DESDE 2013
https://repository.uaeh.edu.mx/revistas/index.php/icbi/issue/archive
Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI

ISSN: 2007-6363

Publicación Semestral Pädi Vol. 10 No. Especial 2 (2022) 56-64

Método híbrido para optimizar el Flexible Job Shop Scheduling Problem
Hybrid method to optimize the Flexible Job Shop Scheduling Problem

N. J. Escamilla-Serna[a,*], J. C. Seck-Tuoh-Mora[a], J. Medina-Marín[a], I. Barragán-Vite[a], J. R. Corona-Armenta[a]

[a] *Área Académica de Ingeniería y Arquitectura, Universidad Autónoma del Estado de Hidalgo, 42184, Pachuca, Hidalgo, México.*

Resumen

Este artículo aborda la programación de tareas en el Flexible Job Shop Scheduling Problem (FJSSP). En este sistema de manufactura es necesario incrementar el número de trabajos a procesar debido a las condiciones actuales del sector industrial en donde existe un aumento en la demanda de productos, lo que conlleva a incrementar la producción. Para encontrar una programación de tareas cercana al óptimo. Se propone un método de optimización híbrida utilizando una búsqueda global basada en algoritmos genéticos (AG) que tienen buena diversificación y para la búsqueda local se aplica una escalada de colinas simple con reinicio (ECR) para mejorar cada solución. La combinación de estas metaheurísticas obtiene el equilibrio necesario para encontrar la mejor programación de tareas con el fin de minimizar el makespan como función costo. Se implementó el algoritmo propuesto en Matlab, para comprobar su eficiencia se compararon los resultados con investigaciones recientemente publicadas.

Palabras Clave: Flexible Job Shop Scheduling Problem, algoritmos genéticos, escalada de colinas, optimización híbrida, makespan.

Abstract

This article addresses task scheduling in the Flexible Job Shop Scheduling Problem (FJSSP). In this manufacturing system, it is necessary to intensify the number of jobs to be processed due to the current conditions of the industrial sector where there is an increase in the demand for products, which leads to an increase in production. To find a task schedule close to the optimum. A hybrid optimization method is proposed using a global search based on genetic algorithms (GA) that have good diversification. A restart hill-climbing process (RHC) is used as a local search method in order to improve each solution. These metaheuristics yield the equilibrium necessary to find the best solution that minimizes the makespan as a cost function. The proposed algorithm was implemented in Matlab, and the results were compared with recently published research to review its efficiency.

Keywords: Flexible Job Shop Scheduling Problem, genetic algorithms, hill climbing, hybrid optimization, makespan.

1. Introducción

Debido al crecimiento de las industrias y la demanda de productos, las empresas se han visto en la necesidad de aumentar su producción donde a mayor demanda se necesita mayor competitividad. De acuerdo con Rodríguez, de Aguilar, y Hideaki (2018), a raíz del incremento de la demanda de productos en el mercado, las industrias están buscando diversas maneras de expandir sus ventajas competitivas, y una manera es mediante la optimización en su producción.

Uno de los principales propósitos que busca el sector manufacturero es la reducción de los tiempos de procesamiento para entregar la producción a tiempo, lo que requiere solucionar diversos problemas de programación. El termino programación se refiere a la asignación de recursos para realizar un conjunto de tareas en un periodo de tiempo deseado (Shen, Dauzère-Pérès, y Neufeld, 2018). Para solventar la manufactura de diferentes tipos de productos es necesario una flexibilidad de máquinas, es decir, diferentes máquinas son capaces de realizar tareas similares. Esto aumenta el número de opciones en la elección de máquinas y de trabajos a programar para conseguir un ahorro en los tiempos de procesamiento.

Para un sistema de producción industrial, desarrollar una programación eficiente de tareas y de asignación de recursos

*Autor para la correspondencia: es281355@uaeh.edu.mx
Correo electrónico: es281355@uaeh.edu.mx (Nayeli Jazmín Escamilla-Serna), jseck@uaeh.edu.mx (Juan Carlos Seck-Tuoh-Mora), jmedina@uaeh.edu.mx (Joselito Medina-Marín), rvingb@uaeh.edu.mx (Irvin Barragán-Vite), jrcorona@uaeh.edu.mx (José Ramón Corona-Armenta).

Fecha de recepción: 04/02/2022 Fecha de aceptación: 22/04/2022 Fecha de publicación: 24/06/2022
https://doi.org/ https://doi.org/10.29057/icbi.v10iEspecial2.8651

se convierte en un objetivo primordial en la planificación y gestión de los procesos de fabricación. Cada industria, de acuerdo con sus necesidades y posibilidades, debe encontrar la mejor programación de tareas posible para optimizar su funcionamiento. El inconveniente radica en que muchos de los problemas no pueden resolverse de manera óptima principalmente por el número de tareas a programar.

