restricciones de capacidad de la máquina. En el grafo se muestra que la operación O_{11} se puede realizar primero o después que la operación O_{21} así O_{11} $\leftarrow\!\!-\!\!\rightarrow$ O_{21} ó primero o después que la operación O_{31} así O_{11} $\leftarrow\!\!-\!\!\rightarrow O_{31}$. De Brucker (2007) se extrae que la idea básica es definir un orden entre las operaciones conectadas por los arcos disyuntivos, es decir, dirigir las aristas.

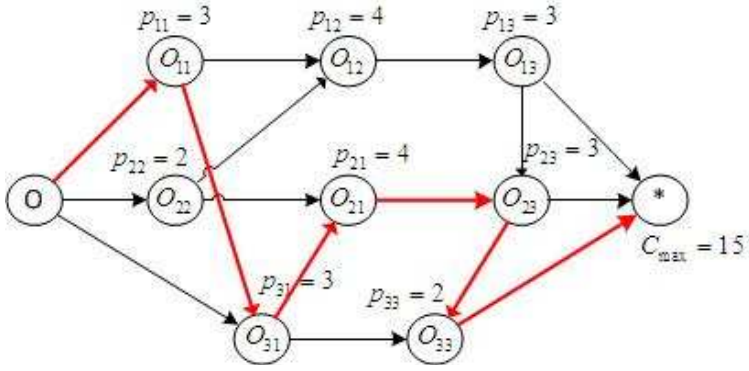
Una selección S es el conjunto de los arcos disyuntivos una vez se les haya dado una dirección, los cuales se llaman arcos “arreglados”.

Se dice que una selección está completa cuando cada arco ha sido arreglado y el grafo resultante $G = \{V, C \cup S\}$ es acíclico. Es decir para cada nodo, de la operación O_{ij}, no hay un camino directo que empiece y termine en el mismo nodo.

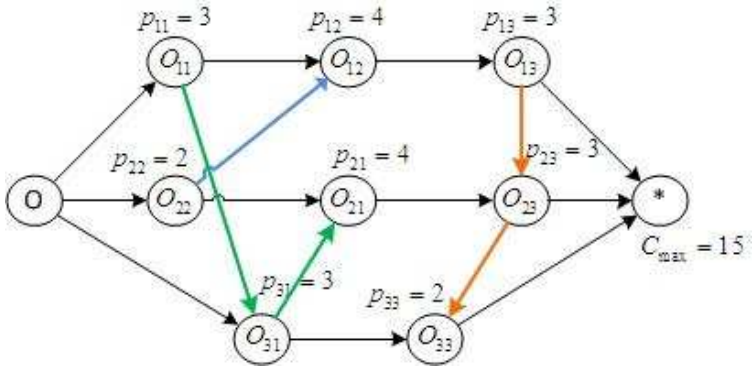
Para una solución S completa, el *makespan* de una programación factible se determina a través del camino más largo desde la fuente hasta el sumidero. Es decir, desde que la primera operación inicie en O con 0 (cero) y termine con el mayor valor entre los que llegan al sumidero. La idea es encontrar una selección de arcos disyuntivos que minimice la longitud de este camino más largo como se muestra en la Figura 13 a, delineado con rojo. La línea roja indica que al finalizar de procesar todos los trabajos con esta posible solución, el último trabajo en salir del taller (trabajo 3) indica un *makespan* de 15 unidades de tiempo.

Figura 13. Grafo de una posible solución para el Ejemplo 2, con un makespan igual a 15 unidades de tiempo

a)



b)



Fuente: Autor

La Figura 13 b, muestra el orden escogido de trabajos para cada máquina. Es decir, la máquina 1 procesa primero el trabajo 1, el trabajo 3 y el trabajo 2. La máquina 2 procesa primero el trabajo 2 y luego el trabajo 3. La máquina 3 procesa el trabajo 1, el trabajo 2 y luego el trabajo 3. Programación con la cual se obtiene un *makespan* de 15 unidades de tiempo.

Esta es una de las posibles soluciones que se pueden encontrar para el Ejemplo 2, la idea es tratar de encontrar una solución que minimice, para este caso, el *makespan* de la programación.

Según Pinedo (1999), existen varias formulaciones matemáticas para el JSP y entre las más usadas, se encuentra la programación disyuntiva. La cual, está muy relacionada con la representación en el Grafo.

3.1.4 Programación disyuntiva.

La programación disyuntiva hace referencia a minimizar (maximizar) una función lineal que satisface un sistema lógico de estados conjuntivos (del tipo "y", "$\wedge$") y disyuntivos (del tipo "o", "$\vee$"), donde cada estado está definido por una restricción lineal.

La siguiente es la notación necesaria para la formulación matemática del JSP:

- y_{ij} es el tiempo de inicio de la operación (i, j), (trabajo i, máquina j)
- p_{ij} es el tiempo de procesamiento de la operación (i, j)
- N es el conjunto de todas las operaciones O_{ij}
- A es el conjunto de restricciones de precedencia $(i, j) \rightarrow (i, k)$

El siguiente es el problema de optimización que permite hallar la solución del problema de secuenciamiento de trabajos (JSP) y minimización del *makespan*.

$$\min C_{\max}$$

$$s.a:$$

$$y_{ik} - y_{ij} \geq p_{ij} \qquad \forall (i, j) \rightarrow (i, k) \in A \qquad (1)$$

$$C_{\max} - y_{ij} \geq p_{ij} \qquad \forall (i, j) \in N \qquad (2)$$

$$y_{ij} - y_{lj} \geq p_{lj} \quad or \quad y_{lj} - y_{ij} \geq p_{ij} \qquad \forall (l, j)\ y\ (i, j)\ \ j = 1, \ldots, m \qquad (3)$$

$$y_{ij} \geq 0 \qquad \forall (i, j) \in N$$

Donde, las restricciones 1 y 2 aseguran que la operación (i, k) empieza una vez que la operación (i, j) haya sido procesada y la restricción 3 asegura que exista

un orden entre las operaciones, de diferentes trabajos, que van a ser procesados en la misma máquina.

A continuación, se presenta en extenso la formulación matemática para el Ejemplo 2.

$$\min C_{\max}$$
$$s.a:$$

El primer conjunto de restricciones (1) consta de CINCO restricciones en total:

Dos para el trabajo 1;

$$y_{12} - y_{11} \geq 3$$
$$y_{13} - y_{12} \geq 4$$

Dos para el trabajo 2;

$$y_{21} - y_{22} \geq 2$$
$$y_{23} - y_{21} \geq 4$$

Una para el trabajo 3;

$$y_{33} - y_{31} \geq 3$$

Para el segundo conjunto se tienen OCHO restricciones, una por cada operación a realizar:

$$C_{\max} - y_{11} \geq 3$$
$$C_{\max} - y_{12} \geq 4$$
$$C_{\max} - y_{13} \geq 3$$

$$C_{\max} - y_{22} \geq 2$$
$$C_{\max} - y_{21} \geq 4$$
$$C_{\max} - y_{23} \geq 3$$

$$C_{\max} - y_{31} \geq 3$$
$$C_{\max} - y_{33} \geq 2$$

Las restricciones disyuntivas son:

Dos para la máquina 1:

$$y_{11} - y_{21} \geq 4 \quad ó \quad y_{21} - y_{11} \geq 3$$
$$y_{21} - y_{31} \geq 3 \quad ó \quad y_{31} - y_{21} \geq 4$$

Una para la máquina 2:

$$y_{12} - y_{22} \geq 2 \quad ó \quad y_{22} - y_{12} \geq 4$$

Y dos para la máquina 3:

$$y_{13} - y_{23} \geq 3 \quad ó \quad y_{23} - y_{13} \geq 3$$
$$y_{33} - y_{23} \geq 3 \quad ó \quad y_{23} - y_{33} \geq 2$$

Finalmente, el último conjunto incluye 8 restricciones de no-negatividad, una por cada tiempo de inicio;

$$y_{11,} y_{12}, y_{13}, y_{22,} y_{21}, y_{23,} y_{31}, y_{33} \geq 0$$

Con la anterior formulación, se puede observar que a medida que el problema se hace más grande las restricciones aumentan de manera exponencial. Entonces, el problema se hace más complejo de resolver por medio de la programación disyuntiva haciéndose necesario el uso de métodos más eficientes tanto en tiempo como en espacio.

3.1.5 Programación lineal entera mixta.

Otro de los modelos matemáticos más usados para resolver el JSP es la programación lineal entera mixta que fue propuesta por Alan Manne (1960). El modelo se compone de una función objetivo lineal y un conjunto de restricciones lineales donde algunas variables de decisión son enteras (y_{ipk}). Estas variables son binarias y se usan para implementar las restricciones disyuntivas.

El modelo tiene los siguientes parámetros y se presenta de la siguiente manera:

- t_{ij}, es el tiempo de inicio de cada operación
- J, es el conjunto de n trabajos a ser procesados
- M, es el conjunto de m recursos o máquinas
- O_{ij}, es la operación del trabajo i que debe ser procesado en la máquina j
- τ_{ij}, es el tiempo de procesamiento que no puede ser interrumpido
- K, un valor arbitrario grande. Debe ser mayor que la suma de los tiempos de procesamiento de todas las operaciones menos el tiempo más pequeño $(K > \sum_{i}^{n}\sum_{j}^{m}\tau_{ij} - \min(\tau_{ij}))$

$$Min\ C_{\max} = \max(t_{ij}, \tau_{ij}) \quad : \forall i \in J,\ j \in M.$$

Sujeto a:

$$t_{ij} \geq 0 \qquad \{i,l\} \in J \quad \{j,k\} \in M$$
$$t_{ij} - t_{ik} \geq \tau_{ik} \qquad si\ O_{ik}\ precede\ a\ O_{ij}$$
$$t_{lj} - t_{ij} + K(1 - y_{ilj}) \geq \tau_{ij} \qquad y_{ilj} = 1, \quad si \quad O_{ij}\ precede\ a\ O_{lj}$$
$$t_{ij} - t_{lj} + K(y_{ilj}) \geq \tau_{lj} \qquad y_{ilj} = 0, \quad en\ otro\ caso.$$

Según Peña y Zumelzu (2006) para n trabajos y m máquinas, en este modelo existen $2nm + \frac{n(n-1)m}{2}$ variables de decisión donde, $\frac{n(n-1)m}{2}$ son variables binarias y $n(m-1) - n(n-1)m + 2nm$ es el número de restricciones. Existen, aproximadamente, $(n!)^m$ posibles soluciones para n trabajos y m máquinas. Entonces, se puede observar que para problemas de tamaño grande, el número de variables de decisión y de restricciones crece de manera exponencial, lo que usualmente, hace que requieran de tiempos computacionales muy altos para resolverlos.

Para el Ejemplo 2 se presenta en extenso la anterior formulación.

$$\min C_{\max} = \max(t_{ij}, \tau_{ij})$$

s.a:

Tiempos de inicio para cada operación: $t_{11}, t_{12}, t_{13}, t_{22}, t_{21}, t_{23}, t_{31}, t_{131} \geq 0$

Restricciones de precedencia;

Para el trabajo 1:
$$t_{11} - t_{12} \geq 4$$
$$t_{12} - t_{13} \geq 3$$

Para el trabajo 2:
$$t_{22} - t_{21} \geq 4$$
$$t_{21} - t_{23} \geq 3$$

Para el trabajo 3: $t_{31} - t_{32} \geq 2$

Restricciones disyuntivas (binarias);

Máquina 1:
$$t_{21} - t_{11} + K(1 - y_{121}) \geq 3 \quad \acute{o} \quad t_{11} - t_{21} + K(y_{121}) \geq 4$$
$$t_{31} - t_{21} + K(1 - y_{231}) \geq 4 \quad \acute{o} \quad t_{21} - t_{31} + K(y_{231}) \geq 3$$

Máquina 2: $t_{22} - t_{12} + K(1 - y_{122}) \geq 4 \quad \acute{o} \quad t_{12} - t_{22} + K(y_{122}) \geq 2$

Máquina 3:
$$t_{23} - t_{13} + K(1 - y_{123}) \geq 3 \quad \acute{o} \quad t_{13} - t_{23} + K(y_{123}) \geq 3$$
$$t_{33} - t_{23} + K(1 - y_{233}) \geq 3 \quad \acute{o} \quad t_{23} - t_{33} + K(y_{233}) \geq 2$$

Luego, para un problema $3x3$ como el del Ejemplo 2 se tienen 27 variables decisión -9 de tipo binario-, 6 restricciones y $(3!)^3 = 216$ posibles soluciones.

3.2 JSP CON MÚLTIPLES OBJETIVOS

La mayoría de las investigaciones están enfocadas en resolver el JSP con la minimización del *makespan*. Es decir, la optimización de un solo objetivo. En un mundo industrial tan complejo como el de hoy en día, la idea es tratar de solucionar el problema del JSP con múltiples objetivos involucrados.

En el *Multi-objective Job Shop Scheduling Problem* (MOJSP) se presentan múltiples objetivos en conflicto y se tienen algunas dificultades relacionadas a éstos. Si los objetivos son combinados en una función escalar al usar pesos, la dificultad está en asignar los pesos por cada objetivo. Si todos los objetivos son optimizados al mismo tiempo, el problema está en diseñar un algoritmo de búsqueda efectivo tomando en cuenta el incremento considerable en la complejidad del tiempo[15].

La idea es generar una gran cantidad de *schedules*, lo más cercanos al óptimo, considerando los múltiples objetivos de acuerdo a los requerimientos de la orden de producción. Considerando el principio de optimización de múltiples objetivos es casi imposible encontrar un *schedule* óptimo que satisfaga todas las funciones objetivo. Es por esto, que se prefiere obtener tantas soluciones de Pareto como sea posible. Estas soluciones son no-dominadas y convergen y divergen cerca del frente de Pareto respecto a los múltiples criterios.

Para reflejar la situación del mundo real en esta investigación se trabajará con la minimización de un problema con dos objetivos, el makespan y la demora total, mediante las soluciones de Pareto.

$(1)\, Makespan\, C_{max} = \max\{C_i\}$, donde C_i es el tiempo en el que se completa el trabajo i . $t(i.j)$ es el tiempo de procesamiento del trabajo $i = 1, 2, ..., n$ en la máquina $j = 1, 2, ..., m$ y $\{\pi_1, \pi_2, ..., \pi_n\}$ es la permutación de los trabajos.

Makespan: $C(\pi, j)$

[15] LEI, Deming. Op. cit.

$$C(\pi_1,1) = t(\pi_1,1)$$
$$C(\pi_i,1) = C(\pi_{i-1},1) + t(\pi_i,1) \qquad i = 2,...,n$$
$$C(\pi_1,\, j) = t(\pi_1,\, j-1) + t(\pi_i,\, j) \quad j = 2,...,m$$
$$C(\pi_i,1) = \max\{C(\pi_{i-1},\, j), C(\pi_i,\, j-1)\} + t(\pi_i,\, j)$$

$$\textit{Makespan, } f_{C\max} = C(\pi_n, m) \qquad (4)$$

El planteamiento en extenso para hallar el makespan se describe a continuación aplicado al Ejemplo 2.

$$C(\pi_1,1) = t(\pi_1,1) = 3$$
$$C(\pi_2,1) = C(\pi_1,1) + t(\pi_2,1) = 3+4 = 7$$
$$C(\pi_3,1) = C(\pi_2,1) + t(\pi_3,1) = 7+3 = 10$$
$$C(\pi_1,2) = t(\pi_1,1) + t(\pi_1,2) = 3+4 = 7$$
$$C(\pi_1,3) = t(\pi_1,2) + t(\pi_1,3) = 7+3 = 10$$
$$C(\pi_2,2) = \max\{C(\pi_1,2), C(\pi_2,1)\} + t(\pi_2,2) = \max(7,7)+2 = 9$$
$$C(\pi_3,2) = \max\{C(\pi_2,2), C(\pi_3,1)\} + t(\pi_3,2) = \max(9,10)+0 = 10$$
$$C(\pi_2,3) = \max\{C(\pi_1,3), C(\pi_2,2)\} + t(\pi_2,3) = \max(10,9)+3 = 13$$
$$C(\pi_3,3) = \max\{C(\pi_2,3), C(\pi_3,2)\} + t(\pi_3,3) = \max(13,10)+2 = 15$$

$$\textit{Makespan, } f_{C\max} = C(\pi_3,3) = 15$$

$$(2)\ \textit{Demora total de los trabajos } f_{tottard} = \sum_{i=1}^{n} \max(0, L_i) \qquad (5)$$

Donde el retraso L_i cuantifica el tiempo que se ha demorado (o no) en terminar el trabajo. Se denomina tardanza L_i del trabajo i, $L_i = C_i - d_i$ si la diferencia es mayor que cero (0). Es decir, el trabajo se terminó después de la fecha de entrega. Si es menor que cero se llama prontitud E_i y quiere decir que se terminó antes de la fecha de entrega.

Suponiendo que las fechas de entrega de los trabajos es la siguiente (Ver. Tabla 4) y continuando con el Ejemplo 2.

Tabla 4. Ejemplo 2, instancia $3x3$ del JSP con fechas de entrega.

Trabajo	Secuencia máquinas	Tiempos de procesamiento	Fechas de entrega
1	1,2,3	$p_{11}=3, \quad p_{12}=4 \quad p_{13}=3$	$d_1=9$
2	2,1,3	$p_{22}=2, \quad p_{21}=4, \quad p_{23}=3$	$d_2=11$
3	1,3	$p_{31}=3, \quad p_{33}=2$	$d_3=14$

Fuente: Autor.

Tardanza de los trabajos;

$$L_1 = C_1 - d_1 = 10 - 9 = 1$$
$$L_2 = C_2 - d_2 = 13 - 11 = 2$$
$$L_3 = C_3 - d_3 = 15 - 14 = 1$$

$$f_{tottard} = \sum_{i=1}^{n} \max(0, L_i) = \max(0,1) + \max(0,2) + \max(0,1) = 4$$

La tardanza total para este ejemplo es de 4 unidades de tiempo.

4. MÉTODOS PARA SOLUCIONAR EL JSP

Las formas más básicas para resolver los problemas de *scheduling* son:

- La programación manual es sencilla de llevar a cabo, sin embargo, genera un gran número de trabajos que están en proceso y los tiempos de entrega son largos. La programación manual no es eficiente para empresas complejas que manejan numerosas órdenes de trabajo con fechas de entrega diferentes.
- La Programación en tableros y hojas de Excel es eficiente para pequeños sistemas pero muy ineficiente e inconveniente para grandes sistemas con operaciones de trabajos en recursos de capacidad finita.

Entre los enfoques más eficientes reportados en la bibliografía de la Investigación de Operaciones para hallar la solución del JSP se encuentran las heurísticas y las metaheurísticas.

4.1 HEURÍSTICAS

Término que deriva de la palabra griega “heuriskein”, que significa encontrar ó descubrir y se usa en el ámbito de la optimización para describir una clase de algoritmos de resolución de problemas.
El método heurístico es más flexible que un método exacto permitiendo por ejemplo, la incorporación de condiciones de difícil modelización. Se utiliza como parte de un procedimiento global que garantiza el óptimo de un problema, en donde existen dos posibilidades:

- Proporciona una buena solución inicial de partida.
- Participa en un paso intermedio del procedimiento.

Se ha demostrado que dependen en gran medida del problema concreto para el que se han diseñado[16].

Un ejemplo de un método heurístico, comúnmente, aplicado a diferentes problemas son las listas de prioridad de despacho, las cuales abrieron la puerta a la aplicación de los métodos metaheurísticos para resolver el JSP.

4.2 METAHEURÍSTICAS

Al principio de la década de los 80 aparecen una serie de metodologías bajo el nombre de *metaheurísticas* cuyo propósito es del superar los mejores resultados

[16] MARTÍ, Rafael. Op. Cit.