El problema de programación ha sido estudiado desde varias décadas atrás. Una característica en común que muestran algunos trabajos como Chen, Gen, y Tsujimura (1996), Xia y Wu (2005), Zhang, Shao, Li y Gao (2009) y recientemente Shen, Dauzère-Pérès, y Neufeld (2018), y Defershaÿ Rooyani (2020), es la necesidad de seguir proponiendo algoritmos que proporcione una solución adecuada, en el que requiera minimizar tiempos de producción. Lo anterior sigue siendo un problema abierto ya que cada vez se requiere producir más, lo que implica un mayor número de tareas que programar.

En este trabajo se aborda el Flexible Job Shop Scheduling Problem (FJSSP). En el FJSSP se desea encontrar la programación de tareas más adecuada en un ambiente flexible, en donde varias máquinas pueden realizar la misma tarea.

El FJSSP es un problema combinatorio en donde a mayor número de tareas existe un crecimiento exponencial de soluciones posibles, siendo un problema de tipo NP-Hard (Garey, Johnson y Sethi, 1976), (Adiri y otros, 1989). El crecimiento radica en que cuando hay *n* tareas, cada tarea tiene *o* operaciones y *m* máquinas pueden hacer cada tarea, el número posible de soluciones está dado primero por la secuenciación de tareas y después por la asignación de máquinas. Para la parte de la secuenciación de tareas, estas se pueden programar en *n!* formas diferentes para cada operación. Como cada tarea implica *o* operaciones, existen más de $z=(n!)^o$ posibles asignaciones de tareas. Para la asignación de máquinas, se tiene que cada tarea se puede asignar a *m* máquinas, dado un total mayor de $(m)^z$ soluciones. Esto demuestra que cuando *n, o* y *m* aumentan, el número de posibles combinaciones crece exponencialmente, haciendo imposible revisar todas las soluciones al utilizar métodos matemáticos precisos o exactos, como algoritmos de programación lineal o programación entera y mixta para encontrar la programación óptima.

Es por esto por lo que, a lo largo de los años, para resolver este tipo de problemas combinatorios se ha optado por utilizar alguna técnica con enfoque heurístico, donde se encuentran soluciones de alta calidad cercanas al valor óptimo en tiempos computacionales razonables y prácticos. Entre los algoritmos de aproximación más comunes se encuentran los algoritmos evolutivos (AE) como los algoritmos genéticos (AG) cuyos principios básicos fueron establecidos por Holland (1975), y descritos por Golberg y Holland (1989); la optimización de colonia de hormigas (ACO) introducida por Dorigo, M. (1992); la optimización de enjambre de partículas (PSO) desarrollado por Kennedy, J. y Eberhart, R. (1995); o la Búsqueda Tabú (TS) atribuida a Glover (1989), solo por mencionar algunos.

En este trabajo se propone una técnica metaheurística híbrida que minimice el tiempo de procesamiento de todas las tareas (o makespan) en problemas FJSSP con alta flexibilidad, es decir, donde un mayor número de máquinas puedan hacer la misma tarea. Esto se realiza aplicando una búsqueda global con algoritmos genéticos y después una búsqueda local usando escalada de colinas con reinicio múltiple. La primera metaheurística explora el conjunto de soluciones para resolver el subproblema de secuenciación de operaciones; la segunda metaheurística realiza la explotación de estas soluciones para encontrar la mejor programación de tareas en las máquinas disponibles.

Los resultados obtenidos con el método anterior se compararon con otros algoritmos clásicos y de reciente publicación, utilizando las instancias propuestas por Hurink, Jurisch, y Thole, (1994) conocidas como el banco de pruebas vdata. Estas instancias han sido ampliamente utilizadas para analizar la eficiencia y desempeño de nuevos algoritmos en la solución del FJSSP con alta flexibilidad.

2. Estado del arte

2.1. Flexible Job Shop Scheduling Problem (FJSSP)

El FJSSP es una generalización del Job Shop Scheduling Problem (JSSP) clásico, se puede resumir como un plan para organizar operaciones en un conjunto de máquinas, este conjunto consiste en máquinas independientes que trabajan en paralelo con características y capacidades operativas que pueden ser similares o diferentes (Brandimarte, 1993). La programación de máquinas paralelas significa que cada trabajo se puede procesar en cualquiera de las máquinas, y los tiempos de procesamiento son propios de cada máquina. Cada máquina puede procesar solo un trabajo a la vez, los trabajos se componen de varias operaciones que deben realizarse en un orden prestablecido. (Zobolas, Tarantilis, e Ioannou, 2008).

Para resolver el problema FJSSP, hay que considerar dos subproblemas, la secuenciación de operaciones (enrutamiento) y la asignación de máquinas (Rodríguez, de Aguilar, y Hideaki 2018). La primera vez que fue introducido el término FJSSP fue por Brucker y Schlie (1990); en su trabajo propusieron un algoritmo polinomial para dos trabajos. Posteriormente Brandimarte (1993) fue uno de los primeros en abordar el problema FJSSP con un enfoque heurístico, su método consistió en la solución independiente del subproblema de enrutamiento y del subproblema de asignación aplicando reglas de despacho y una búsqueda tabú, proponiendo 15 instancias de prueba.

A partir de entonces hasta nuestros días, el FJSSP ha sido estudiado por diversos investigadores que han propuesto nuevos métodos heurísticos y metaheurísticos para encontrar una mejor optimización. El criterio más utilizado sigue siendo el makespan como función objetivo. Gao y otros (2016), mencionan que el makespan es un objetivo importante para un FJSSP, en dicha investigación, toman el criterio relacionado con la fecha de vencimiento en las empresas de fabricación, destacando la importancia en los entornos de producción justo a tiempo (JIT).

Deng y otros (2017) utilizan un AG y una guía evolutiva de abejas denominado (EG-NSGA-II) para una función costo multiobjetivo. Yang y otros (2020) tratan la maximización de la carga de trabajo y la minimización del tiempo de proceso de cada operación, tomando en cuenta las averías de la máquina. Ding y Gu (2020) usan un algoritmo de optimización de aprendizaje humano (HLO) y el cúmulo de partículas (PSO) aplicando estrategias de programación basadas en reglas para promover la eficiencia y eficacia de la producción.

Dada la naturaleza combinatoria del FJSSP, una línea de investigación activa consiste en combinar métodos heurísticos para formar algoritmos más robustos. Esto conlleva a la generación de metaheurísticas, entendidas como procedimientos de alto nivel que coordinan reglas y heurísticas simples para calcular soluciones adecuadas en problemas de optimización complejos. Glover (1989) acuña el término de metaheurística al proponer la búsqueda tabú. A continuación, se mencionan los conceptos de las metaheurísticas utilizadas en este trabajo.

2.2. Algoritmos Genéticos (AG)

Los Algoritmos Genéticos (AG) forman una gran técnica de búsqueda y optimización con un comportamiento inspirado en el principio de la selección natural y reproducción genética de Charles Darwin de 1859. Este principio se tomó como base en el campo de la optimización para crear un proceso de búsqueda paralela.

Forgel, Owens y Walsh (1966) trabajaron con la programación evolutiva e hicieron una serie de experimentos, que muestran un algoritmo evolutivo capaz de reconocer números que son divisibles por dos o tres. En el trabajo de Rechenberg (1973) se presenta la primera idea de aplicar estrategias evolutivas. La primera aplicación de los AG fue en el trabajo de John Holland (1975), donde planteó la posibilidad de incorporar mecanismos naturales de selección y supervivencia a la resolución de problemas de inteligencia artificial. El trabajo de Koza (1992) se muestra un modelo de programación genética que proporciona soluciones a una serie de problemas en diferentes campos de aplicación.

Desde entonces se han trabajado con AG para solucionar distintos problemas de optimización combinatoria. Por ejemplo, los trabajos de Chen, Gen, y Tsujimura, (1996) que dan un tutorial utilizando AG para un problema de JSSP. Por su parte Salido, y otros (2016) resolvieron un problema para obtener la solución óptima en términos de tiempo de procesamiento, costos y calidad como sus objetivos de optimización. Sreekara y otros (2018) utilizan AG de ordenamiento no dominante con operadores que mejoraron la versión NSGA. Defersha y Rooyani (2020), proponen un AG de dos etapas, codificando la decisión de asignación y la otra la decisión de secuencia, mediante un proceso aleatorio.

En el trabajo de Amjad (2018) se da una revisión de la literatura muy completa de diferentes autores que aplican los AG al FJSSP.

2.3. Escalada de Colinas (EC)

La (EC) es una técnica de optimización enfocada para la optimización local. Este algoritmo es clasificado como algoritmo codicioso debido a que después de cada iteración del algoritmo, solo acepta una nueva solución si hay una mejora, teniendo más oportunidades de obtener buenos resultados.

La forma de trabajo de este algoritmo es, primero comenzar con una solución aleatoria (probablemente pobre), y genera iterativamente pequeños cambios a la solución con el objetivo de obtener un resultado de la función objetivo mejor. El algoritmo finaliza cuando alcanza los criterios de paro. Por supuesto, no se garantiza que se haya obtenido el mejor resultado (Rodríguez y otros, 2018).