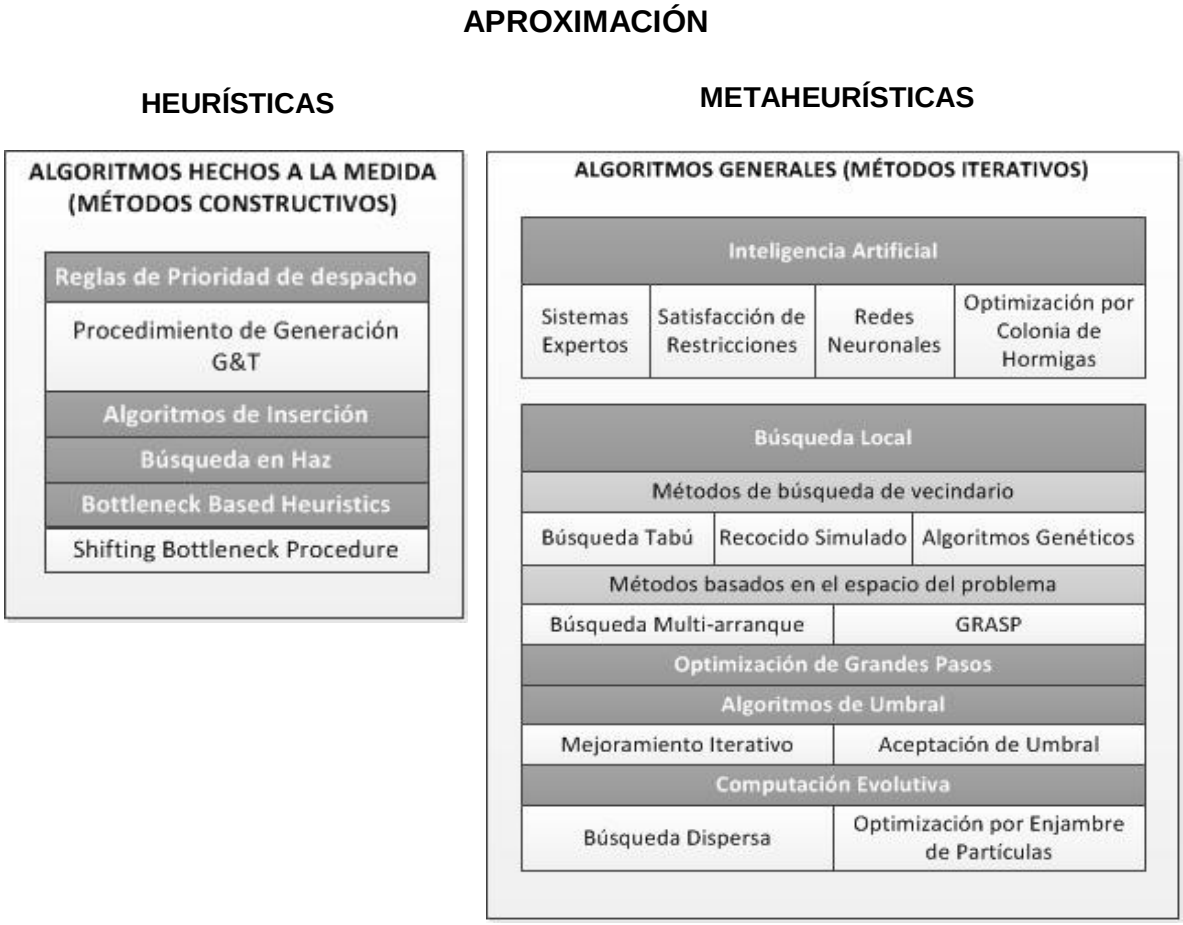
que se han obtenido por las heurísticas tradicionales. Las metaheurísticas se sitúan –conceptualmente- por encima de las heurísticas. El término metaheurística fue introducido por primera vez por Fred Glover en 1986.

Osman y Kelly (1996) introducen la siguiente definición: “los procedimientos metaheurísticos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. Las Metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial y la evolución biológica”.
Los problemas de *scheduling* hacen uso de los algoritmos genéticos, el recocido simulado, la búsqueda tabú, el enjambre de partículas y la optimización por colonia de hormigas[17].

Diferentes metodologías han sido usadas para la solución del problema del JSP, en la Figura 14, se muestran las principales heurísticas y metaheurísticas.

[17] YEN, G. G. y IVERS, B. Op. Cit.

Figura 14. Principales técnicas de aproximación utilizadas para resolver el JSP.



Autor: Adaptación al presente proyecto a partir de JAIN, A.S y MEERAN, S. Pág. 393.

4.3 REVISIÓN DE METODOLOGÍAS APLICADAS AL JSP

Existen dos grandes vertientes, la primera vertiente se dedica a la elaboración de métodos exactos de solución (*métodos de optimización)* son muy eficientes para hallar la solución óptima para problemas que tienen un número pequeño de trabajos y máquinas. La segunda vertiente son los *métodos de aproximación* que son más eficientes pero no garantizan una solución óptima. Estos pueden ser

constructivos (heurísticas) que crean una solución a partir de los datos de un problema o *iterativos (metaheurísticas)* que modifican la solución al reordenar continuamente la secuencia de operaciones[18].

4.3.1 Métodos de Optimización (Métodos Exactos).

La característica más importante, de los métodos exactos para hallar la solución de un problema de optimización, es que exploran todo el espacio solución y están diseñados para encontrar una solución óptima. Sin embargo, para problemas que se consideran de tamaño moderado, grande y muy grande son inadecuados por el alto uso de tiempo computacional.

4.3.1.1 Métodos enumerativos. Los métodos enumerativos son aquellos que buscan la solución de los problemas de programación lineal entera mixta al explorar, metódicamente, el conjunto de las soluciones posibles enteras para que no sea necesario considerar individualmente todas las posibilidades[19].

a) Formulaciones matemáticas.

- Programación entera mixta: Es una técnica de programación matemática. De acuerdo al conjunto de restricciones y a la función objetivo, se clasifica en programación lineal entera mixta o no lineal. Algunas de las variables de decisión toman valores en el conjunto de los números enteros y otras en el conjunto de los números reales. El uso de variables binarias, en algunos casos, se hace con el propósito de implementar las restricciones disyuntivas[20]. Para problemas de tamaño muy grande del JSP, casi tamaño real, la programación entera mixta no es apropiada debido a que da lugar a tiempos de ejecución elevados, variables con un crecimiento exponencial y un gran número de restricciones.

[18] JAIN, A.S y MEERAN, S. Op. Cit.

[19] BARCELÓ, J. Panorama Actual de los Métodos Enumerativos. Seminario de Programación Matemática PM' 77, Madrid, Parte I.

[20] Las cuales indican que las operaciones correspondientes a una misma máquina no se pueden procesar simultáneamente.

- Métodos de descomposición: Dividen el problema inicial en una serie de sub-problemas más pequeños y manejables que se pueden resolver óptimamente, usualmente, en sub-problemas de una máquina y/o de máquinas paralelas. Sin embargo, la manera en que se descomponga un problema, el tipo de descomposición y el orden que se considere afectan la calidad de la solución generada.

b) Ramificación y poda (*Branch & Bound - BB*). La ramificación y poda es una técnica enumerativa donde el problema raíz se divide en sub-problemas. Puede ser visto como un árbol donde se representa el espacio solución de las posibles soluciones factibles del JSP. Ésta técnica formula procedimientos y reglas que permite que porciones del árbol sean eliminadas de la búsqueda. El método es bastante eficiente en instancias pequeñas, sin embargo, requiere un tiempo computacional excesivo que no permite su aplicación a problemas más grandes. Por muchos años fue una de las técnicas más populares para resolver el JSP. A pesar de los eficientes procedimientos de ramificación y poda que han sido desarrollados para agilizar la búsqueda todavía se requiere de un conocimiento muy amplio sobre el JSP y de procedimientos de selección para aplicarlos a grandes problemas del mismo.

4.3.1.2 Métodos Eficientes. Un algoritmo eficiente crea una solución óptima de los datos del problema al seguir un conjunto simple de reglas que determinan exactamente el orden de procesamiento. Entre las primeros aplicaciones se encuentra la de Johnson (1954), un algoritmo eficiente para el problema de dos máquinas en ambiente flow shop que minimiza el tiempo de flujo máximo. Jackson (1956) abordó el problema de dos máquinas de Johnson en un ambiente Job Shop donde cada trabajo consiste en dos operaciones. En 1982, Hefetz y Adiri (1982) desarrollan un enfoque eficiente para el problema de dos máquinas en Job Shop donde todas las operaciones producen una sola unidad.

4.3.2 Métodos de Aproximación.

Debido a la limitación general de los métodos exactos, los métodos de aproximación se convirtieron en una nueva alternativa. A pesar de no generar soluciones óptimas, pueden encontrar soluciones cercanas al óptimo en un tiempo computacional razonable y pueden ajustarse a problemas de tipo real.

4.3.2.1 Algoritmos Hechos a La Medida (Tailored) – Métodos Constructivos.
Estos métodos -en la mayoría de los casos- construyen una sola solución al aplicar reglas específicas sobre el problema.

a) Reglas de prioridad de despacho (Priority Dispatching Rules, PDRS). Es uno de los métodos más usados para resolver problemas de scheduling por su fácil implementación y por la baja complejidad de tiempo de cómputo. En cada paso, la operación con la mayor prioridad entre todas las operaciones a programar se selecciona y se programa lo más pronto posible. Son procedimientos que proveen buenas soluciones a problemas complejos en tiempo real.

Uno de los procedimientos más usados que se considera la base común de todas las reglas de prioridad de despacho -debido a que puede generar programaciones activas que son óptimas- es el procedimiento de Giffler y Thompson.

- Procedimiento de generación de Giffler y Thompson: Éste algoritmo asigna las operaciones disponibles para procesar en las máquinas correspondientes. En caso de un conflicto con las operaciones de la misma máquina, éste escoge aleatoriamente una de las operaciones. Explora el espacio de búsqueda por medio de un árbol. Los nodos en el árbol representan las programaciones parciales, los arcos representan las posibles elecciones y las hojas del árbol son el conjunto de programaciones (*schedules*). Ésta técnica, en general, es usada para generar

soluciones iniciales. Lo más importante de esta técnica es que genera programaciones activas que son óptimas.

b) Algoritmos de Inserción y Búsqueda en Haz.

- Algoritmo de Inserción (*Insertion Algorithm*)[21]: Es un algoritmo que contiene dos fases. La primera aplica una estrategia constructiva que minimiza la longitud del camino más largo y la segunda aplica una estrategia de reinserción para mejorar iterativamente la solución inicial. Usualmente, se aplica el método de búsqueda de haz para mejorar la búsqueda.

- Búsqueda en Haz (Beam Search)[22]: Incorpora heurísticas en la búsqueda para evitar el seguimiento a cada selección posible manteniendo, al mismo tiempo, la naturaleza combinatoria del problema. Es una adaptación del método de ramificación y poda donde sólo se busca en ciertos nodos del árbol. Posee problemas en la evaluación imperfecta de los nodos prometedores.

c) Heurísticas Basadas en Cuellos de Botella (Bottleneck based heuristics). A pesar de que, por mucho tiempo, los únicos métodos de aproximación viables han sido las reglas de prioridad de despacho han aparecido heurísticas basadas en los cuellos de botella. Entre ellas, el *Shifting Bottleneck Procedure* (SBP) el cual identifica el cuello de botella en cada iteración. La principal contribución del SBP es la forma en que se usa la relajación de una máquina para decidir el orden en el que las máquinas deben ser secuenciadas[23]. Genera programaciones con una gran calidad dependiendo del orden en que los problemas de una sola máquina fueron considerados. Un inconveniente general de las aproximaciones del SBP es el hecho de que se debe completar el procedimiento en su totalidad antes de que

[21] WERNER, Frank, y WINCKLER, Andreas. Insertion techniques for the heuristic solution of the job shop problem. En: Discrete Applied Mathematics, 1995, Vol. 58, Issue 2, pp. 191-211.
[22] MORTON, T. E. y PENTICO, D. W. Heuristic Scheduling Systems. Wiley Series in Engineering and Technology Management, 1993. Wiley, New York.
[23] Ibid.

se obtenga una solución. Requiere un nivel sofisticado de programación y presenta dificultad al realizar la reoptimización y la generación de programaciones no factibles.

d) Razonamiento oportunista. Ha sido usado para caracterizar un proceso de solución de un problema, donde una actividad es dirigida hacia las acciones más prometedoras, en términos del estado actual del problema[24]. Es decir, la programación se concentra en los aspectos más importantes del problema. Puede lograr soluciones óptimas o cercanas al óptimo, incluso para instancias consideradas difíciles.

4.3.2.2 Algoritmos Generales – Métodos Iterativos. Estos métodos generan varias soluciones en tiempos considerablemente bajos. Mejoran las soluciones, a cada paso, al lograr una mejora de la solución anterior. Soluciones que sobresalen en comparación con los métodos constructivos.

a) Inteligencia Artificial: La Inteligencia Artificial (IA) es una disciplina que se fundamenta en el conocimiento del mundo biológico y usa principios de la naturaleza para encontrar soluciones. El objetivo es construir programas en computador que se desempeñen en altos niveles en las tareas cognitivas.

- Sistemas expertos y basados en el conocimiento: El propósito de estos es usar la IA para reemplazar un humano experto en un área. Para esto, necesita tener algún nivel de entendimiento de los datos que guarda. De esta manera, se le asignan reglas para que el sistema le dé un significado a la información que encuentre.

[24] BLAZEWICKZ, Jacek y DOMSCHKE, Wolfgang y PESCH, Erwin. The Job Shop Scheduling Problem: Conventional and new solution techniques. En: European Journal of Operational Research, 1996, Vol. 93, pp. 1-33.

- Clausura y satisfacción de restricciones (CSP): Aplica restricciones que restringen el orden en que las variables se seleccionan y la secuencia en que los posibles valores se asignan a cada variable, para así reducir el tamaño efectivo del espacio de búsqueda. El problema termina cuando la asignación completa de las variables no viola las restricciones del problema. El problema principal es que muchos de estos métodos sólo proveen buenas soluciones pero son de poco uso práctico[25] y a veces pueden no minimizar la función objetivo. Permite integrarse en un marco diseñado para minimizar las fechas de vencimiento relacionadas con la función objetivo. Esta técnica también se usa para minimizar la tardanza ponderada total (total weighted tardiness).

- Redes neuronales: Han sido estudiadas por muchos años en un intento de imitar el aprendizaje y las habilidades de predicción de los seres humanos. Se clasifican de acuerdo a la topología de la red, a las características de los nodos, y las reglas de entrenamiento o aprendizaje. En estas técnicas, la información se lleva a cabo a través de una red masiva interconectada de unidades paralelas de procesamiento.
    - Las redes de Hopfield: Han sido aplicadas al JSP para minimizar el *makespan* basado en la función de energía, sujeto a las restricciones de precedencia y a las de capacidad. Si se rompe alguna restricción, se agrega un valor de penalización que aumenta el valor de la función de energía.

- Optimización mediante Colonia de Hormigas (Ant Colony Optimization - ACO): Es un método metaheurístico que resuelve problemas combinatorios y trata de imitar el comportamiento de comunicación de las hormigas. El procedimiento de ACO ha sido utilizado, exitosamente, en la solución del problema del vendedor viajero, el problema de la asignación cuadrática y el enrutamiento en las redes de

[25] PESCH, E., TETZLAFF, U. A. W. Constraint Propagation Based Scheduling of Job Shops. En: Journal on Computing, Spring Vol. 8(2), 1996, pp. 144-157.

comunicación. Sin embargo, aplicado al JSP el algoritmo no encuentra buenas soluciones debido a que puede llegar a estancarse en las mismas soluciones generadas por el excesivo crecimiento de feromonas en algunos arcos o muy poco en otros arcos[26].

b) Búsqueda local: Basado en la suposición de que es posible encontrar una secuencia de soluciones ligeramente diferente a la inmediatamente anterior[27]. Éste método agrega iterativamente pequeños cambios (perturbaciones) a la programación inicial la cual es obtenida por cualquier heurística. Es decir, comienzan con una solución inicial y la van mejorando progresivamente. El algoritmo finaliza hasta que ya no encuentre ninguna mejora en la función objetivo. El atractivo de la búsqueda local se debe a su amplia aplicación y a su baja complejidad empírica[28].

- Técnica de Escalada (*Hill Climbing*): Inicia con una solución generada aleatoriamente. Se dirige hacia el vecino con el mejor valor evaluado y si encuentra un mínimo local comienza de nuevo con otra solución. La búsqueda se detiene cuando no detecta ninguna mejora en la siguiente operación, lo que hace que tienda a detenerse en el óptimo local. Para evitar que se quede atrapado en el mismo óptimo local se han hecho nuevas modificaciones como, generar un conjunto de posibles soluciones iniciales (método del descenso más rápido) y usar un criterio de aceptación que permita que la función objetivo se degrade temporalmente, y así el proceso de búsqueda escape del mínimo local.

➢ *Métodos de búsqueda de vecindario (Neighborhood search methods):* Proveen buenas soluciones y ofrecen posibilidades de mejorar al ser combinados con otras heurísticas. Entre las técnicas que pertenecen a ésta familia se

[26] SEO, M y VENTURA, J. A. Ant Colony Optimization for Job Shop Scheduling. Proceedings of the 2008 Industrial Engineering Research Conference, pp. 1433-1438.

[27] PEÑA, V y ZUMELZU, L. Op. Cit.

[28] BLAZEWICKZ, Jacek y DOMSCHKE, Wolfgang y PESCH, Erwin. Op. Cit. pp. 16.

encuentran, la búsqueda tabú, el recocido simulado y los algoritmos genéticos. Cada una de las cuales tiene sus propios métodos de perturbación, reglas de parada y métodos para evadir un óptimo local.

- Búsqueda Tabú (Tabu Search – TS): Se basa en una solución que es modificada sucesivamente mediante desplazamientos que, aunque no siempre mejoran la función objetivo, tienden a desplazar la solución a las regiones más cercanas al óptimo evitando que caiga en un óptimo local. Los métodos de TS han ido evolucionando a marcos más avanzados que incluyen mecanismos de memoria a término más amplios (Programación de Memoria Adaptiva - AMP). Han sido aplicados satisfactoriamente al JSP y se ha demostrado, por medio de extensos estudios comparativos, que la búsqueda tabú es superior a otros métodos tales como el recocido simulado, algoritmos genéticos, redes neuronales y otros métodos de búsqueda local[29].

- Recocido Simulado (*Simulated annealing - SA*): Imita la tendencia de los materiales metálicos a alcanzar estructuras de mínima energía a medida que disminuye la temperatura. El estado actual del sistema termodinámico es análogo a la solución actual de la programación. La ecuación de la energía para el sistema termodinámico es análoga a la función objetivo y el estado fundamental es análogo al del óptimo global. Usando esta analogía, la técnica genera aleatoriamente nuevas programaciones al tomar muestras de la distribución de probabilidad del sistema[30]. Aplicado al JSP, esta técnica requiere de altos tiempos computacionales debido a que deben realizarse muchas corridas hasta encontrar buenas soluciones y no es capaz de hacerlo rápidamente. También, suele combinarse con otras metaheurísticas en aras de mejorar los resultados y disminuir el tiempo computacional.

[29] GLOVER, F. Tabu Search and Adaptive Memory Programming - Advances, Applications and Challenges. Interfaces in Computer Science and Operations Research, 1996.
[30] JONES, A y RABELO, L. C. Op. Cit.

- Algoritmos Genéticos (Genetic algorithms, GA): Se basan en un modelo abstracto de la evolución natural y las mutaciones en la reproducción biológica. Imitan el procedimiento de selección natural sobre el espacio de soluciones del problema considerado. Parten de una solución inicial de cromosomas generados aleatoriamente que representan una posible solución al problema. Los cromosomas se evalúan mediante una función fitness que indica el grado de adaptación del individuo al entorno. Esta técnica se puede combinar con otras técnicas como la búsqueda tabú o el recocido simulado[31].

➢ *Métodos basados en el espacio del problema*: Son heurísticas de dos niveles que generan diferentes soluciones iniciales al usar técnicas constructivas específicas más rápidas que luego son mejoradas por la búsqueda local.

- Búsqueda multi-arranque: Tiene dos fases, construcción de la solución y aplicación de un método de búsqueda sobre ésta hasta que se alcance el criterio de parada. Puede no llegar al óptimo local y su uso se limita al campo de los métodos heurísticos.

- GRASP (Greedy Randomized Adaptative Search Procedure): Es una metaheurística que combina procedimientos constructivos y de búsqueda local. En GRASP la construcción de la solución se caracteriza por ser dinámica y aleatoria[32]. Pero aplicado al JSP, el algoritmo no se ajusta correctamente al problema debido a su naturaleza aleatoria.

c) *Large Step Optimization*. Incluye una transición de Optimización de Grandes Pasos -por la aplicación de técnicas específicas del problema que permitan que el óptimo local mejore- y un método de búsqueda local (pasos pequeños) -usualmente por mejora iterativa o SA-. Requiere de un tiempo computacional

[31] SIERRA, María Rita. Op. Cit.
[32] JAIN, A.S y MEERAN, S. Op. Cit. pp. 22.

extenso pero provee mejores soluciones que al aplicar solamente la técnica de búsqueda local.

d) Algoritmos de Umbral (*Threshold Algorithms-TA*): Cambian de configuración si la diferencia de costo entre la solución y un vecino es menor al de un umbral dado.

- Mejoramiento Iterativo (*Iterative Improvement - IM*): Es la clase más simple de la búsqueda local y es la base de otros métodos más elaborados. Comienza con una programación completa preliminar que se mejora a través de varios pasos iterativos. Sólo acepta las mejores configuraciones, ya que su umbral es Cero, y dirige la búsqueda a un óptimo local donde la solución es tan buena o mejor que todas las soluciones en el vecindario.

- Aceptación de Umbral (*Threshold Acceptance - TA*): Simplifica el recocido simulado al introducir un umbral determinista que permite aceptar una solución -si la diferencia con la solución actual es igual o más pequeña al umbral- que es no negativo. Este método sólo ha sido aplicado por Aarts *et al*[33] donde se aplica un vecindario con 30 conjuntos de umbrales diferentes a un problema de 10x10 para determinar los valores adecuados de los parámetros.

e) Computación evolutiva. Constituye un conjunto de heurísticas emergentes que han sido utilizadas con éxito en la resolución de una amplia gama de problemas en las áreas de optimización combinatoria. Estas técnicas imitan el proceso biológico de adaptación de los organismos vivos al entorno. Son algoritmos de carácter probabilista que mantiene una población de individuos o cromosomas. Cada individuo representa una solución potencial del problema y éstos se evalúan mediante la función fitness que determina la calidad de la solución.

- Búsqueda Dispersa (*Scatter Search – SS*): Mediante la combinación de la información sobre la calidad de un conjunto de reglas, restricciones o soluciones,

[33] AARTS, E.H.L., *et al*. A Computational Study of Local Search Algorithms for Job-Shop Scheduling. En: ORSA Journal on Computing, 1994, Vol. 6 (2), pp. 118-125.

se puede obtener una mejor solución a las generadas inicialmente. Está fundamentado en las elecciones estratégicas y sistemáticas de un conjunto pequeño de soluciones. Genera buenos resultados al aplicarse al JSP.

- Colonia de Abejas (*Honey Bee Colony*): Las abejas buscan en el ambiente por fuentes de comida y comparten la información con otras abejas cuando regresan a la colmena. Se caracteriza por la gran organización, adaptación y robustez. Se ha aplicado al JSP logrando buenos resultados.

- Optimización por Enjambre de Partículas (*Particle Swarm Optimization – PSO*): Es una rama de la inteligencia computacional que ha demostrado ser prometedora para resolver los problemas de optimización. El algoritmo PSO fue modelado según el comportamiento de vuelo de un grupo de aves y es considerado, actualmente, una de las técnicas de convergencia más rápidas de la inteligencia computacional y un algoritmo muy ajustable a funciones multimodales (Song and Gu, 2004). Ha sido aplicada a otros problemas de scheduling como el problema de Flow Shop (FSP), el Flexible Job Shop Scheduling (FJSP) y el Open Job Shop, entre otros.

4.3.3 Híbridos.

Definitivamente, son de los mejores métodos que están logrando las mejores soluciones al problema del JSP. Los híbridos son aquellos que combinan heurísticos de búsqueda local con alguna otra metaheurística. Potencializan sus capacidades entre sí y resuelven un problema dado. Obtienen, la mayoría de las veces, mejores resultados que los resultados obtenidos por un solo método.

4.3.4 Hiper-heurísticas.

Debido a que cada heurística tiene diferentes fortalezas y debilidades se hace necesario conocer si se pueden combinar con otras. En aras de que una heurística compense las debilidades de la otra. Son métodos que trabajan con un espacio de búsqueda de heurísticas y no con un espacio de búsqueda de soluciones como lo hacen las metaheurísticas[34]. El objetivo es elevar el nivel de generalización en que la mayoría de las metaheurísticas funcionan[35]. La idea principal es usar miembros de un conjunto conocido y razonable de heurísticas para transformar el estado de un problema. Es decir, en cada paso del problema se va escogiendo una heurística apropiada para cambiar, gradualmente, del estado inicial del problema al estado resuelto del mismo. El enfoque resultante debería ser más económico y fácil de implementar donde se requiera menos nivel de experticia en el problema o en los métodos heurísticos. Además de que se pueda manejar un amplio rango de instancias en diferentes dominios. Aplicado al JSP, han surgido algunas aplicaciones que tratan de solucionar cada parte del problema de manera diferente y no de encontrar un algoritmo poderoso que solucione todo los problemas de éste. Es un concepto emergente, relativamente, que permitiría obtener soluciones en tiempos computacionales más aceptables y más económico de implementar para las empresas.

[34] BURKE, Edmund, HART, Emma, KENDALL, Graham, NEWALL, Jim, ROSS, Peter y SCHULENBURG, Sonia. Hyper-Heuristics: An Emerging Direction in Modern Search Technology. Handbook of Metaheuristics (F. Glover and G. Kochenberger, eds.), Kluwer, 2003, pp. 457-474.

[35] BURKE E.K., LANDA SILVA J.D., SOUBEIGA E. Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling, En: Meta-heuristics: Progress as Real Problem Solvers, Selected Papers from the 5th Metaheuristics International Conference (MIC 2003), 2005, pp 129-158.

5. DESCRIPCIÓN DEL MÉTODO PARTICLE SWARM OPTIMIZATION

Optimización por Enjambre de Partículas *(Particle Swarm Optimization - PSO*) es una técnica de la computación evolutiva que imita el comportamiento de vuelo de las aves y su forma de intercambio de información. El PSO fue introducido por James Kennedy y Russell Eberhart en 1995 para la optimización de funciones continuas no lineales inspirado en el comportamiento social de un grupo de aves. Tiene sus raíces en la *vida artificial* y en la *computación evolutiva.* Conceptualmente, se encuentra entre los algoritmos genéticos y la programación evolutiva.

Computación evolutiva: El término computación evolutiva comprende un conjunto de técnicas inspiradas en los principios de la teoría Neo-Darwiniana de la evolución natural. Trabajan sobre una población de soluciones candidatas para el problema, las cuales siguen los principios darwinianos de la evolución natural, produciendo así mejores soluciones al problema. Las soluciones potenciales se evalúan mediante una función de adecuación o función *fitness* que toma en cuenta el problema que se plantea resolver[36]. Las técnicas de computación evolutiva emulan el proceso biológico de adaptación de los organismos vivos al entorno y las condiciones del medio, aplicándolo a la resolución de problemas en diversas áreas[37].

El PSO se encuentra en la categoría de Inteligencia por Enjambre (*Swarm Intelligence*), la cual es una estrategia de optimización que se basa en el conocimiento de muchos agentes que interactúan localmente en un ambiente para encontrar una solución global a un problema dado. No hay un control sobre éstos

[36] NESMACHNOW, S. Algoritmos Genéticos Paralelos y su Aplicación al Diseño de Redes de Comunicaciones Confiables, Capítulo 2: Técnicas De Computación Evolutiva. (Tesis de Maestría en Informática), Montevideo, Uruguay, 2004.
[37] Ibid.

agentes y éstos interactúan aleatoriamente intercambiando información acerca del entorno[38].

El PSO puede ser usado para resolver muchos de los mismos tipos de problemas que los Algoritmos Genéticos (GAS). La diferencia es que en el PSO la interacción entre el grupo mejora en vez de retraer el progreso de la solución. El PSO tiene memoria, los individuos que vuelan más allá de la solución óptima son obligados a devolverse a ésta y todas las partículas retienen el conocimiento de las buenas soluciones encontradas[39].

5.1 PARTICLE SWARM OPTIMIZATION VERSIÓN ORIGINAL

La versión original del PSO fue presentada para la optimización de funciones no lineales en un espacio continuo. Comprende un concepto simple y se puede implementar en unas pocas líneas de código en un computador. Según sus autores, es computacionalmente económico en términos de memoria y velocidad.

Entre los conceptos básicos se encuentran las *partículas*, que son las posibles soluciones al problema y el *enjambre,* que es la población de partículas.

El sistema del PSO inicia con una población (*enjambre*) de soluciones generadas aleatoriamente buscando la solución óptima al actualizar las soluciones generadas. Cada individuo o solución potencial (p*artícula*) vuela en el espacio dimensional del problema con una velocidad que se ajusta, dinámicamente, de acuerdo a las experiencias de vuelo de ésta y de la de sus colegas. En el PSO, las

[38] YEN, G. G. y IVERS, B. Op. Cit.

[39] EBERHART, R y KENNEDY, J. A new optimizer using particle swarm theory. En: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, 1995, pp. 39–43.

soluciones potenciales están “volando” a través del espacio del problema al seguir las partículas óptimas actuales[40].

En términos más simples, el PSO consiste en un enjambre de partículas en el cual cada partícula tiene una *posición* en el espacio de búsqueda. La calidad de la posición de la partícula es evaluada por la función *fitness* especificada (función objetivo o criterio de optimización). Las partículas vuelan por el espacio de búsqueda con una *velocidad*, que es influenciada por la *mejor posición* encontrada por *él* hasta el momento y por la *mejor posición* encontrada por sus *vecinos*. Después de que se alcanza el número de iteraciones especificado u otro tipo de criterio de parada, el enjambre converge a la *solución más apropiada (*cercana a la *óptima)* para el problema.

El conjunto de partículas en movimiento se encuentra inicialmente dentro del espacio de búsqueda. Cada partícula tiene las siguientes características:

- Tiene una posición y una velocidad.
- Conoce su posición, y el valor de la función objetivo de su posición.
- Recuerda su mejor posición, que se denota como *pbest.*
- Conoce a sus vecinos, la mejor posición anterior, la mejor posición global que se denota como *gbest* y el valor de la función objetivo.

En cada iteración el comportamiento de una partícula dada es un compromiso entre tres escenarios posibles:

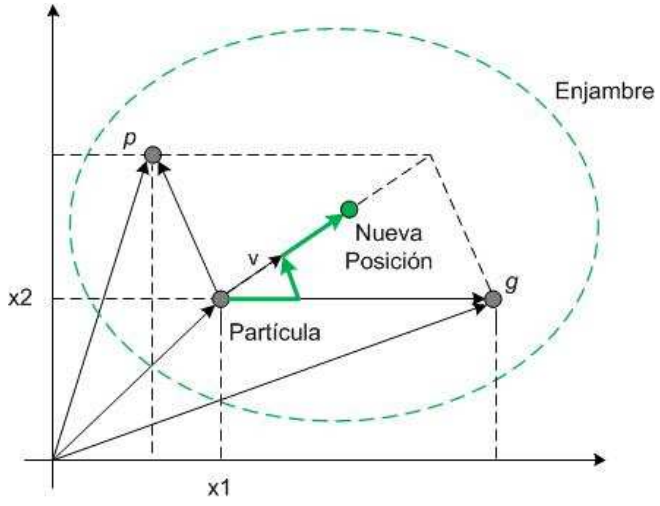
- Seguir su propio camino.
- Devolverse a su mejor posición anterior.
- Devolverse a la mejor posición del vecino, o hacia el mejor vecino.

El comportamiento de una partícula se explicará en el plano cartesiano de la siguiente manera (Ver Figura 15.): Se tiene una partícula en una posición aleatoria

[40] LIAN, Z., JIAO, B y GU, X. Op. Cit.

(x_1, x_2). El mejor de los vecinos de esta partícula es g (*gbest*) y la mejor posición de la partícula es p (*pBest*). La partícula tiene tres posibilidades, irse para g, irse para p ó seguir explorando el espacio para llegar a una mejor posición entre los dos valores anteriores. Las constantes c_2 y c_3 se usan para acortar el espacio de búsqueda y evitar pasar el óptimo global.

Figura 15. Compromiso psicosocial de la partícula.



Fuente: Autor

Conceptualmente, el *pbest* (*personal best*) se asemeja a la memoria autobiográfica donde cada partícula recuerda su experiencia. En otras palabras, es la mejor solución (de acuerdo al valor *fitness*) alcanzada por la partícula hasta el momento.

Conceptualmente, el *gbest (global best)* es un estándar que las partículas quieren alcanzar. El *gbest* es el mejor valor obtenido de la función *fitness* y la respectiva posición obtenida por cualquier partícula del enjambre.

Este compromiso es formalizado por las siguientes ecuaciones:

$$v_{t+1} = c_1 v_1 + c_2 \left(p_{i,t} - x_t \right) + c_3 (p_{g,t} - x_t) \quad (6)$$

$$x_{t+1} = x_t + v_{t+1} \quad (7)$$

Donde,

- v_t, es la velocidad en el tiempo t
- x_t, es la posición en el tiempo t
- $p_{i,t}$, es la mejor posición anterior en el tiempo t
- $p_{g,t}$, es la mejor posición anterior del vecindario (o mejor vecino) en el tiempo t
- c_1, es un coeficiente constante
- c_2 y c_3 son los coeficientes de confianza social y cognitiva

En la Ecuación (6) cada uno de los tres coeficientes de la parte derecha de la ecuación tiene la siguiente interpretación:

- La autoconfianza, esto es, qué tanto cree la partícula *en sí misma.*
- La experiencia*,* es qué tanto cree en su *experiencia* pasada.
- La relación social, es decir, qué tanto cree en la experiencia de sus *vecinos.*

El algoritmo del PSO compuesto por las ecuaciones (6) y (7) ha sido mejorado por múltiples autores al introducir nuevos elementos como el número de iteraciones, el peso inercial y los números aleatorios.

Para cada partícula r y de dimensión s, la nueva *velocidad* v_s^r y la posición x_s^r puede calcularse por las siguientes ecuaciones:

$$v_s^r(\tau) = w \times v_s^r(\tau-1) + c_1 \times rand_1 \times \left[pbest_s^r(\tau-1) - x_s^r(\tau-1) \right] + c_2 \times rand_2 \times \left[gbest_s^r(\tau-1) - x_s^r(\tau-1) \right] \quad (8)$$

$$x_s^r(\tau) = x_s^r(\tau-1) + v_s^r(\tau-1) \quad (9)$$

Donde,

- τ, es el número de iteraciones
- w, es el peso inercial usado para controlar la exploración y la explotación.
    a. Un valor grande mantiene a las partículas moviéndose a gran velocidad y previene que queden atrapadas en un óptimo local.
    b. Un valor pequeño, asegura una velocidad baja y anima a la partícula a explotar la misma área de búsqueda.
- Las constantes *c1* y *c2* son coeficientes de aceleración para determinar si las partículas prefieren moverse cerca de las posiciones de *pbest* o *gbest*.
- Los números *rand1* and *rand2* son dos números uniformemente distribuidos entre 0 y 1.

La velocidad de las partículas en cada dimensión está sujeta a la máxima velocidad v_{max}. Si la suma de las aceleraciones causa que la velocidad en otra dimensión exceda la v_{max} (parámetro especificado por el usuario) entonces, la velocidad de la dimensión está limitada a la v_{max} [41].

A continuación, se describe el procedimiento básico del PSO presentado por Eberthart *et al* (2001):

1) Iniciar una población de partículas con posición y velocidad generadas aleatoriamente en d dimensiones en el espacio del problema. La población se representa mediante una matriz.

[41] Ibid.

2) Para cada partícula, evaluar la función *fitness* de optimización que depende de d variables.

3) Comparar el valor de la función *fitness* de la partícula con el *pbest* de la misma. Si el valor actual es mejor que el *pbest*, asignar al *pbest* el valor actual y la ubicación del *pbest* igual a la posición en el espacio *d*-dimensional.

4) Comparar la evaluación del *fitness* con el promedio del mejor valor anterior de la población. Si el valor actual es mejor que el *gbest*, reajuste *gbest* al valor y el índice de la matriz de la partícula actual.

5) Cambiar la velocidad y la posición de la partícula de acuerdo a las ecuaciones (8) y (9), respectivamente.

6) Volver al paso 2) hasta que se cumpla cierto criterio dado con anterioridad. Usualmente, el criterio consiste en obtener un buen valor de la función *fitness* o alcanzar un número máximo de iteraciones (generaciones).

Básicamente el PSO, en cada iteración, actualiza la velocidad de cada partícula hacia la ubicación *pbest* y *gbest* (versión global) y la pondera por un término aleatorio generado por la aceleración hacia las posiciones *pbest* y *gbest*[42].

Existe una versión "local" del PSO donde las partículas tienen información del mejor valor de ellas y de sus vecinos y no de todo el enjambre. De igual manera las partículas se mueven hacia puntos definidos por el *pbest* y el *lbest (local best*, donde, *lbest* es el índice de la partícula con la mejor evaluación en el vecindario de la partícula.

5.1.1 Parámetros a especificar en el PSO[43].

Para el buen desempeño del algoritmo PSO se deben especificar una serie de parámetros que se describen a continuación:

[42] Ibid.
[43] Ibid.

a) El parámetro de la velocidad máxima *Vmax* se usa para afinar la búsqueda de las regiones entre la posición actual y la posición objetivo (el mejor hasta el momento).

- Si $v_{\max}$ es muy grande, las partículas volarán más allá de las buenas soluciones.
- Si $v_{\max}$ es muy pequeño, las partículas no explorarán más allá de las buenas regiones locales y pueden quedarse atrapadas en un óptimo local sin moverse a encontrar una mejor solución en el espacio de búsqueda. Se fija como el 10-20% del rango en el que se mueve la variable en cada dimensión.

b) El ajuste a las constantes c1 y c2 cambia la cantidad de "tensión" en el sistema. Usualmente equivalen a 2 (en la mayoría de las aplicaciones).

- Valores pequeños permiten que la partícula vaya más lejos de las regiones objetivo antes de ser haladas (tiradas) de nuevo a la región óptima.
- Grandes valores resultan en un movimiento abrupto hacia, o más allá, de las regiones objetivo.

5.1.2 Factor de constricción (*Constriction Factor*).

El factor de constricción K se usa para asegurar la convergencia en el PSO, el cual es una función de c1 y c2. El factor fue propuesto para iniciar la fundamentación matemática del PSO.

$$v_{id} = K*[v_{id} + c_1*rand()*(p_{id} - x_{id}) + c_2*rand()*(p_{gd} - x_{id})]$$
$$K = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \quad \varphi = c_1 + c_2, \quad \varphi > 4 \tag{10}$$

$$\varphi = 4.1$$

Usualmente, $K = 0.729$

Inicialmente, se usaba para no tener que limitar a $V_{\max}$ pero estudios posteriores de Eberhart (2001) concluyeron que al limitar $V_{\max}$ a $x_{\max}$ y hallar w, c1 y c2 con las ecuaciones anteriores se logra una mejor convergencia.

5.1.3 Ventajas y desventajas del PSO.

Entre las ventajas que sobresalen del PSO se encuentra que es simple de implementar, tiene pocos parámetros para ajustar, es muy eficiente en la búsqueda global y provee buenas soluciones al problema del JSP. Entre las desventajas, tiende a una rápida y prematura convergencia en los puntos medios óptimos y tiende una habilidad local débil.

5.1.4 Eficiencia de Pareto (*Pareto Optimality*).

La toma de decisiones en problemas de scheduling, usualmente, involucra la optimización de un solo objetivo pero también es necesario considerar los conflictos que se generan entre dos o más objetivos. El conjunto de soluciones de Pareto contiene soluciones en la que ninguna otra solución desmejora las soluciones de las otras funciones objetivo.

Sea $x = (x_{1,} x_2, ..., x_m)$ y se tiene un problema de optimización con múltiples funciones objetivo $f_1(x_1, x_2, ..., x_m), ..., f_n(x_1, x_2, ..., x_m)$.

$$Minimizar: \quad F(x) = (f_1(x), f_2(x), ..., f_n(x)) \text{, donde } x \in \Re^m, \ F(x) \in \Re^n \quad (11)$$

[44] SHA, D. Y., LIN, H. H. A Particle Swarm Optimization for Multiobjective Flow-Shop Scheduling. En: The International Journal of Advanced Manufacturing Technology, Vol. 45, No. 7-8, pp. 749-758.

Una solución p del problema (11) domina una solución q, si y sólo si,

$$f_k(p) \leq f_k(q) \qquad \forall\, k \in \{1,2,...,n\}$$

y existe un $k \in \{1,2,...,n\}$ tal que $f_k(p) < f_k(q)$

La solución no-dominada es una solución que domina a otra pero no a sí misma. Una solución p se llama la solución de Pareto si no existe ninguna otra solución q en el espacio factible que pudiera dominar a p. El conjunto que incluye todas las soluciones de Pareto se llama el Conjunto óptimo de Pareto o el conjunto eficiente. Las soluciones que genera el conjunto de Pareto son soluciones (es decir, no es una única solución) no-dominadas o no-inferiores. En la mayoría de los casos, con los problemas de optimización multi-objetiva se prefiere buscar un grupo de soluciones de Pareto para el problema de decisión.

6. PARTICLE SWARM OPTIMIZATION APLICADO AL JOB SHOP SCHEDULING PROBLEM

Como se mencionó con anterioridad, el PSO fue diseñado para un espacio de solución continuo, pero hoy en día se encuentran variaciones del PSO para aplicarlo a problemas de optimización combinatoria. El PSO ha sido aplicado a problemas tales como el Problema del Vendedor Viajero (Traveling Salesman Problem –TSP) (Pang *et al*. 2004), el Problema del Ruteo de Vehículos (Chen *et al.*, 2006) y a problemas de scheduling (Liao *et al.,* 2007; Lian *et al*., 2006; entre otros).

Dado al espacio solución discreto de los problemas de scheduling y al espacio continuo del PSO tradicional se hace necesario realizar modificaciones –para adecuar el PSO al espacio discreto- para aplicarlo al JSP. Es por esto, que para el

desarrollo del presente proyecto se toma en cuenta el trabajo propuesto por Sha y Lin (2010) en el que se modifica la representación, el movimiento y la velocidad de la partícula para así, aplicarlo eficientemente al JSP con múltiples objetivos.