La EC sigue una dirección de ascenso/descenso a partir de su posición inicial, una ventaja de este algoritmo es que requiere muy poco costo computacional. La definición clásica de la EC tiene el problema de quedar atrapada en máximos (o mínimos) locales, como se muestra en la Figura 1, donde la línea rosa indica el ascenso de la EC y su máximo local. Por tal motivo surge la necesidad de hibridar el algoritmo con otro algoritmo de optimización.

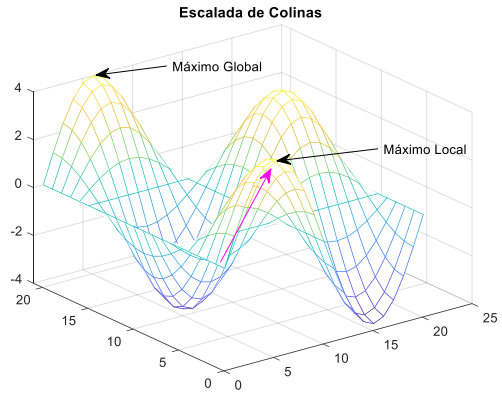

Figura 1. Ejemplo de la ruta de la Escalada de Colina

En el trabajo de Mailing (2003) se señalan algunos autores que han trabajado en encontrar el mejor método de búsqueda local. El trabajo más difundido fue el de Aarts y Korst (1989) donde utilizan el recocido simulado modelado a partir de las cadenas de Markov. Por su parte Kern (1989) estudio aspectos teóricos y probabilísticos del comportamiento de la búsqueda local para el problema del agente viajero.

Dados los problemas de los métodos de búsqueda local al quedar atrapados en óptimos locales, se han creado alternativas para mejorar su funcionamiento. En el trabajo de Dong y otros (2013) se estudia la problemática de la búsqueda local iterada, permitiendo extensiones para escapar de los óptimos locales.

Esta idea ha dado lugar a la aplicación de metaheurísticas de arranque múltiple, las de entorno variables y las búsquedas no monótonas. Algunos autores que trabajaron en el estudio de estas metaheurística son Dong, y otros (2013), donde aplicaron el reinicio múltiple para la minimización del flujo de tiempo para un problema de secuenciación del flow shop. Otro trabajo es el de Michallet, y otros (2014) quienes usan el reinicio múltiple para la obtención de una mejor ruta de vehículos.

El reinicio múltiple significa que una vez que hay un estancamiento en la solución, se reinicia en otra nueva solución

para que escapar del óptimo local. La nueva solución debe ser diferente a la original para no ser atrapada nuevamente, pero no demasiado distinta como para empezar nuevamente la búsqueda desde un lugar pobre.

Rodríguez, de Aguilar, y Hideaki (2018) abordan el FJSSP utilizando un enfoque híbrido de cúmulo de partículas (PSO) y el la Escala de Colinas con Reinicio (ECR), su objetivo es minimizar el makespan, la carga de trabajo y la carga total de las máquinas.

Alzaqebah (2020) trabaja el FJSSP presentando un método denominado lluvia de ideas (BSO). Este es un algoritmo de enjambre que simula el proceso de lluvia de ideas de los humanos y para superar la convergencia lenta del algoritmo, se aplica la EC de aceptación tardía con tres vecindarios distintos.

Los trabajos anteriores muestran que, al utilizar un método de búsqueda local, para evitar que la solución quede atrapada en un mínimo no óptimo es necesario implementar nuevas estrategias. En esta investigación se usa la ECR.

3. Fundamentos y formulación del Flexible Job Shop Scheduling Problem

La representación del FJSSP puede ser explicada como sigue (Gao, y otros, 2006).

Se tiene un conjunto de n trabajos $J= \{J_1, J_2, J_3, \ldots, J_n\}$ y un conjunto de m máquinas $M= \{M_1, M_2, M_3, \ldots, M_m\}$. Cada trabajo J_i consiste de una secuencia de O operaciones $J_1= \{O_{i,1}, O_{i,2}, O_{i,3}, \ldots, O_{i,ni}\}$ donde n_i es el número de operaciones de un trabajo i que comprende cada operación $O_{i,j}$. Cada operación O_{ij} (i=1, 2, …, n; j=1, 2, …, n_i) es procesada por una máquina de un conjunto de máquinas dadas $M_{ij} \subseteq M$.

En el FJSSP se necesita programar la secuencia de las operaciones y la asignación de las operaciones en las máquinas, tomando en consideración las siguientes condiciones:

- Todos los trabajos, las operaciones y las máquinas candidatas están disponibles en el tiempo t=0.
- Existe un orden previamente asignado para las operaciones de cada trabajo.
- Las diferentes operaciones de un trabajo no se pueden procesar simultáneamente.
- Los trabajos son independientes entre sí.
- Cada máquina puede ejecutar solo una operación a la vez en un momento dado.
- No se puede interrumpir la operación una vez iniciado el proceso en la maquina asignada.
- No se consideran averías, ni tiempos de preparación ni transportes.
- El tiempo de procesamiento de una operación O_{ij} en una máquina M_r es $P_{ijr}>0$.