6.1 POSICIÓN DE LA PARTÍCULA

Para representar un *schedule* es necesario realizar una codificación al espacio de partículas. La codificación se basa en una lista de preferencias (*preference list-based - PL*) propuesta por Davis (1985). La lista usa una cadena de operaciones por cada máquina. Es decir, para cada máquina hay una lista de preferencia de 1 a n, donde n es el número de trabajos. Para un problema nxm, la posición de la partícula k se representa a través de una matriz mxn donde la j-ésima fila es la lista de preferencia de la máquina j, por ejemplo:

$$X^k = \begin{bmatrix} x_{11}^k & x_{21}^k & \cdots & x_{n1}^k \\ x_{12}^k & x_{22}^k & \cdots & x_{n2}^k \\ \vdots & \vdots & & \vdots \\ x_{1m}^k & x_{2m}^k & \cdots & x_{nm}^k \end{bmatrix}$$, donde $x_{ij}^k \in \{1, 2, ...n\}$ denota el trabajo en la ubicación i en la lista de preferencia de la máquina j.

El método trabaja con *schedules* activos lo que permite obtener *schedules* óptimos. Para decodificar la posición de una partícula en un *schedule* activo los autores Sha y Lin (2010) implementan el método heurístico de Giffler & Thompson (1960), método que se usa como un intérprete para modificar la matriz de secuencia de cualquier trabajo en un *schedule* activo en un diagrama de Gantt[45].

[45] KOBAYASHI, S., ONO, I., YAMAMURA, M. An Efficient Genetic Algorithm for Job Shop Scheduling Problems. En: Proc. of ICGA'95, 1995, pp.506-511.

6.1.1 Método heurístico de Giffler & Thompson (G&T).

La notación que emplea el método se describe a continuación:

- (i, j), operación del trabajo i que debe ser procesado en la máquina j
- S contiene el *schedule* parcial que contiene las operaciones a programar.
- Ω es el conjunto de operaciones que pueden ser programadas. Inicia con las operaciones que no tienen predecesores.
- $s_{(i,j)}$, es el tiempo más temprano en el que la operación (i, j) que pertenece a Ω puede iniciar.
- $p_{(i,j)}$, es el tiempo de procesamiento de la operación (i, j)
- $f_{(i,j)}$, es el tiempo más temprano en el que la operación (i, j) que pertenece a Ω puede finalizar, $f_{(i,j)} = s_{(i,j)} + p_{(i,j)}$

Algoritmo G&T:

Paso 1. Inicializar $S = \varnothing$ y Ω quien, inicialmente, contiene las operaciones sin predecesor.

Paso 2. Hallar los tiempos de finalización f_{ij} de las operaciones que pertenecen a Ω. Escoger el mínimo valor f_{ij} de las operaciones de Ω y denotarlo como $f^* = \min_{(i,j)\in\Omega}\{f_{(i,j)}\}$. Asignar la máquina m^*, a la máquina a la que corresponde el mínimo valor f^*.

Paso 3.

1) Identificar el conjunto de operaciones $(i', j') \in \Omega$ que también necesiten la máquina m^* y que cumplan $s_{(i',j')} < f^*$

2) Escoger la operación (i, j) del conjunto del paso anterior que tenga la mayor prioridad (1>2>3>n).

3) Agregar la operación (i, j) escogida al conjunto S

4) Asignar a $s_{(i,j)}$ el tiempo de finalización de (i, j), $s_{(i,j)} \leftarrow f_{(i,j)}$

Paso 4. Si se ha generado un *schedule* completo, el algoritmo G&T se detiene. Si no, se debe eliminar la operación (i, j) del conjunto Ω e incluir el sucesor inmediato en Ω y seguir con el paso 2).

A continuación se muestra un ejemplo de cómo funciona el algoritmo de G&T en un problema $3x3$. El ejercicio completo se encuentra en el Anexo A.

Ejemplo 3. Se tiene un problema de 3 trabajos y 3 máquinas de la siguiente manera:

- El trabajo 1 (se denota como J1) debe pasar por las máquinas 2-3-1 con tiempos de procesamiento 5, 2 y 3 respectivamente.
- El trabajo 2 (se denota como J2) debe pasar por las máquinas 1-2-3 con tiempos de procesamiento 2, 4 y 1 respectivamente.
- El trabajo 3 (se denota como J3) debe pasar por las máquinas 1-3-2 con tiempos de procesamiento 1, 4 y 3 respectivamente.

Esta información se presenta por medio de la Tabla 5.

Tabla 5. Ejemplo de una instancia 3×3 del JSP.

Trabajo	Secuencia máquinas	Tiempos de procesamiento
1	2,3,1	$p_{12}=5, \quad p_{13}=2 \quad p_{11}=3$
2	1,2,3	$p_{21}=2, \quad p_{22}=4, \quad p_{23}=1$
3	1,3,2	$p_{31}=1, \quad p_{33}=4, \quad p_{32}=3,$

Fuente: Instancia FT03.

Se genera una matriz *Posición* $X^k \ _{mxn}$ aleatoria por filas de $[1, n]$ sin repetición para la partícula k, donde x_{ij} representa la prioridad del trabajo i en la máquina j.

$$X^k = \begin{array}{c} \\ m1 \\ m2 \\ m3 \end{array} \begin{array}{c} \begin{array}{ccc} j1 & j2 & j3 \end{array} \\ \begin{bmatrix} 2 & 3 & 1 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix} \end{array}$$

En este algoritmo se trabaja con las operaciones
$(i, j); \quad trabajos\ i = 1, 2, ..., n, \quad máquinas\ j = 1, 2, ..., m$ de cada partícula.
Las operaciones para el Ejemplo 2 se extraen de la Tabla 5 y son:

$$Operaciones = \begin{array}{c} \\ J1 \\ J2 \\ J3 \end{array} \begin{array}{c} Secuencia \\ \left\{ \begin{array}{l} (1,2) \rightarrow (1,3) \rightarrow (1,1) \\ (2,1) \rightarrow (2,2) \rightarrow (2,3) \\ (3,1) \rightarrow (3,3) \end{array} \right\} \end{array}$$

Algoritmo G&T.

- S, es el conjunto que guarda el *schedule* (programación) generado para la partícula. Comienza *vacío*, $S = \varnothing$.
- s_{ij}, es el tiempo de inicio más temprano en el que puede comenzar el trabajo en la máquina. Inicia con valor igual a *cero*, $s_{ij} = 0$.
- Ω, es el conjunto que contiene las operaciones que se van a programar. Al inicio de la iteración contiene las operaciones sin predecesores (primera columna del conjunto de las *operaciones*). Luego, elimina la operación programada y agrega su sucesor inmediato (basándose en el conjunto *operaciones*).

Inicialización:

Paso 1. Con $S=\varnothing$, $s_{ij}=o$, Asignar a Ω las operaciones que no tienen predecesor. Es decir, dirigirse hacia el conjunto *operaciones* y buscar la primera columna.

$$\Omega=\{(1,2),(2,1),(3,1)\}$$

Iteración 1.

Paso 2. Hallar el tiempo de finalización $f_{ij}=s_{ij}+p_{ij}$ para cada operación que se encuentra en Ω. $s_{ij}=0$ y p_{ij} se encuentra en la Tabla 6.

$$f_{12}=0+5=5; \quad f_{21}=0+2=2; \quad f_{31}=0+1=1$$

Hallar el valor mínimo de los valores anteriores, $f^*=\min(f_{ij})$

$$f^*=\min(f_{(1,2)}, f_{(2,1)}, f_{(3,1)})=\min(5,2,1)=1$$

Ahora se mira a qué operación pertenece ese valor mínimo f^* y se asigna m^* igual a la máquina a la que corresponde. Para este ejemplo, el valor $f^*=1$ es de la operación (3,1) que se encuentra en la máquina 1.

$$m^*=1$$

Paso 3. 1) Buscar qué operaciones de Ω también necesitan a m^*, la máquina 1.

$$\{(2,1),(3,1)\}$$

2) Buscar en la matriz *posición* X^k las operaciones anteriores y buscar cuál de ellas tiene mayor *prioridad*, tomando en cuenta que 1>2>…>n.

$$\begin{array}{cccc} & J1 & J2 & J3 \\ X^1= M1 & 2 & 3 & 1 \end{array}$$

La operación (2,1), el trabajo J2 en la máquina M1 tiene prioridad 3.

La operación (3,1), el trabajo J3 en la máquina M1 tiene prioridad 1.

Se escoge la operación (3,1).

3) Asignar al conjunto de S la operación (3,1) ya que tiene la mayor prioridad.

$S = \{(3,1)\}$, se agrega a S como se ilustra en la Figura 16 (a).

4) Cambiar el valor de inicio s_{ij} por el valor de finalización f_{31} de la operación (3,1) así, $s_{(3,1)} = 1$. Hay que tener en cuenta que el nuevo valor de s_{ij} afecta también a todas las operaciones que contengan, ya sea, el trabajo 3 o la máquina 1 para la siguiente iteración.

Paso 4. Actualizar Ω eliminando del conjunto a la operación (3,1) que ya fue programada. Se incluye a su sucesor inmediato, en este caso, la operación (3,3). (Se busca en la el conjunto de *operaciones*, quién sigue después de la operación programada).

$$\Omega = \{(1,2),(2,1),(3,3)\}$$

Iteración 2.

P.2. $f_{(1,2)} = 0+5 = 5; \quad f_{(2,1)} = 1+2 = 3; \quad f_{(3,3)} = 1+4 = 5; \qquad f^* = 3; \quad m^* = 1$

P.3. (1) $\{(2,1)\}$

(2) $(2,1)$

(3) $S = \{(3,1),(2,1)\}$, se agrega a S como se ilustra en la Figura 16 (a).

(4) $s_{(2,1)} = 3$

P.4. $\Omega = \{(1,2),(2,2),(3,3)\}$

Iteración 3.

P.2. $f_{(1,2)} = 0+5 = 5; \quad f_{(2,2)} = 3+4 = 7; \quad f_{(3,3)} = 1+4 = 5;$ cuando hay dos valores mínimos iguales se escoge cualquiera aleatoriamente. $f^* = 5; \quad m^* = 3$

P.3. (1) $\{(3,3)\}$

(2) $(3,3)$

(3) $S = \{(3,1),(2,1),(3,3)\}$, se agrega a S como se ilustra en la Figura 16 (b).

(4) $s_{(3,3)} = 5$

P.4. $\Omega = \{(1,2),(2,2),(3,2)\}$

Iteración 4.

P.2. $f_{(1,2)} = 0+5 = 5; \quad f_{(2,2)} = 3+4 = 7; \quad f_{(3,2)} = 5+3 = 8; \qquad f^* = 5; \quad m^* = 2$

P.3. (1) $\{(1,2),(2,2),(3,2)\}$

(2) $(2,2)$

(3) $S = \{(3,1),(2,1),(3,3),(2,2)\}$, se agrega a S como se ilustra en la Figura 16 (b).

(4) $s_{(2,2)} = 7$

P.4. $\Omega = \{(1,2),(2,3),(3,2)\}$

Iteración 5.

P.2. $f_{(1,2)} = 7+5 = 12; \quad f_{(2,3)} = 7+1 = 8; \quad f_{(3,2)} = 7+3 = 10; \qquad f^* = 8; \quad m^* = 3$

P.3. (1) $\{(2,3)\}$

(2) $(2,3)$

(3) $S = \{(3,1),(2,1),(3,3),(2,2),(2,3)\}$, se agrega a S como se ilustra en la Figura 16 (c).

(4) $s_{(2,3)} = 8$

P.4. $\Omega = \{(1,2),(3,2)\}$

Iteración 6.

P.2. $f_{(1,2)} = 7 + 5 = 12; \quad f_{(3,2)} = 7 + 3 = 10; \qquad f^* = 10; \quad m^* = 2$

P.3. (1) $\{(1,2),(3,2)\}$

(2) $(1,2)$

(3) $S = \{(3,1),(2,1),(3,3),(2,2),(2,3),(1,2)\}$, se agrega a S como se ilustra en la Figura 16 (c).

(4) $s_{(1,2)} = 10$

P.4. $\Omega = \{(1,3),(3,2)\}$

Iteración 7.

P.2. $f_{(1,3)} = 12 + 2 = 14; \quad f_{(3,2)} = 12 + 3 = 15; \qquad f^* = 14; \quad m^* = 3$

P.3. (1) $\{(1,3)\}$

(2) $(1,3)$

(3) $S = \{(3,1),(2,1),(3,3),(2,2),(2,3),(1,2),(1,3)\}$, se agrega a S como se ilustra en la Figura 16 (d).

(4) $s_{(1,3)} = 14$

P.4. $\Omega = \{(1,1),(3,2)\}$

Iteración 8.

P.2. $f_{(1,1)} = 14 + 3 = 17; \quad f_{(3,2)} = 12 + 3 = 15; \qquad f^* = 15; \quad m^* = 2$

P.3. (1) $\{(3,2)\}$

(2) $(3,2)$

(3) $S = \{(3,1),(2,1),(3,3),(2,2),(2,3),(1,2),(1,3),(3,2)\}$, se agrega a S como se ilustra en la Figura 16 (d).

(4) $s_{(3,2)} = 15$

P.4. $\Omega = \{(1,1)\}$

Iteración 9.

P.2. $f_{(1,1)} = 14 + 3 = 17; \qquad f^* = 17; \quad m^* = 1$

P.3. (1) $\{(1,1)\}$

(2) $(1,1)$

(3) $S = \{(3,1),(2,1),(3,3),(2,2),(2,3),(1,2),(1,3),(3,2),(1,1)\}$, se agrega a S como se ilustra en la Figura 16 (d).

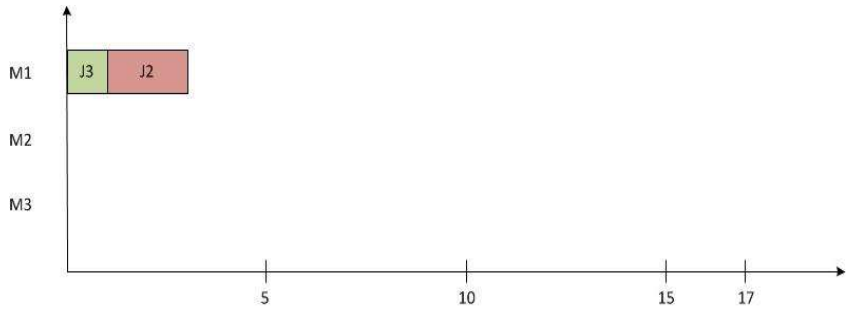
(4) $s_{(1,1)} = 17$

P.4. $\Omega = \{\varnothing\}$

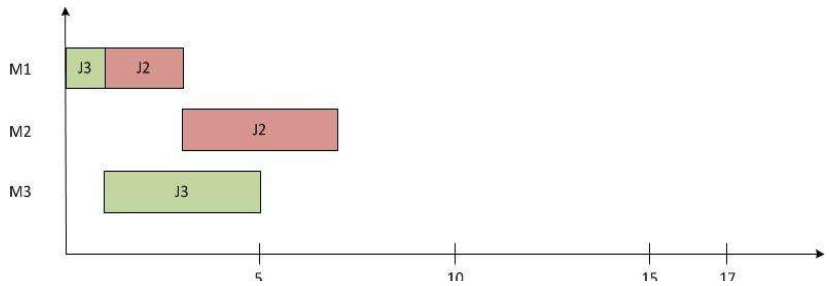
Se ha generado un *schedule* completo, entonces, el algoritmo se detiene.

Figura 16. Decodificación de la posición de la partícula en un schedule

a.

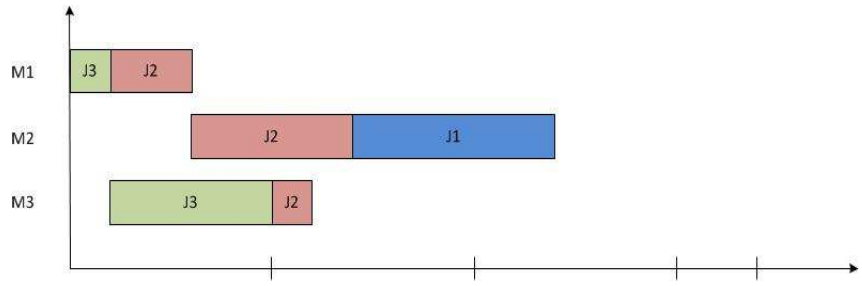


b.

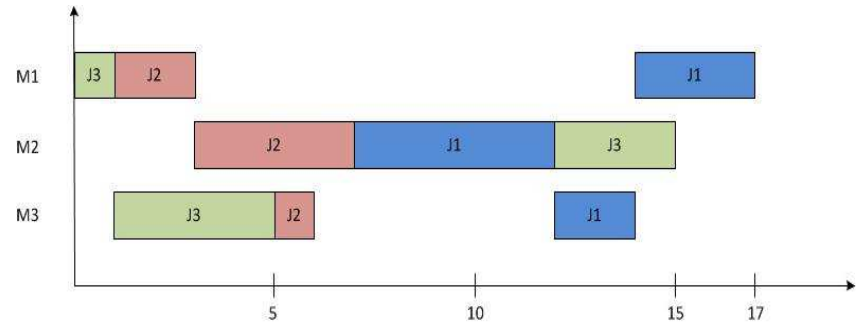


Continuación Figura 16.

c.



d.



Fuente: Autor.

En MOPSO la información que se almacena en $pbest$ y en $gbest$ es el mejor *schedule* S^k generado por el algoritmo de G&T. Mientras que en el PSO original se almacena la mejor posición X^k encontrada hasta el momento.

Entonces, el siguiente paso es generar la matriz S^k;

Se tiene el *schedule* generado anteriormente,

$$S = \{(3,1),(2,1),(3,3),(2,2),(2,3),(1,2),(1,3),(3,2),(1,1)\}$$

Las preferencias se asignan tomando en cuenta el orden en el que se encuentran en el conjunto S y se mira por máquinas. Para la máquina 1 se puede observar que la primera operación en el conjunto S es la operación (3,1), a la cual se le asigna prioridad 1. Luego va la operación (2,1) con prioridad 2 y luego la operación (1,1) con prioridad 3.

$$S = \{(3,1),(2,1),(3,3),(2,2),(2,3),(1,2),(1,3),(3,2),(1,1)\}$$
$$\quad\ 1 \quad\ 2 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 3$$

Así, para las demás máquinas.

$$S = \{(3,1),(2,1),(3,3),(2,2),(2,3),(1,2),(1,3),(3,2),(1,1)\}$$
$$\quad 1 \quad 2 \quad 1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 3$$

La matriz S^k se forma con las prioridades asignadas, así:
La operación (3,1), trabajo 3 en la máquina 1 se le asigna el valor 1.

$$S^k = \begin{array}{c} \\ M1 \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccc} J1 & J2 & J3 \end{array} \\ \begin{bmatrix} 3 & 2 & 1 \\ & & \\ & & \end{bmatrix} \end{array}$$

Así sucesivamente, hasta tener S^k completa. $S^k = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix}$

6.2 VELOCIDAD DE LA PARTÍCULA

En MOPSO, Sha y Lin (2010) sólo toman en cuenta el prevenir que las partículas se queden atrapadas en un óptimo local. Mientras que en el PSO original también

se considera que las partículas se muevan hacia la solución $gbest(pbest)$, esto se hace con la diferencia entre la posición de la partícula y la posición del *gbest/pbest*.

Para un problema nxm, la velocidad de la partícula k se puede representar como una matriz mxn:

$V^k = \begin{bmatrix} v_{11}^k & v_{21}^k & \cdots & v_{n1}^k \\ v_{12}^k & v_{22}^k & \cdots & v_{n1}^k \\ \vdots & \vdots & & \vdots \\ v_{1m}^k & v_{2m}^k & \cdots & v_{nm}^k \end{bmatrix}$, donde v_{ij}^k es la velocidad de la operación O_{ij} de la partícula k, $v_{ij}^k \in \{-1, 0, 1\}$

La velocidad inicial de las partículas es generada aleatoriamente al inicio del algoritmo. El peso inercial w influencia la velocidad de las partículas en el PSO. Para cada partícula k y operación O_{ij} si la velocidad v_j^k es diferente de cero (0) entonces, v_j^k se asignará igual a cero (0) con probabilidad $(1-w)$. De esta manera, si x_{ij}^k aumenta (disminuye), x_{ij}^k ya no crecerá (disminuirá) en esta iteración con una probabilidad de $(1-w)$. Entre mayor sea la inercia w, el valor de la prioridad seguirá creciendo o disminuyendo en más iteraciones y de esta manera es más difícil para la partícula devolverse a su posición anterior.

Continuando con el Ejemplo 3, las velocidades iniciales generadas al inicio del algoritmo PSO son:

$$V^1 = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$$

La velocidad se actualiza al inicio de cada iteración y para cada operación.

Si la velocidad de cada operación v_{ij} es diferente de CERO, $v_{ij} \neq 0$. Un número aleatorio $rand$ es mayor e igual que el peso inercial w, $rand \geq w$, la nueva v_{ij} es igual a CERO, $v_{ij} \leftarrow 0$.

La nueva matriz V^1 que podría resultar para este ejemplo es: $V^1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix}$

6.3 MOVIMIENTO DE LA PARTÍCULA

El movimiento de la partícula es modificado a través del Operador de Cambio (*Swap Operator*) de Sha (2006). Si $v_{ij}^k = 0$, el trabajo i en x_j^k se moverá a la posición $pbest_j^k$ con probabilidad c_1 y a la posición del $gbest_j$ con probabilidad c_2. Donde, x_j^k es la lista de preferencia de la máquina j de la partícula k, $pbest_j^k$ es la lista de preferencia de la máquina j de la mejor solución de las partículas k, $gbest_j$ es la lista de preferencia de la máquina j de la mejor solución $gbest$ y c_1 y c_2 son unas constantes entre 0 y 1 y la suma de las dos es menor que 1, $c_1 + c_2 \leq 1$

El procedimiento es el siguiente:

Paso 1. Escoger aleatoriamente una posición p de x_j^k

Paso 2. Asignar a A_1 el valor encontrado de la posición p

Paso 3. Si un número generado aleatoriamente es menor que c_1, $rand < c_1$ se busca A_1 en $pbest_j^k$; si el número aleatorio está entre $c_1 < rand \leq c_1 + c_2$, se busca A_1 en $gbest_j$. Se asigna esta nueva posición encontrada como p' y el valor encontrado en esta posición como A_2

Paso 4. Si A_2 ya se ha especificado y si, $v_{jA_1}^k = 0$ y $v_{jA_2}^k = 0$, se intercambia A_1 y A_2 en x_j^k, y se cambia la velocidad del parámetro en la posición A_1, $v_{jA_1} = 1$

Paso 5. Si todas las posiciones de x_j^k han sido consideradas, se detiene el procedimiento. De otra manera y considerando si $p < n$, entonces $p \leftarrow p+1$, si no $p \leftarrow 1$. Ir al paso 2.

A continuación se explica el procedimiento en extenso aplicado al Ejemplo 3, además el ejemplo se encuentra completo en el Anexo A.

Se tiene la matriz posición, velocidad, *pbest* y *gbest*, así:

$$X^k = \begin{matrix} m1 \\ m2 \\ m3 \end{matrix}\begin{bmatrix} 2 & 3 & 1 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix}, \quad c_1 = 0,7, \quad c_2 = 0,1, \quad pbest_j^k = \begin{bmatrix} 2 & 3 & 1 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix}, \quad gbest_j = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix},$$

$$V^k = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix},$$

Para la máquina 1:

$p = 3$, el trabajo en la posición 3 en x_1^k es igual a 1, $A_1 = 1$

Como $rand = 0,428 < c_1 = 0,7$, se busca $A_1 = 1$ en $pbest_j^k$, luego $p' = 3$ y $A_2 = 1$. En este caso $A_1 = A_2$, entonces no se realiza el intercambio entre A_1 y A_2.

$p = 1$, el trabajo en la posición 1 en x_1^k es igual a 1, $A_1 = 2$

Como $rand = 0,25 < c_1 = 0,7$, se busca $A_1 = 2$ en $pbest_j^k$, luego $p' = 2$ y $A_2 = 3$.

Como las velocidades $v_{jA_1} = 0$, $v_{jA_2} = 0$ y $A_1 \neq A_2$ se realiza el intercambio entre A_1 y A_2.

$$X_j = [2 \quad 3 \quad 1] \rightarrow \quad X_j = [3 \quad 2 \quad 1]$$

Y la velocidad de $v_{jA_1} = 1$, así:

$$V_1^k = [1 \quad 0 \quad 0] \rightarrow \quad V_1^k = [1 \quad 1 \quad 0]$$

$p = 1 + 1 = 2$, el trabajo en la posición 2 en x_1^k es igual a 2, $A_1 = 2$

Como $rand = 0,486 < c_1 = 0,7$, se busca $A_1 = 2$ en $pbest_j^k$, luego $p' = 1$ y $A_2 = 3$.

Dado que las velocidades $v_{jA_1} = 0$ y $v_{jA_2} = 1$ y $A_1 \neq A_2$, no se realiza el intercambio entre A_1 y A_2.

Como ya se tuvieron en cuenta todas las posiciones, el procedimiento se detiene y continúa con las demás máquinas.

Finalmente, se obtiene la siguiente matriz de posición y de velocidad:

$$X^k = \begin{matrix} m1 \\ m2 \\ m3 \end{matrix} \begin{bmatrix} 3 & 2 & 1 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix}, \; V^k = \begin{bmatrix} 1 & 1 & 0 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}.$$

Este procedimiento se realiza igualmente para todas las partículas del enjambre.

Ahora, se agrega un **Operador de Mutación** (*Mutation Operator*) después de que una partícula se mueve a una nueva posición. Este operador escoge, aleatoriamente, una máquina y dos trabajos de ésta, y los intercambia sin considerar la velocidad que tienen. Este procedimiento es parecido al anterior pero sólo se realiza para una máquina agregando una diferenciación a las posiciones para evitar generar posiciones parecidas.

$M \leftarrow$ número aleatorio entre 1 y m para escoger la máquina

$p \leftarrow$número aleatorio entre 1 y n para escoger la posición

$p' \leftarrow$ número aleatorio entre 1 y n para escoger la segunda posición

Entonces,

$$A_1 \leftarrow x_{Mp}^k; \quad A_2 \leftarrow x_{Mp'}^k$$
$$x_{Mp}^k \leftarrow A_2; \quad x_{Mp'}^k \leftarrow A_1$$
$$v_{mA_1}^k \leftarrow 1; \quad v_{mA_2}^k \leftarrow 1$$

6.4 FUNCIÓN FITNESS

La función *fitness* es una medida de la calidad de las soluciones generadas. Como se quiere optimizar *dos objetivos*, para el primer objetivo se busca el valor s_{ij} de la última iteración y se nombra como $C_{\max}$ (*makespan*), $C_{\max} = s_{ij}$. Para el segundo objetivo se halla la tardanza L_i, $L_i = s_{ij} - d_i$, d_i viene en los datos iniciales del problema para cada trabajo.

*Función fitness***:** Minimizar la Función Objetivo

$$fitness(gbest) = \min\ fitness(pbest^k)$$

6.5 ESTRATEGIA DE DIVERSIFICACIÓN

Si todas las partículas tienen la misma solución no-dominada, pueden quedarse atrapadas en un óptimo local. Para prevenir esto, Sha y Lin (2010) proponen una estrategia de diversificación que permita que las soluciones no-dominadas de las partículas sean diferentes.

La solución $pbest$ de cada partícula no es la mejor solución encontrada por ella si no una de las mejores N soluciones encontradas por el enjambre hasta el momento donde, N es el tamaño del enjambre.

Apenas una nueva solución es generada por las partículas, las soluciones $pbest$ y $gbest$ se actualizan de acuerdo a 3 situaciones:

- Si el valor *fitness* de la partícula (por ejemplo, $C_{\max}(S^k)$) es mejor que el valor *fitness* de la solución $gbest$ ($C_{\max}(gbest)$), se asigna la solución de la partícula como el nuevo $gbest$.
- Si el valor *fitness* de la partícula es igual a cualquier solución $pbest / gbest$, se reemplaza la solución no-dominada ($pbest / gbest$) por la solución de la partícula.
- Si el valor *fitness* de la partícula es peor que la peor solución $pbest^{worst}$ y es diferente a cualquier solución no-dominada, se asigna la peor solución $pbest$ igual a la solución de la partícula.

6.6 DATOS DE ENTRADA Y CRITERIO DE PARADA

Los datos de entrada para el algoritmo son:

- Datos del problema, secuencia y tiempos de procesamiento p_{ij}
- n, número de trabajos
- m, número de máquinas
- w, peso inercial que disminuye linealmente entre mayor número de iteraciones. $w_{\max}, w_{\min}$, peso inercial máximo y mínimo respectivamente.

$$w = w_{\max} - \frac{w_{max} - w_{\min}}{iter_{\max}} * iter \qquad (11)$$

- c_1 y c_2, factores de aprendizaje asignados por el usuario
- Tamaño del enjambre N
- O_{ij}, es la operación del trabajo i en la máquina j
- $iter_{\max}$, es el máximo número de iteraciones, escogido por el usuario.

- *Iter,* es la iteración actual.
- *I,* contador

El criterio de parada del algoritmo que se usa, usualmente, es hasta que éste llega a un límite máximo de iteraciones, el cual es especificado por el usuario.

7. PSEUDOCÓDIGO

INICIO DE MOPSO

Inicializar una población de partículas con posiciones generadas aleatoriamente por filas entre $[1, n]$ sin repetición.

Inicializar las velocidades de cada partícula aleatoriamente entre [-1,1].

Para $k = 1$ hasta *N* ***hacer***

//Aplicar el algoritmo G&T para decodificar la posición de la partícula en un Schedule

$S = \varnothing$

$\Omega \leftarrow operaciones \sin predecesor$

$s_{ij} = 0$

Para cada Operación $O_{ij} \in \Omega$ ***hacer***

$f_{ij} \leftarrow s_{ij} + p_{ij}$

$f^* \leftarrow \min\{f_{ij}\}$

$m^* \leftarrow máquina\ que\ corresponde\ a\ f^*$

Identificar (i', j') que necesite a m^* y que cumpla $s_{ij} < f^*$

Buscar en X^k la operación (i', j') con menor valor entre [1,n] $\leftarrow (i, j)$

$S \leftarrow (i, j)$

Actualizar $s_{ij} = f_{ij}$

Eliminar $(i, j)\ de\ \Omega$

Agregar sucesor inmediato de $(i, j)\ a\ \Omega$

finalizar para

Asignar preferencia al conjunto S

Ubicar en matriz S^k

//Evaluar la función *fitness*

1) $C_{\max} : fitness(S^k) = s_{ij}$

2) $L_i : fitness(S^k) = \sum \max(s_{ij} - d_i)$

$pbest^k \leftarrow S^k$

$fitness(pbest^k) = fitness(S^k)$

$P \leftarrow almacenar\ todos\ los\ pbest^k$

finalizar para

//Escoger el mejor valor del enjambre

1) $fitness(gbest) \leftarrow \min(fitness(pbest^k))$

$gbest \leftarrow S^k \quad (del\ \min\ asociado)$

repetir

//Actualizar la velocidad de las partículas

para cada partícula k y operación O_{ij} ***hacer***

si $(v_{ij} \neq 0\ y\ rand \geq w)$ ***entonces***

$v_{ij} \leftarrow 0$

finalizar si

finalizar para

para cada partícula k ***hacer***

//Mover la partícula *k*

Para $j = 1$ ***to*** *m* ***hacer***

$p \leftarrow rand\ entre\ [1, n]$

Para $i = 1$ ***to*** *n* ***hacer***

$rand \sim U(0,1)$

si $(rand \leq c_1)$ **entonces**

$A_1 \leftarrow x_{jp}^k$

$p' \leftarrow$ *ubicación de* A_1 *en* $pbest_j^k$

$A_2 \leftarrow x_{jp'}^k$

si $(v_{jA_1}^k = 0)\ y\ (v_{jA_2}^k = 0)\ y\ (A_1 \neq A_2)$ **entonces**

$x_{jp}^k \leftarrow A_2$

$x_{jp'}^k \leftarrow A_1$

$v_{jA_1}^k \leftarrow 1$

finalizar si

sino si $(c_1 < rand \leq c_1 + c_2)$ **entonces**

$A_1 \leftarrow x_{jp}^k$

$p' \leftarrow$ *ubicación de* A_1 *en* $gbest_j$

$A_2 \leftarrow x_{jp'}^k$

si $(v_{jA_1}^k = 0)\ y\ (v_{jA_2}^k = 0)\ y\ (A_1 \neq A_2)$ **entonces**

$x_{jp}^k \leftarrow A_2$

$x_{jp'}^k \leftarrow A_1$

$v_{jA_1}^k \leftarrow 1$

finalizar si

finalizar si

$p = p + 1$

finalizar si

finalizar para

finalizar para

//Operador de mutación

$M \leftarrow$ Escoger aleatoriamente una máquina entre 1 y m

$p \leftarrow$ Escoger aleatoriamente una posición entre 1 y n

$p' \leftarrow$ Escoger aleatoriamente otra posición entre 1 y n

$A_1 \leftarrow x^k_{Mp}; \quad A_2 \leftarrow x^k_{Mp'};$

$x^k_{Mp} \leftarrow A_2; \quad x^k_{Mp'} \leftarrow A_1;$

$V^k_{MA_1} \leftarrow 1; \quad V^k_{MA_2} \leftarrow 1;$

//Aplicar de nuevo el algoritmo de G&T para hallar el nuevo S^k

//Asignar como el peor *pbest* al mayor valor *fitness pbest* de la partícula

$fitness(pbest^{worst}) \leftarrow \max(fitness(pbest^k))$

// Aplicar la estrategia de diversificación para actualizar *pbest, gbest* y el conjunto *P*.

si $(fitness(S^k) < fitness(gbest)$

$pbest^{worst} \leftarrow gbest$

$gbest \leftarrow S^k$

sino si $(fitness(S^k) \leq fitness(pbest^{worst})$ ***entonces***

$pbest^{worst} \leftarrow S^k$

si $(fitness(S^k) = fitness(gbest)$ ***entonces***

$gbest \leftarrow S^k$

sino

para $k' \leftarrow 1 \quad hasta \quad N$ **entonces**

si $fitness(S^k) = fitness(pbest^k)$ **entonces**

$pbest^{k'} \leftarrow S^k$

break

finalizar si

finalizar para

finalizar si

finalizar si

//Almacenar los valores *gbest* generados en el conjunto de soluciones factibles *F*

$F \leftarrow fitness\,(gbest)\,de\,1)\,y\,2)$ de los objetivos a analizar

//Comparar todos los valores de las funciones objetivo y determinar las soluciones no-dominadas.

si $fitness(f_1) \leq fitness(f_1)$ ***entonces***

// f_1 domina a f_2

Asignar la solución al conjunto de soluciones no-dominadas *ND.*

$ND \leftarrow f_1$

sino si $fitness(f_2) \leq fitness(f_1)$ ***entonces***

$ND \leftarrow f_2$

// Pareto Externo, el conjunto de las soluciones no-dominadas entra al conjunto Eficiente de Pareto

$EP \leftarrow ND$

//Cuando una nueva solución no-dominada es generada y es mejor que alguna solución del conjunto, la última se elimina del conjunto

Cuando el tamaño del conjunto *EP* ***sea mayor*** al especificado

si $soluciones \geq tamaño\ del\ archivo$ ***entonces***

para cada fitness dentro del conjunto EP

//Hallar la distancia euclidiana entre las posiciones de los valores no-dominados y el nuevo valor generado

$$d_{ij} = \sqrt{\sum_{i=1}^{n} (X_i^1 - X_i^2)^2}$$

Finalizar para

//La solución del conjunto eficiente con la mayor distancia con respecto a la nueva, se elimina.

Sino continúe el bucle

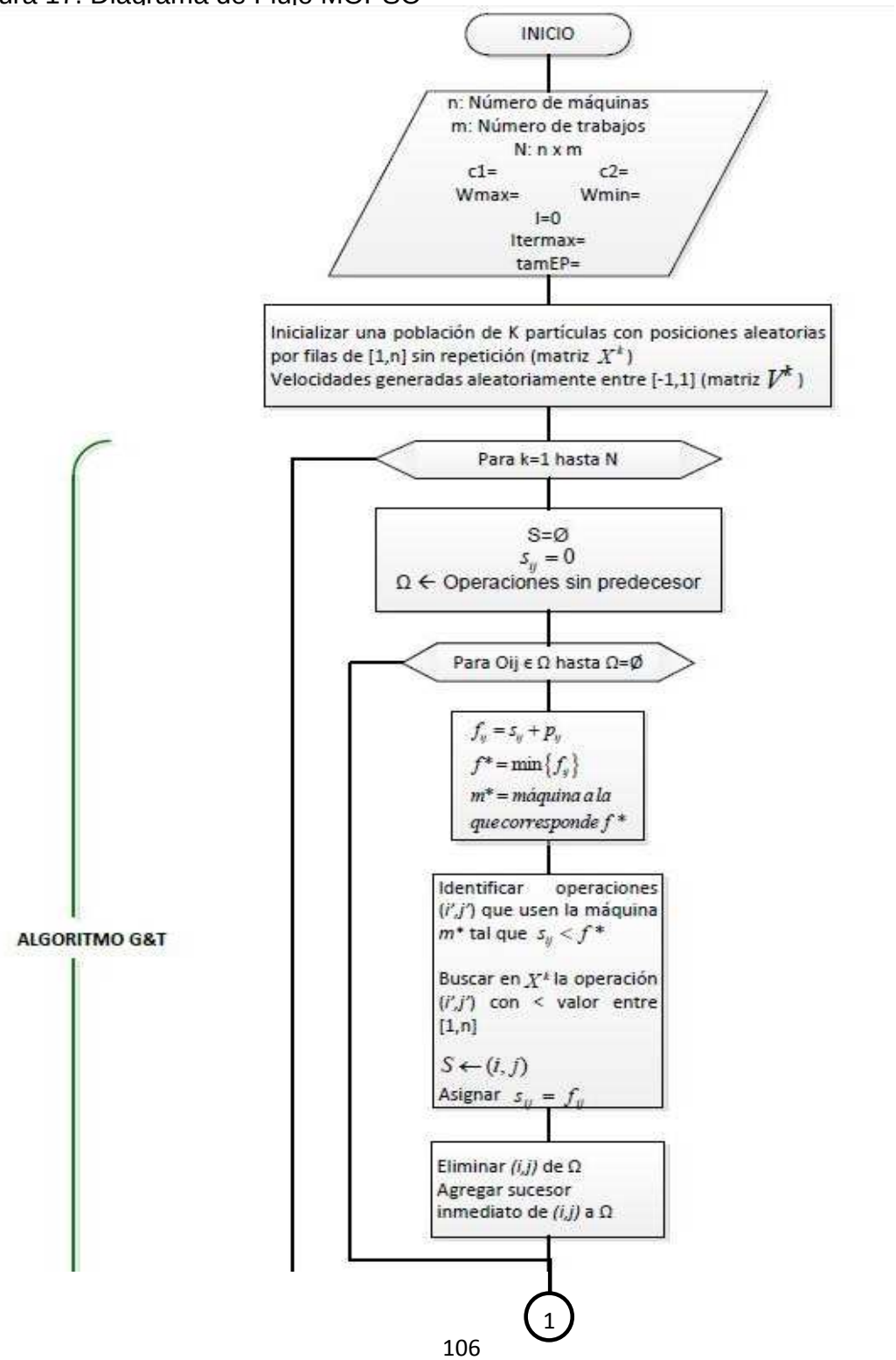
finalizar para

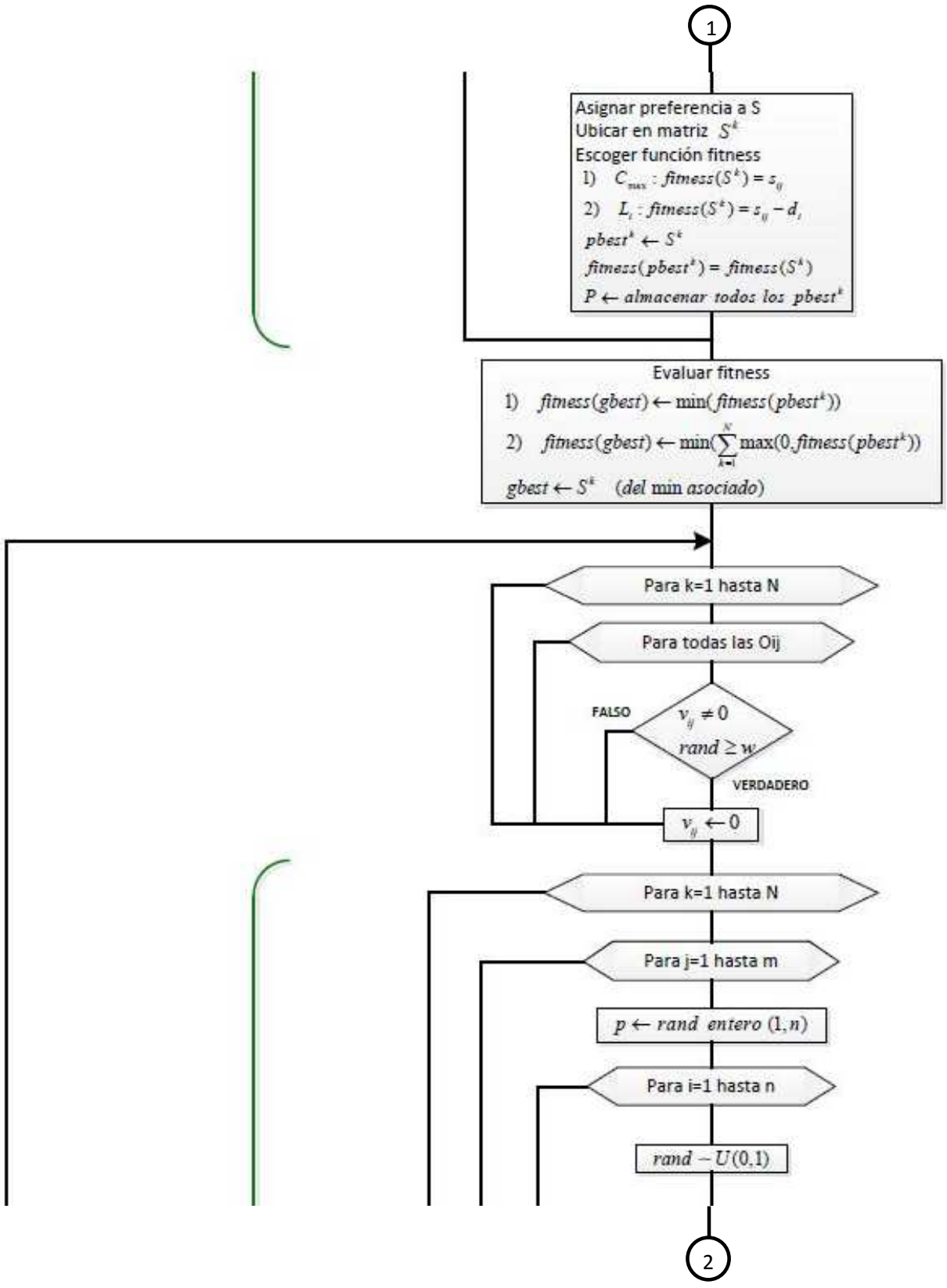
// El criterio de parada se establece hasta que se alcance el número máximo de iteraciones especificado por el usuario