El makespan es el criterio de optimización más usado en la revisión de la literatura (Li y Gao 2016), (Xie y Chen 2018), (Dai y otros 2019) por mencionar algunos.

El objetivo al solucionar el FJSSP será la minimización del makespan, entendiéndose como el tiempo máximo que tardan en completarse todas las operaciones. La función objetivo es minimizar el makespan C_{max} y es definido como:

$C_{max,} = max\ \{C_i\}$, donde C_i es el tiempo de finalización para todas las operaciones del trabajo J_i para $1 \leq i \leq n$.

4. Propuesta del algoritmo híbrida para el FJSSP

Este trabajo propone un algoritmo híbrido aplicando los AG y la ECR

4.1 Algoritmos Genéticos (AG)

El AG se basa en una población de soluciones que evoluciona utilizando los operadores genéticos de selección, cruce y mutación, como se aprecia en la Figura 2.

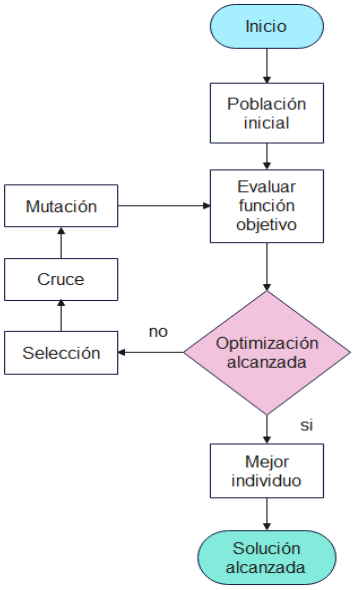

Figura 2. Diagrama de funcionamiento de los AG

El AG trabaja con una población inicial de cadenas para la representación del problema, las cadenas se evalúan en la función objetivo para realizar una transformación de posibles soluciones mediante la aplicación de operadores de selección, cruce y mutación, generando así nuevas soluciones en el espacio de búsqueda hasta alcanzar el mejor costo posible en la función objetivo y tener un individuo mejorado.

La selección implica escoger un subconjunto de soluciones, ya sea por elitismo o por torneo. El cruce combina soluciones padres para generar nuevas soluciones. En la mutación se seleccionan aleatoriamente algunas soluciones que sufren pequeñas modificaciones en su estructura. El ciclo de optimización continua mientras no se alcance el criterio de paro, regresando al final una solución mejorada (Figura 3).

Procedimiento del Algoritmo Genético ()
1. Genera la población inicial de P soluciones
2. Evaluar Aptitud(P)
3. **While** no se alcance el criterio de parada **do**
4. *Padres*←Selección(P)
5. *Hijos*←Recombinación de padres(*padres*)
6. *Hijos*←Aplicar Mutación (*Hijos*)
7. EvaluarAptitud(*hijos*)
8. P← Seleccionar Nueva Población (Hijos U Padres)
9. **End_while**
10. **Return** la mejor solución encontrada;

End

Figura 3. Plantilla de un algoritmo genético

4.2 Escalada de colinas (EC)

La EC es un método de mejora iterativa, empezando con una solución inicial y aplicando pequeñas modificaciones para mejorar su calidad. Un problema con la EC es que el mejoramiento iterativo se mantiene en una pequeña área del espacio de soluciones y pude llegar a estancarse sin lograr ningún avance. Cuando este es el caso, se debe de empezar en otro punto aleatorio, lo que se conoce como escalada de colinas con reinicio (ECR). Al aplicar este proceso se pueden obtener posibles mejoras seleccionando una solución que conduzca a un mejor valor o usando un límite de iteraciones para la escalada (Figura 4).

```
Procedimiento tradicional de la Escalada de colinas ()
1.  Población_Actual ← Estado Inicial
2.  Fin ← falso
3.  While fin sea = falso do
4.          Hijos ← generar_desendientes (Población_Actual)
5.          Mayor← selecciona_mayor(hijos)
6.          If valor(mayor)<Valor (Población_Actual)
7.                  Fin←true
8.          Else
9.                  Población_Actual←Mayor
10.         End if
11. End while
12. Return Poblacion_Actual
```

Figura 4. Plantilla de un algoritmo de escalada de colinas

4.3 Algoritmo Híbrido AG-ECR

El objetivo de este trabajo es encontrar el mínimo makespan o función costo para instancias del FJSSP con alta flexibilidad, usando un enfoque híbrido con dos técnicas aplicadas de forma jerárquica. Debido a que es difícil minimizar el makespan por la explosión combinatoria de soluciones, y no se tiene un método general que dé una solución adecuada para todas las instancias del FJSSP, surge la oportunidad de investigar un nuevo método que encuentre la mejor programación de tareas, resolviendo el subproblema de secuenciación de operaciones y de asignación de máquinas.