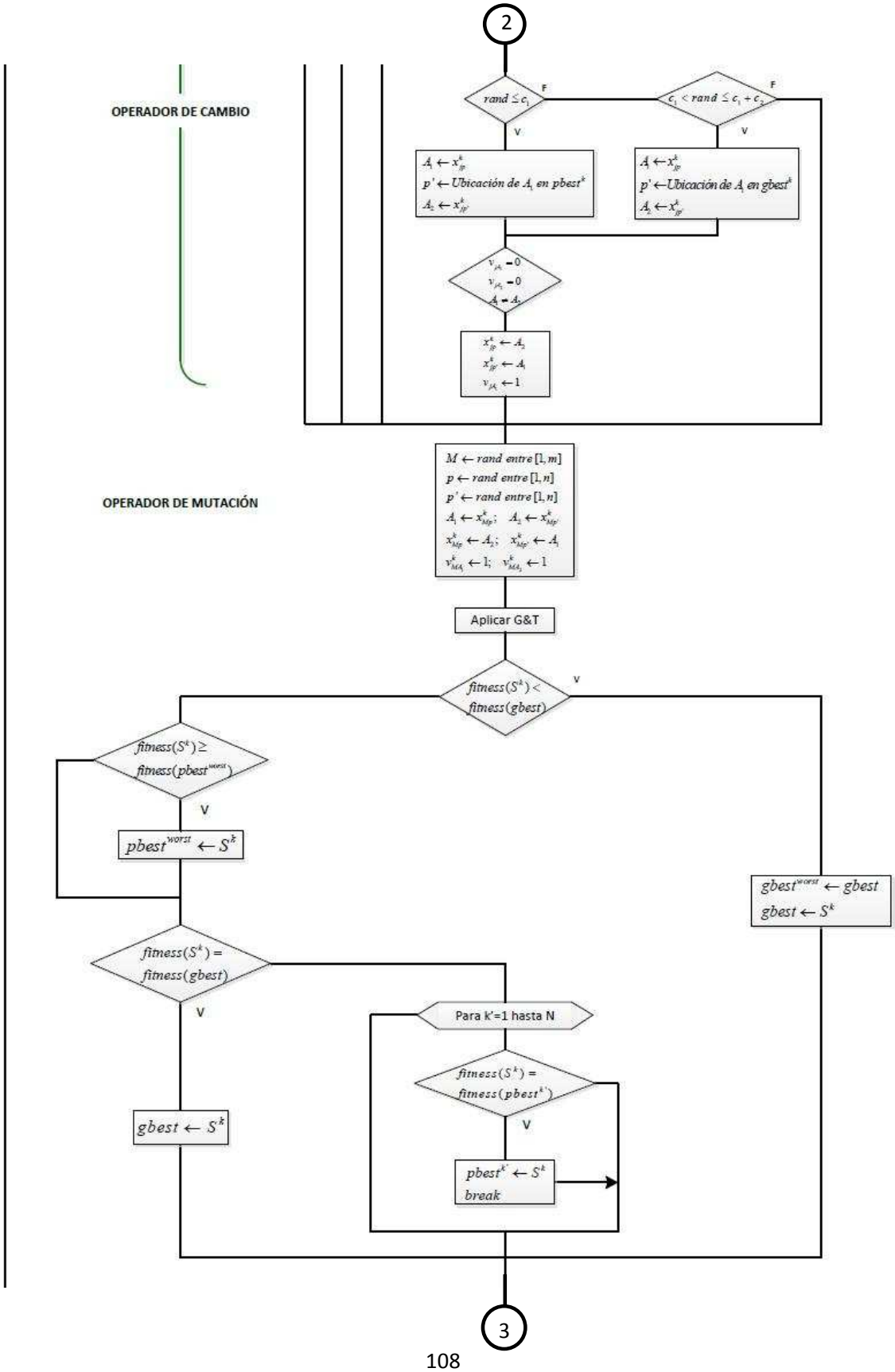
hasta que $iter = iter_{\max}$

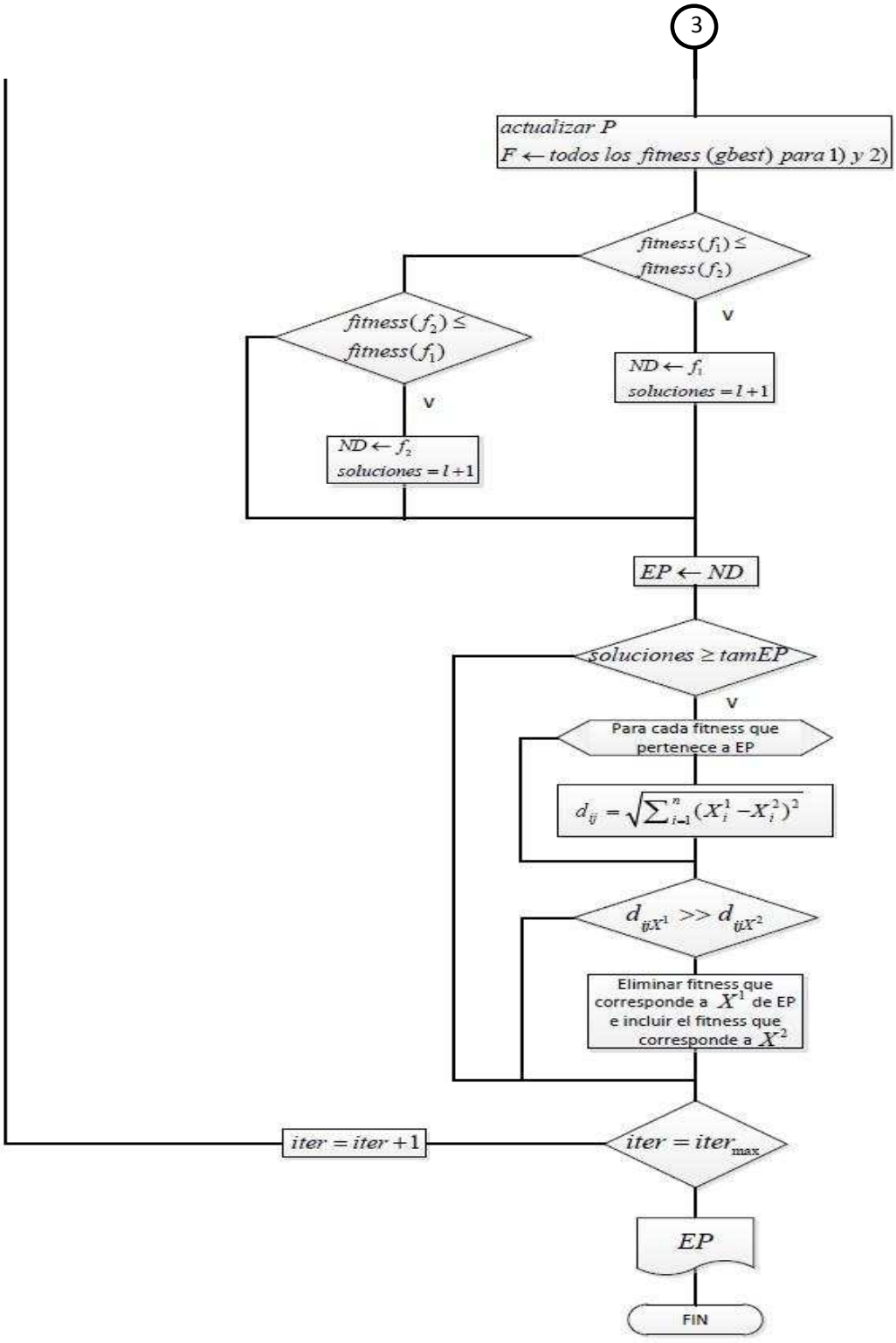
8. DIAGRAMA DE FLUJO

Figura 17. Diagrama de Flujo MOPSO




1
Asignar preferencia a S
Ubicar en matriz S^k
Escoger función fitness
1) C_{max} : $fitness(S^k) = s_{ij}$
2) L_i : $fitness(S^k) = s_{ij} - d_i$
$pbest^k \leftarrow S^k$
$fitness(pbest^k) = fitness(S^k)$
$P \leftarrow almacenar\ todos\ los\ pbest^k$
Evaluar fitness
1) $fitness(gbest) \leftarrow \min(fitness(pbest^k))$
2) $fitness(gbest) \leftarrow \min(\sum_{k=1}^{N} \max(0, fitness(pbest^k))$
$gbest \leftarrow S^k$ (del min asociado)
Para k=1 hasta N
Para todas las Oij
FALSO
$v_{ij} \neq 0$
$rand \geq w$
VERDADERO
$v_{ij} \leftarrow 0$
Para k=1 hasta N
Para j=1 hasta m
$p \leftarrow rand\ entero\ (1,n)$
Para i=1 hasta n
$rand - U(0,1)$
2


2
OPERADOR DE CAMBIO
rand ≤ c1
F
V
c1 < rand ≤ c1 + c2
F
V
A1 ← x^k_jp
p' ← Ubicación de A1 en pbest^k
A2 ← x^k_jp'
A1 ← x^k_jp
p' ← Ubicación de A1 en gbest^k
A2 ← x^k_jp'
v_jA1 = 0
v_jA2 = 0
A1 = A2
x^k_jp ← A2
x^k_jp' ← A1
v_jA1 ← 1
OPERADOR DE MUTACIÓN
M ← rand entre [1, m]
p ← rand entre [1, n]
p' ← rand entre [1, n]
A1 ← x^k_Mp; A2 ← x^k_Mp'
x^k_Mp ← A2; x^k_Mp' ← A1
v^k_MA1 ← 1; v^k_MA2 ← 1
Aplicar G&T
fitness(S^k) < fitness(gbest)
V
fitness(S^k) ≥ fitness(pbest^worst)
V
pbest^worst ← S^k
gbest^worst ← gbest
gbest ← S^k
fitness(S^k) = fitness(gbest)
V
Para k'=1 hasta N
fitness(S^k) = fitness(pbest^k')
V
gbest ← S^k
pbest^k' ← S^k
break
3



Fuente: Autor

9. BIBLIOTECA DE "BENCHMARK PROBLEMS" PARA EL JSP

En aras de realizar comparaciones entre los diferentes métodos y técnicas en el JSP éstos deben ser probados en la misma clase de instancias. Los *benchmark problems* actúan como una plataforma estándar para que los algoritmos puedan probarse y medirse.

Los problemas de comparación (*benchmark problems*) se usan para encontrar las ventajas comparativas de los diferentes métodos. Los problemas de comparación se prueban en los mismos tipos de problemas y se trata de establecer un estándar común para que los algoritmos sobre el JSP se puedan comparar entre sí. Los *benchmark problems* son usados comúnmente para medir la efectividad de los diferentes métodos. Como los problemas de *benchmark* son de diferentes dimensiones y grados de dificultad pueden realizar sugerencias sobre las mejoras que requieren los métodos.

Estos problemas han sido propuestos por varios autores, entre los que se destacan, Fisher y Thompson (1963), Lawrence (1984), Adams *et al.* (1988), Applegate y Cook (1991), Storer *et al.* (1992) y Yamada y Nakano (1992), entre otros. Los dos problemas de *benchmark* más conocidos son los formulados por Muth y Thompson[46] de tamaño 10x10 y 20x5 (mt10 y mt20 respectivamente) que son usados como "test beds[47]" para medir la efectividad de cierto método. El problema mt10 permaneció sin resolver por más de 20 años aunque ya no es un reto computacional. Applegate y Cook proponen un conjunto de problemas de benchmark llamado los "10 problemas más difíciles" ("ten tough problems") como

[46] MUTH, J.F., y THOMPSON, G.L. Industrial Scheduling. Prentice-Hall, Englewood Cliffs, N.J, 1963.
[47] Banco de pruebas.

un reto computacional mayor que el mt10 de los cuales algunos no han sido resueltos[48].

A continuación se nombrarán algunos autores junto con sus *benchmark problems* y un link de internet donde se pueden conseguir gratuitamente.

1) La OR-Library tiene una colección de datos para una variedad de problemas de IO, entre ellos el JSP. La siguiente página contiene los siguientes benchmark problems.

Link: 1. http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt
2. http://web.cecs.pdx.edu/~bart/cs510ss/project/jobshop/jobshop/

- **ABZ:** abz5-abz9 propuestos por J. Adams, E. Balas y D. Zawack (1988). Donde, abz5-abz6 son instancias 10x10 (10 trabajos, 10 máquinas) y abz7-abz9 son instancias 20x15.
- **FT:** ft06, ft10, y ft20 propuestos por H. Fisher y G.L. Thompson (1963). FT06 (o mt06) es una instancia 6x6, ft10 (mt10) es 10x10 y ft20 (mt20) es 20x5.
- **LA:** la01-la40 propuestos por S. Lawrence (1984). La01(setf1/F1)-la05(setf5/F5) son instancias 10x5, la06(setg1/G1)-la10(setg5/G5) son 15x5, la11(seth1/H1)-la15(seth5/H5) son 20x5, la16(seta1/A1)-la20(seta5/A5) son 10x10, la21(setb1/B1)-la25(setb5/B5) son 15x10, la26(setc1/C1)-la30(setc5/C5) son 20x10, la31(setd1/D1)-la35(setd5/D5) son 30x10, la36(seti1/I1)-la40(seti5/I5) son 15x15.
- **ORB:** orb01-orb10 propuestos por D. Applegate y W. Cook (1991). Todos son instancias 10x10 propuestos por diferentes personas, entre ellas, Bill Cook (BIC1), Monika (MON1), Bruce Gamble (BRG2), Bruce Sheperd (BRS2).
- **SWV:** swv01-swv20 propuestos por R.H. Storer, S.D. Wu y R. Vaccari (1992). swv01-swv05 son 20x10, swv06-swv10 son 20x15, swv11-swv20 son 50x10.

[48] YAMADA, T., NAKANO, R. Chapter 7: Job-Shop Scheduling. Genetic Algorithms in Engineering Systems, 1997, pp. 134–160. The Institution of Electrical Engineers, London, UK.

- **YN:** yn1-yn4 instancias 20x20 propuestos por T. Yamada y R. Nakano (1992).

2) En esta página se pueden descargar los datos propuestos por Taillard, Singer y Morton. Además, proponen unas posibles soluciones.
Link: http://www.emn.fr/z-auto/clahlou/mdl/Benchmarks.html#widget1

- **TD:** Problemas propuestos por E. Taillard (1993) para el criterio del makespan. Contiene 260 datos generados aleatoriamente, con instancias de tamaño real (tipo industrial) desde 20x5 hasta 500x20. Link: http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html.
- Singer y Pinedo (1998), fue propuesto para la suma total de la tardanza ponderada. Se basa en una selección de instancias clásicas para el makespan donde se agregan fechas de vencimiento y pesos.
- Morton y Pentico (1993) puede ser usado para cualquier criterio.
- Extensión de las instancias de Taillard. Los autores de la página han aplicado a las intancias de Taillard, las instancias de Singer y Pinedo para generar fechas de vencimiento y pesos.

3) JOBSHOP es un conjunto de programas en C para el JSP donde los códigos están basados en la investigación de Applegate (1991).
Link: http://www2.isye.gatech.edu/~wcook/jobshop/

4) En esta página propuesta por Oleg V. Shylo se pueden encontrar unos buenos niveles bajos y altos de los problemas benchmark de Taillard y Demirkol et al. Además, de los datos generados por Taillard y Demirkol et al. (1996) el cual contiene casi de 10.000 conjuntos de datos.
Link: http://plaza.ufl.edu/shylo/jobshopinfo.html

5) En esta página se pueden encontrar benchmarks para el JSP con tiempos de preparación y para el criterio de demora máxima $(L_{\max})$. Cada benchmark contiene 160 archivos, uno por cada instancia.
Link: http://pst.istc.cnr.it/~angelo/OUdata/

6) Los 10 problemas más difíciles del JSP de resolver según Applegate y Cook (1991):

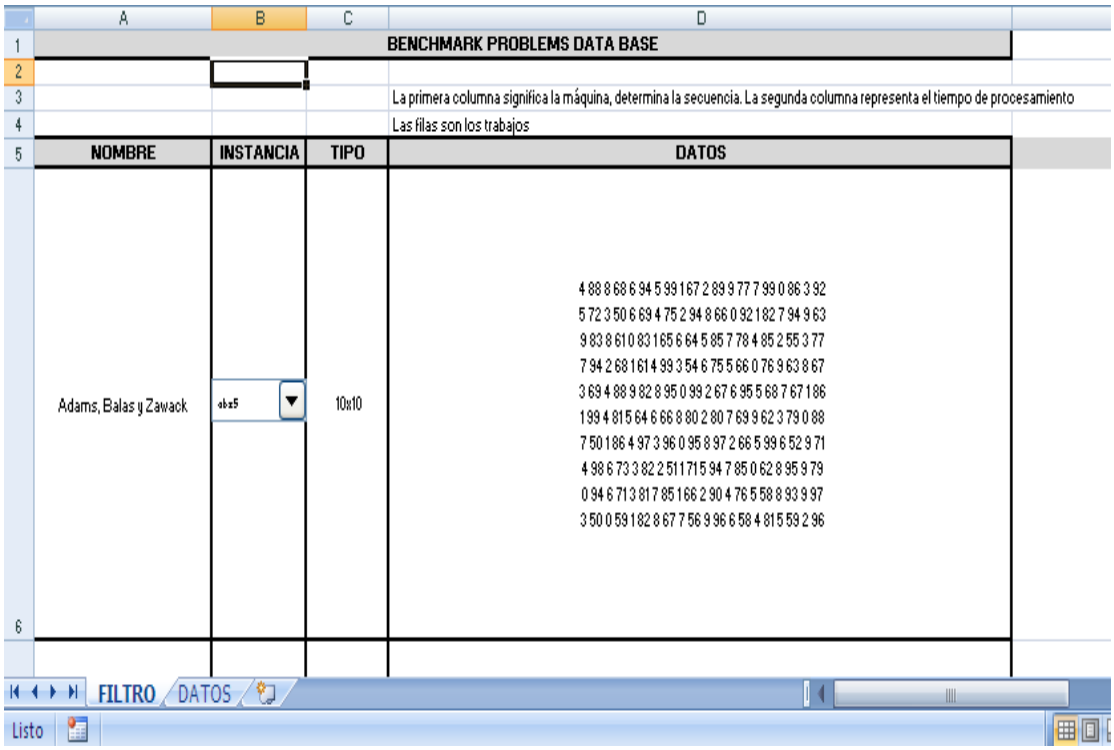
Tabla 6. 10 Problemas más difíciles del JSP

Problema	Tamaño	Mejor Solución	Mejor LB	Estado
ABZ7	15X20	668	654	ABIERTO
ABZ8	15X20	667	635	ABIERTO
ABZ9	15X20	707	656	ABIERTO
LA21	10X15	1053	1040	ABIERTO
LA24	10X15	935	935	RESUELTO
LA25	10X15	977	977	RESUELTO
LA27	10X20	1269	1235	ABIERTO
LA29	10X20	1195	1120	ABIERTO
LA38	15X15	1209	1184	ABIERTO
LA40	15X15	1222	1222	RESUELTO

Fuente: Applegate, David y Cook, William.

Además, se adjunta en el Anexo B un archivo de Excel que contiene los datos de los *benchmark problems* más usados, donde se especifica el autor, la instancia y el tipo de problema, como se observa en la Figura 18.

Figura 18. Benchmark problems data base.



BENCHMARK PROBLEMS DATA BASE

La primera columna significa la máquina, determina la secuencia. La segunda columna representa el tiempo de procesamiento

Las filas son los trabajos

NOMBRE	INSTANCIA	TIPO	DATOS
Adams, Balas y Zawack	abz5	10x10	4 88 8 68 6 94 5 99 1 67 2 89 9 77 7 99 0 86 3 92 5 72 3 50 6 69 4 75 2 94 8 66 0 92 1 82 7 94 9 63 9 83 8 61 0 83 1 65 6 64 5 85 7 78 4 85 2 55 3 77 7 94 2 68 1 61 4 99 3 54 6 75 5 66 0 76 9 63 8 67 3 69 4 88 9 82 8 95 0 99 2 67 6 95 5 68 7 67 1 86 1 99 4 81 5 64 6 66 8 80 2 80 7 69 9 62 3 79 0 88 7 50 1 86 4 97 3 96 0 95 8 97 2 66 5 99 6 52 9 71 4 98 6 73 3 82 2 51 1 71 5 94 7 85 0 62 8 95 9 79 0 94 6 71 3 81 7 85 1 66 2 90 4 76 5 58 8 93 9 97 3 50 0 59 1 82 8 67 7 56 9 96 6 58 4 81 5 59 2 96

FILTRO DATOS

Listo

Fuente: Autor

10. CONCLUSIONES

- En el presente proyecto se presenta de manera clara y rigurosa el algoritmo MOPSO. El algoritmo funciona tanto para el caso de un solo objetivo como el de múltiples objetivos. Se realiza la codificación de las partículas en el PSO para la representación de las soluciones del JSP y luego, la decodificación de la partícula del espacio continuo del PSO a una solución en el espacio discreto del JSP.

- Las modificaciones del PSO consisten en el uso del algoritmo de Giffler & Thompson para la decodificación de la posición de la partícula en un Schedule activo óptimo, el uso de un operador de intercambio para modificar y mover la partícula en el espacio, el uso de un operador de mutación para diversificar las posiciones de las partículas, y una estrategia de diversificación para darle mayor diferenciación a los valores generados y evitar que se quede atrapado en un óptimo local.

- El algoritmo que se estudió permite por medio de la eficiencia de Pareto determinar soluciones no dominadas. Se presenta de manera clara la estrategia y el correspondiente pseudocódigo que se puede implementar en algún lenguaje de programación.

- En el proyecto se presenta el pseudocódigo para la solución del JSP por medio del PSO de manera amplia en su documentación desarrollando un problema de escritorio de pequeña instancia que permite hacer un seguimiento de los pasos del mismo.

- La biblioteca de *benchmark problems* servirá para futuras investigaciones de la Escuela para la comparación de algoritmos que puedan desarrollarse en esta área. Se ofrecen links para la descarga de problemas de comparación, además, de resaltar se enuncian problemas que todavía permanecen abiertos de gran interés para la comunidad científica.

- El presente proyecto continúa con una de las líneas de investigación que se desarrolla en el área de programación de trabajos por parte de la Escuela de Ingeniería Industrial. Aunque se han desarrollado proyectos de grado usando métodos heurísticos y metaheurísticos que ofrecen buenas soluciones (casi óptimas), no se ha llevado a cabo ninguno usando el algoritmo PSO que en su filosofía se creó para la solución de problemas continuos.

11. RECOMENDACIONES

- Realizar un proyecto de grado junto a estudiante de Ingeniería de Sistemas con el objetivo de crear un software académico para la solución del JSP por medio del algoritmo MOPSO con el seudocódigo investigado, y poder comparar la eficiencia del método con los otros algoritmos desarrollados en la Escuela para la solución del JSP en los problemas *benchmark*.

- Dado que este tipo de temas es poco tratado por los estudiantes de Ingeniería Industrial UIS sería útil hacer proyectos interdisciplinarios de la mano de los estudiantes de Ingeniería de Sistemas UIS y así, complementar los conceptos del problema con la programación del método que lo resuelve.

- Se propone investigar las diferentes estrategias de representación de la partícula para adecuarla al espacio continuo del PSO y comparar el impacto de la codificación en la solución del JSP. Además, considerar otro tipo de restricciones como la recirculación, los tiempos de alistamiento, entre otros.

- Debido al carácter combinatorio del JSP es necesario usar algoritmos que usen la programación en paralelo, por consiguiente se propone crear un vínculo con el Parque Tecnológico de Guatiguará para usar la nueva Unidad de Supercomputación y Cálculo Científico, la cual permite realizar grandes cálculos para disminuir los tiempos computacionales requeridos y aumentar la eficiencia del método a utilizar. Además para fortalecer las relaciones en el ámbito investigativo entre el Parque y los grupos de investigación de estudiantes UIS.

- Como trabajo futuro se puede considerar la idea de investigar sobre un híbrido entre el PSO y entre un método de búsqueda local, como la Búsqueda Tabú o el Recocido Simulado para observar las principales diferencias entre el

híbrido y el PSO modificado. Además, se propone aplicar el método PSO a otros tipos de problemas de *Scheduling* tales como, el *Flow Shop, el Open Shop y el Flexible Open/Flow Shop*, entre otros. Aplicado, tanto en ámbitos de múltiples objetivos como de un solo objetivo.

BIBLIOGRAFÍA

AARTS, E. H. L., *et al*. A Computational Study of Local Search Algorithms for Job-Shop Scheduling. En: ORSA Journal on Computing, 1994, Vol. 6 (2), pp. 118-125.

ADAMS, J., BALAS, E., y ZAWACK, D. The Shifting Bottleneck Procedure for Job Shop Scheduling. En: Management Science, 1988, Vol. 34, pp. 391-401.

APPLEGATE, D., y COOK, W. A Computational Study of the Job-Shop Scheduling Instance. En: ORSA Journal on Computing, Vol. 3, 1991, pp. 149-156.

ASADZADEH, Leila., y ZAMANIFAR, Kamran. An Agent-Based Parallel Approach for the Job Shop Scheduling Problem with Genetic Algorithms. En: Mathematical and Computer Modeling, 2010, Vol. 52, pp. 1957-1965.

BARCELÓ, J. Panorama Actual de los Métodos Enumerativos. Seminario de Programación Matemática PM' 77, Madrid, Parte I.

BLUM, Christian y ROLI, Andrea. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. En: ACM Computing Surveys, September, 2003, Vol. 35 No. 3, pp. 268–308.

BRUCKER, Peter. Scheduling Algorithms. Fifth Edition, Germany, Springer, 2007, p. 367.

BÜLBÜL, Kerem. A Hybrid Shifting Bottleneck-Tabu Search Heuristic for the Job Shop Total Weighted Tardiness Problem. En: Computers & Operational Research, 2011, Vol. 38, pp. 967-983.

BURKE E.K., LANDA SILVA J.D., y SOUBEIGA, E. Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling, En: Meta-heuristics: Progress as Real Problem Solvers, Selected Papers from the 5th Metaheuristics International Conference (MIC 2003), 2005, pp 129-158.

BURKE, Edmund, *et al.* Hyper-Heuristics: An Emerging Direction in Modern Search Technology. Handbook of Metaheuristics (F. Glover and G. Kochenberger, eds.), Kluwer, 2003, pp. 457-474.

DAVIS, L. Job Shop Scheduling with Genetic Algorithm. En: Proceedings of the 1st International Conference on Genetic Algorithms, Pittsburgh, 24-26 July 1985, pp.136-140.

EBERHART, R y KENNEDY, J. A new optimizer using particle swarm theory. En: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, 1995, pp. 39–43.

EBERHART, Russel., y SHI, Yuhuy. Particle Swarm Optimization: Developments, Applications and Resources. En: Proceedings of the 2001 Congress on Evolutionary Computation, 2001, pp. 81-86.

FISHER, H., y THOMPSON, G.L. Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. J.F. Muth, G.L. Thompson (eds.), Industrial Scheduling, Prentice Hall, Englewood Cliffs, New Jersey, 1963, pp. 225-251.

GAO, Liang., ZHANG, Guohui., ZHANG, Liping y LI, Xinyu. An Efficient Memetic Algorithm for Solving the Job Shop Scheduling Problem. En: Computers & Industrial Engineering, 2011, Vol. 60, pp. 699-705.

GARY, M. R. JOHNSON, D. S. y RAVI Sethi. The Complexity of Flowshop and Jobshop Scheduling. En: Mathematics of Operations Research, Vol. 1, No. 2 (May), 1976, pp. 117-129.

GLOVER, F. Tabu Search and Adaptive Memory Programming - Advances, Applications and Challenges. En: Interfaces in Computer Science and Operations Research, 1996.

GOWER, M. WILKERSON, R. R-by-C Crozzle: An NP-Hard Problem. Tesis Maestría, Department of Computer Science, University of Missouri – Rolla.

HEFETZ, N y ADIRI, I. An Efficient Optimal Algorithm for the Two-Machines Unit-Time Job-Shop Schedule-Length Problem. En: Mathematics of Operations Research 7, 1982, pp. 354-360.

HUANG, Kuo-Ling., y LIAO, Ching-Jong. Ant colony optimization combined with taboo search for the job shop scheduling problem. En: Computers & Operations Research, 2008, Vol. 35, pp. 1030-1046.

JACKSON, J. R. An extension of Johnson's result on job lot scheduling. En: Naval Research Logistics Quarterly 3 (3), 1956, pp. 201-203.

JAIN, A.S y MEERAN, S. Deterministic Job Shop Scheduling: Past, Present and Future. En: European Journal of Operational Research, 1999, Vol. 113, pp. 390-434.

JOHNSON, S. M. Optimal two- and three-stage production schedules with set-up times included. En: Naval Research Logistics Quarterly 1, 1954, pp. 61-68.

KENNEDY, J. EBERHART, R. Proceedings of IEEE International Conference on Neural Networks, Service Center, Piscataway, NJ, Vol. IV. pp. 1942–1948.

KOBAYASHI, S., ONO, I., y YAMAMURA, M. An Efficient Genetic Algorithm for Job Shop Scheduling Problems. En: Proc., of ICGA'95, 1995, pp.506-511.

LAWRENCE, S. Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement), 1984, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

LEI, Deming. A Pareto archive particle swarm optimization for multi-objective job shop scheduling. En: Computers & Industrial Engineering, 2008, Vol. 54, pp. 960-971.

LI, Ye., y CHEN, Yan. An Effective TPA-Based Algorithm for Job-Shop Scheduling Problem. En: Expert Systems with Applications, 2011, Vol. 38, pp. 2913-2918.

LIAN, Z y JIAO, B y GU, X. A Similar Particle Swarm Optimization Algorithm for Job-Shop Scheduling to Minimize Makespan. En: Applied Mathematics and Computation, 2006, Vol. 183, pp. 1008–1017.

LIN, Tsung-Lieh., *et al.* An Efficient Job-Shop Scheduling Algorithm Based on Particle Swarm Optimization. En: Expert Systems with Applications, 2010, Vol. 37, pp. 2629-2636.

LONDOÑO, Pablo. MORA, Sebastián y XIBILLÉ, Juan Esteban. Módulo Job-Shop Scheduling para el Proyecto Arquímedes mediante el Heurístico Shifting Bottleneck. (Proyecto de Grado) Medellín, Universidad EAFIT, Departamento de Ingeniería de Producción, 2004, p. 100.

MANNE, Alan S. On the Job Shop Scheduling Problem. En: Operations Research, Mar. - Abr., 1960, Vol. 8, No. 2, pp. 219-223.

MARTÍ, Rafael. Procedimientos Heurísticos en Optimización Combinatoria. (Tesis Pregrado) España, Universidad de Valencia, Departamento de Estadística e Investigación Operativa, p. 60.

MATI, Yazid., DAUZÈRE-PÉRÈS, Sthèphane., y LAHLOU, Chams. A General Approach for Optimizing Regular Criteria in the Job-Shop Scheduling Problem. En: European Journal of Operational Research, 2011, Vol. 212, pp. 33-42.

MATTFELD, Dirk C., y BIERWIRTH, Christian. An Efficient Genetic Algorithm for Job Shop Scheduling with Tardiness Objectives. En: European Journal of Operational Research, 2004, Vol. 155, pp. 616- 630.

MORTON, T. E. y PENTICO, D. W. Heuristic Scheduling Systems. Wiley Series in Engineering and Technology Management, 1993. Wiley, New York.

MUTH, J.F., y THOMPSON, G.L. Industrial Scheduling. Prentice-Hall, Englewood Cliffs, N.J, 1963.

NAWAZ, Kazi S. Hybrid Evolutionary Approach for Multi-Objective Job-Shop Scheduling Problem. En: Malaysian Journal of Computer Science, 2007, Vol. 20 (2), pp. 183-198.

NESMACHNOW, S. Algoritmos Genéticos Paralelos y su Aplicación al Diseño de Redes de Comunicaciones Confiables, Capítulo 2: Técnicas De Computación Evolutiva. (Tesis de Maestría en Informática), Montevideo, Uruguay, 2004.

NIU, Q., JIAO, B y GU, X. Particle Swarm Optimization Combined With Genetic Operators For Job Shop Scheduling Problem With Fuzzy Processing Time. En: Applied Mathematics and Computation, 2008, Vol. 205, pp. 148–158.

OSMAN, I.H. y KELLY, J.P. Meta-Heuristics: Theory and Applications. Boston, USA. Ed. Kluwer Academic, 1996.

PEÑA, V., y ZUMELZU, L. Estado del Arte del Job Shop Scheduling Problem. Departamento de Informática, Universidad Técnica Federico Santa María, Valparaíso, Chile, 2006.

PESCH, E., TETZLAFF, U. A. W. Constraint Propagation Based Scheduling of Job Shops. En: Journal on Computing, Spring Vol. 8(2), 1996, pp. 144-157

QING-DAO-ER-JI, Ren., WANG, Yuping. A New Hybrid Genetic Algorithm for Job Shop Sheduling Problem. En: Computers & Operations Research, 2012, Vol. 39, pp. 2291-2299.

SELS, Veronique., CRAEYMEERSCH, Kjeld., y VANHOUCKE, Mario. A Hybrid single and dual population search procedure for Job Shop Scheduling Problem. En: European Journal of Operational Research, 2011, Vol. 215, pp. 512-523.

SEO, M y VENTURA, J. A. Ant Colony Optimization for Job Shop Scheduling. Proceedings of the 2008 Industrial Engineering Research Conference, pp. 1433-1438.

SHA, D. Y., LIN, H. H. A Particle Swarm Optimization for Multiobjective Flow-Shop Scheduling. En: The International Journal of Advanced Manufacturing Technology, Vol. 45, No. 7-8, pp. 749-758.

SHA, D. Y., LIN, Hsing-Hung. A Multi-objective PSO for Job Shop-Scheduling Problems. En: Expert Systems with Applications, 2010, Vol. 37, pp. 1065-1070.

SIERRA, María Rita. Mejora de Algoritmos de Búsqueda Heurística Mediante Poda por Dominancia. Aplicación a Problemas de Scheduling. Tesis Doctoral, Universidad de Oviedo, Departamento de Informática, 2009.

SINGER, M., y PINEDO, M. Computational Study of Branch and Bound Techniques for Minimizing the Total Weighted Tardiness in Job Shops. En: IIE Transactions, 1998, Vol. 29, pp. 109-119.

STORER, R.H., WU, S.D., y VACCARI, R. New Search Spaces for Sequencing Instances with Application to Job Shop Scheduling. En: Management Science, 1992, Vol. 38, pp. 1495-1509.

SUREKHA, P., y SUMATHI, S. Solution to the Job Shop Scheduling Problem Using Hybrid Genetic Swarm Optimization Based on (λ, 1)-Interval Fuzzy Processing Time. En: European Journal of Scientific Research, 2011, Vol.64 No.2, pp.168-188.

SURESH, R.K., MOHANASUNDARAM, K.M. Pareto Archived Simulated Annealing for Job Shop Scheduling With Multiple Objectives. En: The International Journal of Advanced Manufacturing Technology, May 2006, Vol. 29 (1-2), pp. 184-196.

TAILLARD, E. Benchmarks for Basic Scheduling Problems. En: European Journal of Operational Research 1993; 64; 278-285)

TAVAKKOLI-MOGHADDAM, R., *et al.* A New Hybrid Multi-Objective Pareto Archive PSO Algorithm for a Bi-Objective Job Shop Scheduling Problem. En: Expert Systems with Applications, 2011, Vol. 38, pp. 10812-10821.

WERNER, Frank, y WINCKLER, Andreas. Insertion techniques for the heuristic solution of the job shop problem. En: Discrete Applied Mathematics, 1995, Vol. 58, Issue 2, pp. 191-211.

YAMADA, T., NAKANO, R. Chapter 7: Job-Shop Scheduling. Genetic Algorithms in Engineering Systems, 1997, pp. 134–160. The Institution of Electrical Engineers, London, UK.

YAMADA, T., y NAKANO, R. A Genetic Algorithm Applicable to Large-Scale Job-Shop Instances. R. Manner, B. Manderick (Eds.), Parallel Instance Solving From Nature 2, North-Holland, Amsterdam, 1992, pp. 281-290.

YEN, G. G y IVERS, B. Job Shop Scheduling Optimization through Multiple Independent Particle Swarms. En: International Journal of Intelligent Computing and Cybernetics, 2009, Vol. 2, pp. 5-33.

ZHANG, ChaoYong., LI, PeiGen., RAO, YunQing., GUAN, ZaiLin. A Very Fast TS/SA Algorithm for the Job Shop Scheduling Problem. En: Computers & Operations Research, 2008, Vol. 35, pp. 282-294.

ZHANG, Rui., y WU, Cheng. A Hybrid Inmune Simulated Annealing Algorithm for the Job Shop Scheduling Problem. En: Applied Soft Computing, 2010, Vol. 10, pp. 79-89.

ANEXOS

ANEXO A. PRUEBA DE ESCRITORIO

Una prueba de escritorio es una forma útil para entender el funcionamiento de un algoritmo sin la necesidad de correrlo. Básicamente, se trata de evaluar el algoritmo paso a paso para comprobar si está cumpliendo su objetivo.

Dada la complejidad del algoritmo MOPSO, en el presente proyecto se realiza la prueba de escritorio para una instancia 3x3 (3 trabajos y 3 máquinas) del JSP, se ejecuta para una iteración y se generan dos partículas. La prueba de escritorio se realizó con la ayuda de la herramienta Excel para facilitar los cálculos.

Los datos con los que se realizó la prueba de escritorio se encuentran en la Tabla 6. El proceso que se explica a continuación se realiza para cada una de las partículas y para cada objetivo a optimizar.

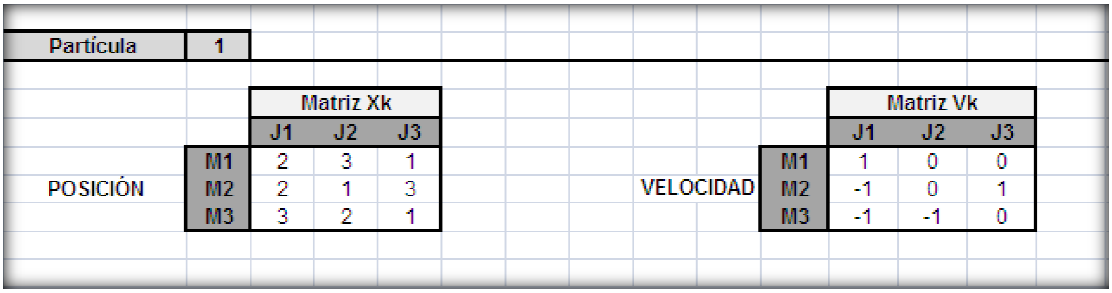
Inicialmente, se introducen los datos como se muestra en Figura 19. A continuación, se construye la matriz *operaciones*, la cual contiene la secuencia de las máquinas j que requieren los trabajos i, es decir, la operación (i, j). Esta matriz se usa en el Algoritmo de G&T dado que funciona tomando en cuenta las operaciones.

Se generó la matriz *posición* (Ver Figura 20) para cada partícula aleatoriamente por filas de 1 a *n* y sin repetición. En Excel se pueden usar las fórmulas de jerarquía y números aleatorios para generarla.

Figura 19. Datos iniciales para la instancia 3x3

	A	B	C	D	E	F	G	H	I	J	K
1	EJEMPLO PARA DOS PARTÍCULAS										
2	DATOS INICIALES		Jobs	Mq							
3			3	3							
4											
5				mq	tp	mq	tp	mq	tp		
6		J1	1	2	5	3	2	1	3		
7		J2	2	1	2	2	4	3	1		
8		J3	3	1	1	3	4	2	3		
9											
10			J1	1	2	3	1				
11	OPERACIONES		J2	2	1	2	3				
12			J3	3	1	3	2				
13											

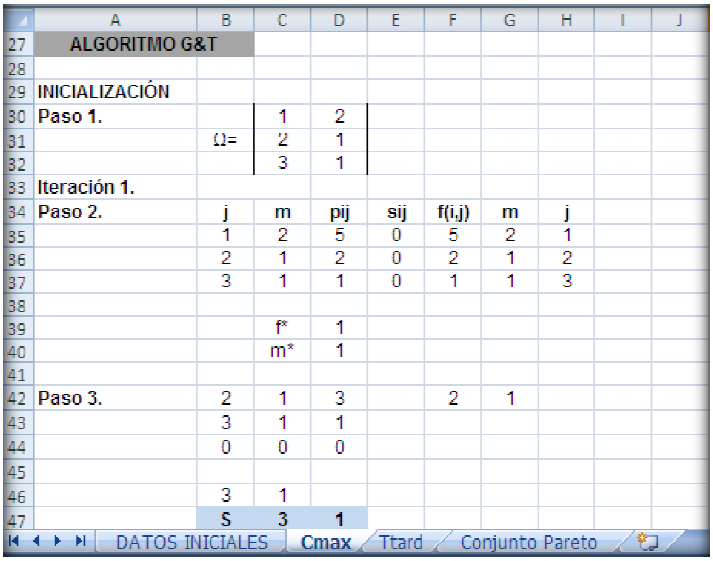
Figura 20. Matriz de posición y matriz de velocidades para la partícula 1.