Se propone un algoritmo híbrido (AG-ECR) que permite combinar heurísticas para encontrar una buena solución. La búsqueda global se realiza con un AG para la exploración de soluciones, mientras que la ECR efectuará la búsqueda local para la explotación de las soluciones, de manera similar a otras propuestas como la de Li y Gao (2016).

Para resolver el problema del FJSSP al aplicar un enfoque jerárquico, en la etapa del subproblema de secuenciación se calcula la cadena OS con la programación del orden en que las operaciones serán procesadas. Para la otra etapa del subproblema de asignación, se encuentra la cadena para la selección de máquinas MS, es decir, la que asigna a cada operación una máquina de un conjunto factible de máquinas posibles. De esta manera, el método híbrido AG-ECR obtiene la mejor programacion de tareas posibles para instancias del FJSSP.

El diagrama de flujo del AG-ECR se observa en la Figura 5 y se describe de la siguiente manera:

- Paso 1: Se inicializa la población con soluciones generadas de forma aleatoria, se obtienen dos cadenas, una contiene el orden de operaciones y la otra la asignación de las máquinas.
- Paso 2. Se evalúa cada solución en la función costo para encontrar su makespan.
- Paso 3. Si ya se alcanzó el número de iteraciones deseado, entonces se presenta la mejor solución obteniendo el makespan mínimo y termina el proceso. En caso contrario ir al paso 4.
- Paso 4. Se selecciona la mejor población por medio de elitismo y torneo.
- Paso 5. Se generan nuevas soluciones mediante el cruce.
- Paso 6: Se hace una mutación de algunas soluciones para obtener un mejor descendiente.
- Paso 7: Para cada solución mejorada se hace una escalada de colina con reinicio. Regresar al paso 2.

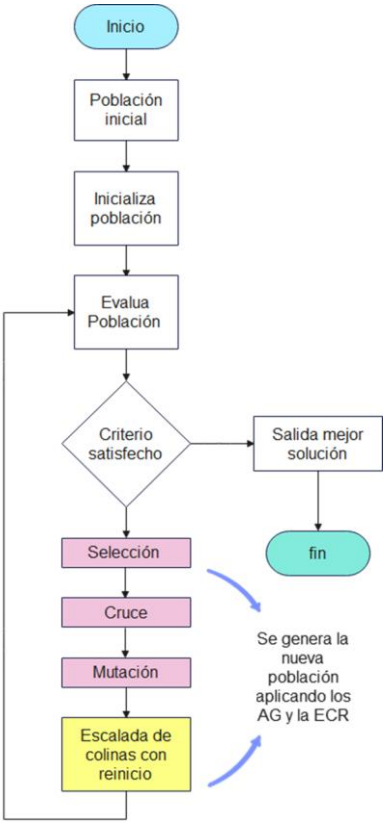

Figura 5. Diagrama de flujo propuesto

5. Desarrollo experimental

Los problemas de prueba se tomaron de las instancias vdata propuestos por Hurink, Jurisch, y Thole (1994), por tener una alta flexibilidad, en donde cada operación puede ser procesada por la mitad de las máquinas disponibles. Estas instancias son adaptaciones de problemas propuestos para el JSSP, en particular de las instancias mt06, mt10 y mt20 de Fisher and Thompson (1963), y las 40 instancias la01-la40 de Lawrence (1984).

Cada instancia contiene la información del número de trabajos, el número de máquinas, el número de operaciones por trabajo y el tiempo de procesamiento de cada operación en cada máquina factible.

Con esta información se genera una población inicial aleatoria donde cada individuo tiene dos cadenas, una con la secuencia de operaciones denominada como OS y la segunda con la selección de cada máquina para cada operación llamada MS (Paso 1).

La cadena OS es una permutación con repeticiones de *n* números de trabajos, cada trabajo contempla *ni* número de operaciones. Esta cadena define el orden de ejecución de las operaciones. La cadena MS indica la máquina seleccionada para realizar cada operación, con la cual se puede conocer el tiempo de procesamiento. Con este proceso se calcula el makespan de cada individuo en la población (Paso 2).

Las cadenas OS, MS y el valor del makespan se utilizan para la búsqueda global de explotación de soluciones mediante la aplicación del AG (Paso 4,5,6).

Se genera una población refinada mediante:

- Selección. Se depura la población con dos tipos de selección, elitismo y torneo. En el primer caso solo los dos individuos más aptos pasan a la siguiente generación. En el torneo se toman aleatoriamente dos individuos y se elige al mejor (con menor makespan), repitiéndose el proceso hasta obtener una población completa depurada.
- Cruce. Se usa un operador de cruce en dos partes con 50% de probabilidad para las cadenas OS y MS, obteniendo una descendencia donde los individuos son combinación de los padres.
- Mutación. Se adoptan dos operadores (inserción e intercambio) para las cadenas OS y MS con 50% de probabilidad.