Partícula	1									
		Matriz Xk						Matriz Vk		
		J1	J2	J3				J1	J2	J3
	M1	2	3	1			M1	1	0	0
POSICIÓN	M2	2	1	3		VELOCIDAD	M2	-1	0	1
	M3	3	2	1			M3	-1	-1	0

El algoritmo G&T inicia con el conjunto Ω, el cual contiene inicialmente las operaciones sin predecesor (primera columna matriz operaciones), a medida que itera elimina la operación programada e incluye su sucesor inmediato. En la figura 21 se muestran los primeros 3 pasos del algoritmo. El algoritmo escoge dependiendo de las prioridades de la matriz *posición.*

Figura 21. Inicialización Algoritmo G&T

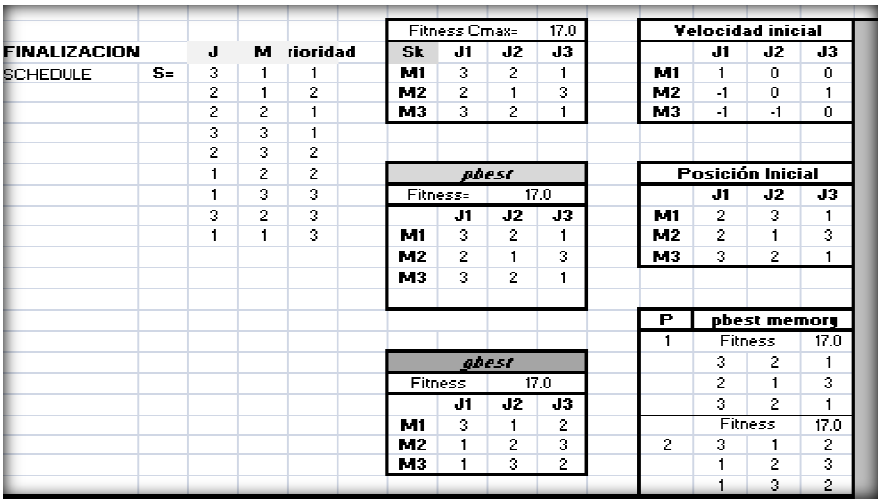


	A	B	C	D	E	F	G	H	I	J
27	ALGORITMO G&T									
28										
29	INICIALIZACIÓN									
30	Paso 1.		1	2						
31		Ω=	2	1						
32			3	1						
33	Iteración 1.									
34	Paso 2.	j	m	pij	sij	f(i,j)	m	j		
35		1	2	5	0	5	2	1		
36		2	1	2	0	2	1	2		
37		3	1	1	0	1	1	3		
38										
39			f*	1						
40			m*	1						
41										
42	Paso 3.	2	1	3		2	1			
43		3	1	1						
44		0	0	0						
45										
46		3	1							
47		S	3	1						

DATOS INICIALES | Cmax | Ttard | Conjunto Pareto

El algoritmo G&T finaliza cuando programa todas las operaciones del problema y las almacena en el conjunto S, la primera columna es el trabajo y la segunda la máquina. Además, se le asigna una prioridad que depende del orden en el que se programó en la máquina. En la Figura 22, se observan además los cuadros del *fitness* de la partícula, el *pbest* y el *gbest* de la partícula hasta el momento.

Figura 22. Schedule, matriz S^k, pbest y gbest para la partícula 1



FINALIZACION

SCHEDULE	S=	J	M	rioridad
		3	1	1
		2	1	2
		2	2	1
		3	3	1
		2	3	2
		1	2	2
		1	3	3
		3	2	3
		1	1	3

Fitness Cmax= 17.0

Sk	J1	J2	J3
M1	3	2	1
M2	2	1	3
M3	3	2	1

pbest

Fitness= 17.0

	J1	J2	J3
M1	3	2	1
M2	2	1	3
M3	3	2	1

gbest

Fitness 17.0

	J1	J2	J3
M1	3	1	2
M2	1	2	3
M3	1	3	2

Velocidad inicial

	J1	J2	J3
M1	1	0	0
M2	-1	0	1
M3	-1	-1	0

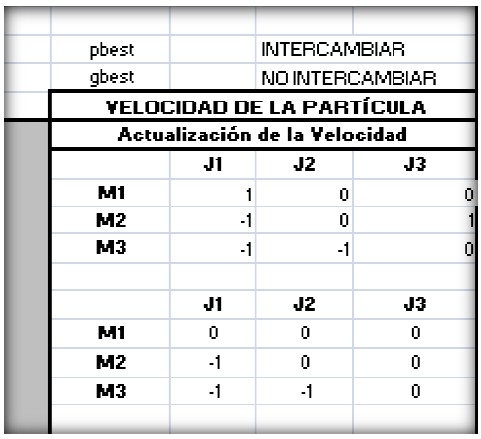
Posición Inicial

	J1	J2	J3
M1	2	3	1
M2	2	1	3
M3	3	2	1

P	pbest memory		
1	Fitness		17.0
	3	2	1
	2	1	3
	3	2	1
	Fitness		17.0
2	3	1	2
	1	2	3
	1	3	2

A continuación, se observa en la figura 23 la actualización de la velocidad, la cual depende de la generación de números aleatorios teniendo en cuenta el peso inercial w.

Figura 23. Actualización de la velocidad



pbest		INTERCAMBIAR
gbest		NO INTERCAMBIAR

VELOCIDAD DE LA PARTÍCULA

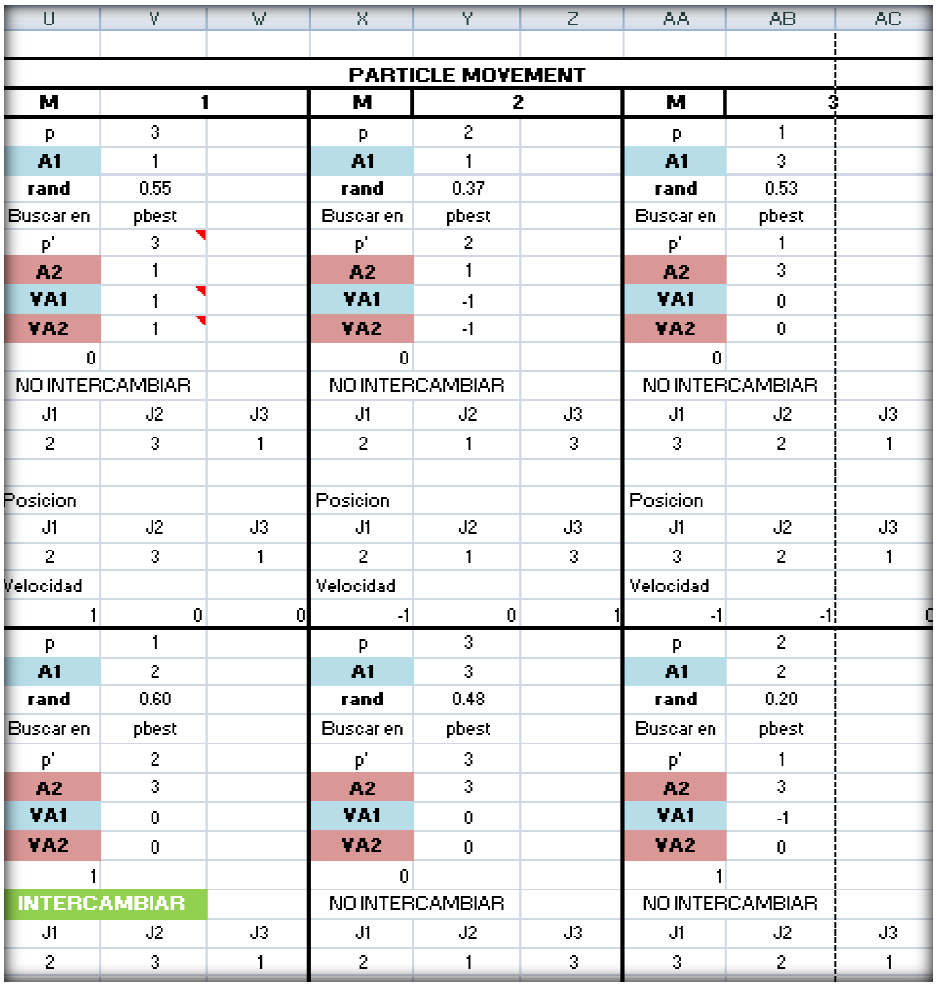
Actualización de la Velocidad

	J1	J2	J3
M1	1	0	0
M2	-1	0	1
M3	-1	-1	0

	J1	J2	J3
M1	0	0	0
M2	-1	0	0
M3	-1	-1	0

Después de actualizar la velocidad se utiliza el operador de intercambio, en el cual por cada máquina y por cada posición se realizan operaciones que determinan si se debe hacer un intercambio de posiciones de los trabajos en la respectiva máquina, esto se puede observar en la siguiente figura.

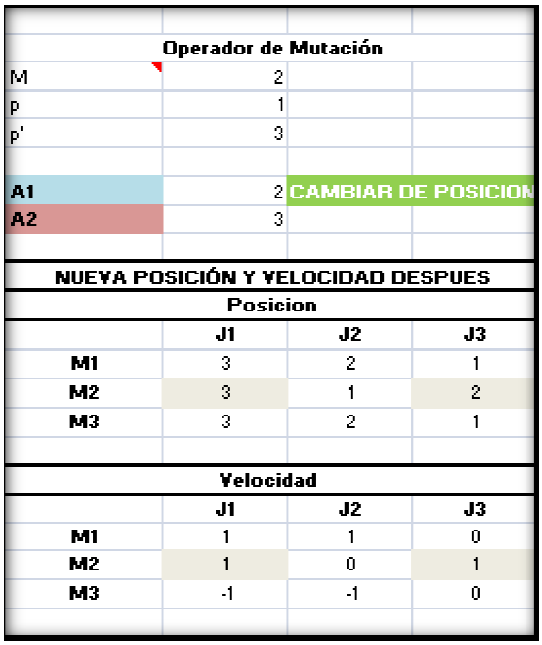
Figura 24. Movimiento de la partícula por medio del Operador de Intercambio



U	V	W	X	Y	Z	AA	AB	AC
PARTICLE MOVEMENT								
M	**1**		**M**	**2**		**M**	**3**	
p	3		p	2		p	1	
A1	1		**A1**	1		**A1**	3	
rand	0.55		**rand**	0.37		**rand**	0.53	
Buscar en	pbest		Buscar en	pbest		Buscar en	pbest	
p'	3		p'	2		p'	1	
A2	1		**A2**	1		**A2**	3	
VA1	1		**VA1**	-1		**VA1**	0	
VA2	1		**VA2**	-1		**VA2**	0	
0			0			0		
NO INTERCAMBIAR			NO INTERCAMBIAR			NO INTERCAMBIAR		
J1	J2	J3	J1	J2	J3	J1	J2	J3
2	3	1	2	1	3	3	2	1
Posicion			Posicion			Posicion		
J1	J2	J3	J1	J2	J3	J1	J2	J3
2	3	1	2	1	3	3	2	1
Velocidad			Velocidad			Velocidad		
1	0	0	-1	0	1	-1	-1	0
p	1		p	3		p	2	
A1	2		**A1**	3		**A1**	2	
rand	0.60		**rand**	0.48		**rand**	0.20	
Buscar en	pbest		Buscar en	pbest		Buscar en	pbest	
p'	2		p'	3		p'	1	
A2	3		**A2**	3		**A2**	3	
VA1	0		**VA1**	0		**VA1**	-1	
VA2	0		**VA2**	0		**VA2**	0	
1			0			1		
INTERCAMBIAR			NO INTERCAMBIAR			NO INTERCAMBIAR		
J1	J2	J3	J1	J2	J3	J1	J2	J3
2	3	1	2	1	3	3	2	1

Después de que se realiza el intercambio con todas las máquinas y posiciones, se usa el operador de mutación, el cual realiza un procedimiento parecido al anterior pero para una sóla máquina escogida aleatoriamente entre [1,n], esto para generar posiciones que sean diferentes.

Figura 25. Operador de mutación



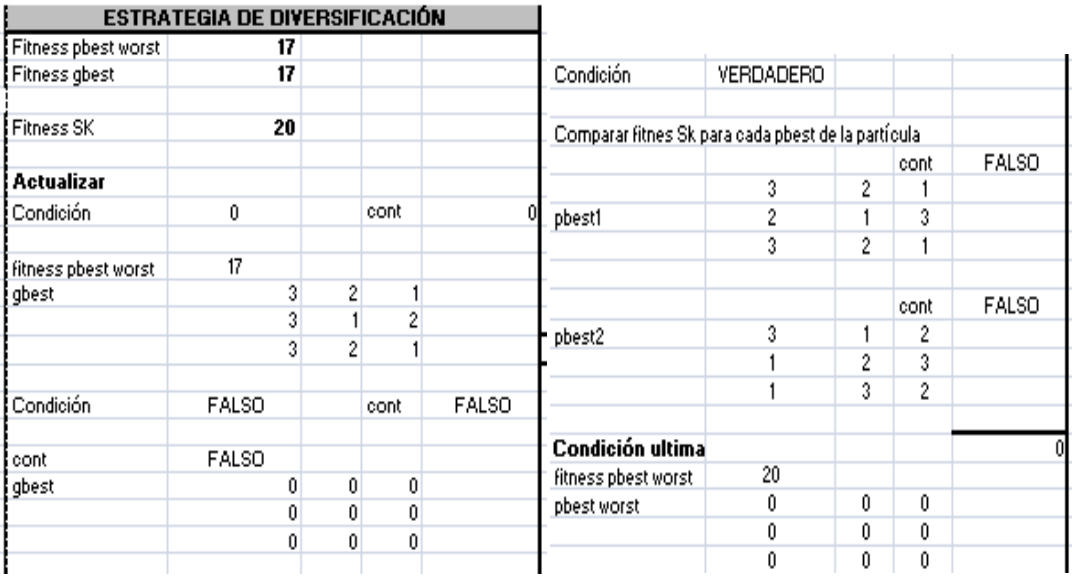
Operador de Mutación			
M	2		
p	1		
p'	3		
A1	2	CAMBIAR DE POSICION	
A2	3		

NUEVA POSICIÓN Y VELOCIDAD DESPUES			
Posicion			
	J1	J2	J3
M1	3	2	1
M2	3	1	2
M3	3	2	1
Velocidad			
	J1	J2	J3
M1	1	1	0
M2	1	0	1
M3	-1	-1	0

El siguiente paso, es volver a usar el Algoritmo de G&T para decodificar la nueva posición de la partícula en un nuevo shedule.

Después de generado todo lo anterior para cada partícula, se realiza la estrategia de diversificación, la cual a medida que una nueva solución es generada la compara para actualizar los valores *gbest, pbest* y $pbest^{worst}$.

Figura 26. Estrategia de diversificación para evitar caer en óptimos locales



ESTRATEGIA DE DIVERSIFICACIÓN					
Fitness pbest worst	**17**				
Fitness gbest	**17**				
Fitness SK	**20**				
Actualizar					
Condición	0		cont		0
fitness pbest worst	17				
gbest	3	2	1		
	3	1	2		
	3	2	1		
Condición	FALSO		cont	FALSO	
cont	FALSO				
gbest	0	0	0		
	0	0	0		
	0	0	0		

Condición	VERDADERO			
Comparar fitnes Sk para cada pbest de la partícula				
			cont	FALSO
	3	2	1	
pbest1	2	1	3	
	3	2	1	
			cont	FALSO
pbest2	3	1	2	
	1	2	3	
	1	3	2	
Condición ultima				0
fitness pbest worst	20			
pbest worst	0	0	0	
	0	0	0	
	0	0	0	

Finalmente, se obtienen los valores $fitness(gbest)$ de las partículas con los cuales se aplica el concepto de Eficiencia de Pareto. Como esto es una prueba de escritorio, esta parte del algoritmo simplemente se nombra debido a los pocos datos que se han obtenido de esta pequeña instancia. Sin embargo en el Anexo A de Excel se puede observar el funcionamiento.

DIAGRAMA DE FLUJO COMPLETO

INICIO

n: Número de máquinas
m: Número de trabajos
N: n x m
c1= c2=
Wmax= Wmin=
l=0
Itermax=
tamEP=

Inicializar una población de K partículas con posiciones aleatorias por filas de [1,n] sin repetición (matriz X^k)
Velocidades generadas aleatoriamente entre [-1,1] (matriz V^k)

ALGORITMO G&T

Para k=1 hasta N

S=Ø
$s_{ij} = 0$
Ω ← Operaciones sin predecesor

Para Oij ϵ Ω hasta Ω=Ø

$f_{ij} = s_{ij} + p_{ij}$
$f^* = \min\{f_{ij}\}$
$m^* = máquina\ a\ la\ que\ corresponde\ f^*$

Identificar operaciones (*i',j'*) que usen la máquina *m** tal que $s_{ij} < f^*$

Buscar en X^k la operación (*i',j'*) con < valor entre [1,n]

$S \leftarrow (i, j)$
Asignar $s_{ij} = f_{ij}$

Eliminar *(i,j)* de Ω
Agregar sucesor inmediato de *(i,j)* a Ω

Asignar preferencia a S
Ubicar en matriz S^k
Escoger función fitness
1) $C_{\max} : fitness(S^k) = s_{ij}$
2) $L_i : fitness(S^k) = s_{ij} - d_i$
$pbest^k \leftarrow S^k$
$fitness(pbest^k) = fitness(S^k)$
$P \leftarrow almacenar\ todos\ los\ pbest^k$

Evaluar fitness
1) $fitness(gbest) \leftarrow \min(fitness(pbest^k))$
2) $fitness(gbest) \leftarrow \min(\sum_{k=1}^{N} \max(0, fitness(pbest^k))$
$gbest \leftarrow S^k$ (*del* min *asociado*)

Para k=1 hasta N

Para todas las Oij

FALSO
$v_{ij} \neq 0$
$rand \geq w$
VERDADERO
$v_{ij} \leftarrow 0$

OPERADOR DE CAMBIO

Para k=1 hasta N

Para j=1 hasta m

$p \leftarrow rand\ entero\ (1, n)$

Para i=1 hasta n

$rand \sim U(0,1)$

$rand \leq c_1$ F
V
$A_1 \leftarrow x^k_{jp}$
$p' \leftarrow Ubicación\ de\ A_1\ en\ pbest^k$
$A_2 \leftarrow x^k_{jp'}$

$c_1 < rand \leq c_1 + c_2$ F
V
$A_1 \leftarrow x^k_{jp}$
$p' \leftarrow Ubicación\ de\ A_1\ en\ gbest^k$
$A_2 \leftarrow x^k_{jp'}$

$v_{jA_1} = 0$
$v_{jA_2} = 0$
$A_1 \neq A_2$

$x^k_{jp} \leftarrow A_2$
$x^k_{jp'} \leftarrow A_1$
$v_{jA_1} \leftarrow 1$

OPERADOR DE MUTACIÓN

$M \leftarrow rand\ entre\ [1, m]$
$p \leftarrow rand\ entre\ [1, n]$
$p' \leftarrow rand\ entre\ [1, n]$
$A_1 \leftarrow x^k_{Mp}$; $A_2 \leftarrow x^k_{Mp'}$
$x^k_{Mp} \leftarrow A_2$; $x^k_{Mp'} \leftarrow A_1$
$v^k_{MA_1} \leftarrow 1$; $v^k_{MA_2} \leftarrow 1$

Aplicar G&T

$fitness(S^k) < fitness(gbest)$ V

$fitness(S^k) \geq fitness(pbest^{worst})$
V
$pbest^{worst} \leftarrow S^k$

$gbest^{worst} \leftarrow gbest$
$gbest \leftarrow S^k$

$fitness(S^k) = fitness(gbest)$
V

Para k'=1 hasta N

$fitness(S^k) = fitness(pbest^{k'})$
V
$pbest^{k'} \leftarrow S^k$
break

$gbest \leftarrow S^k$

actualizar P
$F \leftarrow todos\ los\ fitness\ (gbest)\ para\ 1)\ y\ 2)$

$fitness(f_1) \leq fitness(f_2)$
V
$ND \leftarrow f_1$
$soluciones = l + 1$

$fitness(f_2) \leq fitness(f_1)$
V
$ND \leftarrow f_2$
$soluciones = l + 1$

$EP \leftarrow ND$

$soluciones \geq tamEP$
V
Para cada fitness que pertenece a EP

$d_{ij} = \sqrt{\sum_{i=1}^{n} (X_i^1 - X_i^2)^2}$

$d_{ijX^1} >> d_{ijX^2}$

Eliminar fitness que corresponde a X^1 de EP e incluir el fitness que corresponde a X^2

$iter = iter + 1$
$iter = iter_{\max}$

EP

FIN