Con el AG anterior se obtiene una nueva población; en la siguiente etapa cada individuo es mejorado utilizando la búsqueda local ECR encargada de la explotación de las soluciones (Paso 7).

Se generan nuevas soluciones vecinas por cambios aleatorios de las máquinas en las cadenas MS con un 10% de probabilidad y se evalúa en la función costo. Si la nueva solución es mejor, reemplaza a la original. En caso contrario, se va guardando en una pila. Si después de un número fijo de iteraciones de la ECR la solución original no se mejora, se sustituye por una de la pila tomada de forma aleatoria y se vacía la pila, dando un reinicio a todo el proceso. La escalada se aplica a toda la población.

El algoritmo termina al cumplir la condición de iteraciones (Paso 3), regresando la mejor solución obtenida hasta el momento. Finalmente, los parámetros que se consideraron fue una población de 100 individuos y 300 iteraciones del algoritmo.

6. Resultados

Las pruebas del método propuesto se realizaron en Matlab, con un equipo de las siguientes características: sistema operativo Windows 10 home de 64 bits, con un procesador Intel® Core™ i5 a 2.3 GHz y memoria RAM de 12 GB.

Los resultados obtenidos por el algoritmo AG-ECR se comparan con otros métodos propuestos para optimizar las mismas instancias vdata, tomando los algoritmos de Li y Gao (2016), Mastrolilli (2000), y el de Hurink (1994), para verificar también si se alcanzó el mejor valor conocido del makespan para cada problema.

Tabla 1. Resultados experimentales vdata de 43 experimentos.

			Li y Gao (2016)	Mastrolillii (2000)	Hurik (1994)	
Problema edata	*n x m*	AG-ECR	HA	TS	TSN1	TSN2
mt06	6 x 6	47	47	47	47	47
mt10	10 x 10	685	686	686	737	737
mt20	20 x 5	1025	1022	1022	1028	1028
la01	10 x 5	575	570	571	577	577
la02	10 x 5	535	530	530	535	535
la03	10 x 5	481	477	478	481	486
la04	10 x 5	505	502	502	509	506
la05	10 x 5	466	457	457	460	458
la06	15 x 5	805	799	799	801	803
la07	15 x 5	750	749	750	752	752
la08	15 x 5	767	765	765	767	768
la09	15 x 5	854	853	853	859	857
la10	15 x 5	806	804	804	806	805
la11	20 x 5	1071	1071	1071	1073	1073
la12	20 x 5	937	936	936	937	937
la13	20 x 5	1038	1038	1038	1039	1039
la14	20 x 5	1071	1070	1070	1071	1071
la15	20 x 5	1090	1090	1090	1093	1093
la16	10 x 10	717	717	717	717	717
la17	10 x 10	646	646	646	646	646
la18	10 x 10	663	666	666	674	673
la19	10 x 10	698	700	700	725	709
la20	10 x 10	756	756	756	756	756
la21	15 x 10	843	835	835	861	861
la22	15 x 10	765	760	760	790	795
la23	15 x 10	857	840	842	884	887
la24	15 x 10	820	806	808	825	830
la25	15 x 10	795	789	791	823	821
la26	20 x 10	1091	1061	1061	1086	1087
la27	20 x 10	1126	1089	1091	1109	1115
la28	20 x 10	1105	1079	1080	1097	1090
la29	20 x 10	1018	997	998	1016	1017
la30	20 x 10	1101	1078	1078	1105	1108
la31	30 x 10	1552	1521	1521	1532	1533
la32	30 x 10	1673	1659	1659	1668	1668
la33	30 x 10	1523	1499	1499	1511	1507
la34	30 x 10	1558	1536	1536	1542	1543
la35	30 x 10	1563	1550	1550	1559	1559
la36	15 x 15	1027	1028	1030	1054	1071
la37	15 x 15	1073	1074	1077	1122	1132
la38	15 x 15	954	960	962	1004	1001
la39	15 x 15	1023	1024	1024	1041	1068
la40	15 x 15	964	970	970	1009	1009

El resultado para los 43 problemas vdata se muestran en la Tabla 1. Se concluye que en el 18.6% de los casos se alcanzó el mejor valor óptimo conocido hasta el momento, superando a los otros algoritmos (en color amarillo). En otro 16.27% de los problemas se obtuvo el mejor valor óptimo junto con los otros algoritmos comparados (en color azul). El resto de los resultados quedaron muy cerca del mejor valor conocido, mostrando en general un resultado satisfactorio para el algoritmo propuesto. En la Figura 6 se presenta una corrida del algoritmo AG-ECR para el problema 1a40 de Hurink, Jurisch, y Thole (1994), obteniendo un mejor resultado en comparación con los otros métodos después de 120 iteraciones.

```
Command Window
>> AGEC
Iteracion: 20 Makespan: 1159
Iteracion: 40 Makespan: 1070
Iteracion: 60 Makespan: 1038
Iteracion: 80 Makespan: 1009
Iteracion: 100 Makespan: 984
Iteracion: 120 Makespan: 970
Iteracion: 140 Makespan: 964
Iteracion: 160 Makespan: 964
Iteracion: 180 Makespan: 964
Iteracion: 200 Makespan: 964
Iteracion: 220 Makespan: 964
Iteracion: 240 Makespan: 964
Iteracion: 260 Makespan: 964
Iteracion: 280 Makespan: 964
Iteracion: 300 Makespan: 964
Solucion: 964
Tiempo: 294.9852
```

Figura 6. Resultados de la corrida computacional del problema vdata la40

En la Figura 7 se exhibe la convergencia del problema la40 conforme transcurren las iteraciones.

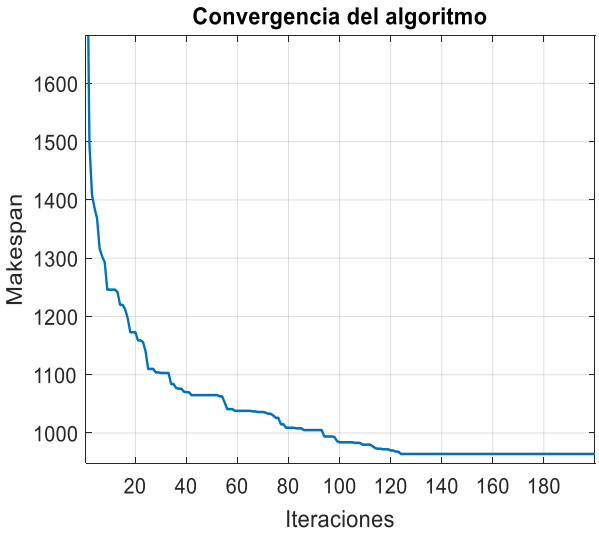

Figura 7. Convergencia del makespan para el problema vdata la40.

En la Figura 8 se muestra también la convergencia del makespan aplicado al problema la31, observando que pasando de la iteración 100, se va estabilizando la convergencia del AG-ECR.

Como ejemplo se presentan 3 diagramas de Gantt del conjunto vdata de Hurink, Jurisch, y Thole (1994), para comprobar la factibilidad de las soluciones en las instancias mt06, mt10 y mt20 (ver Figuras 9, 10 y 11). La instancia mt06 cuenta con 6 trabajos y 6 máquinas, la mt10 tiene 10 trabajos y 10 máquinas, y finalmente la mt20 tiene 20 trabajos y 5 máquinas.

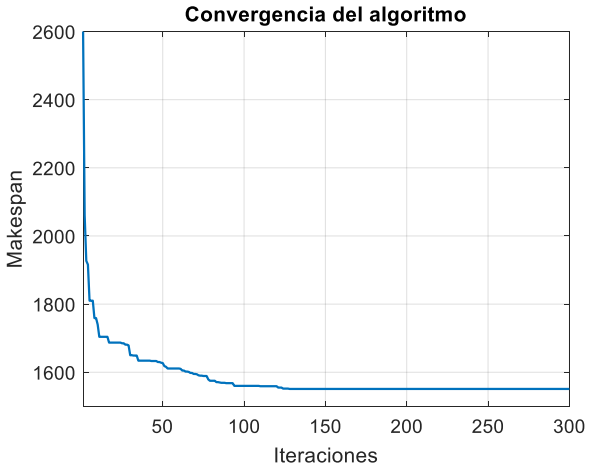

Figura 8. Convergencia del makespan para el problema vdata la31.

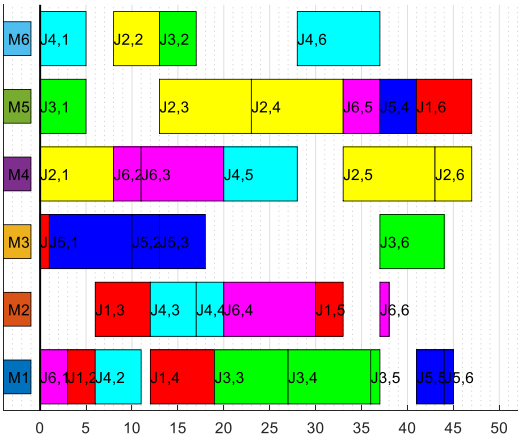

Figura 9. Diagrama de Gantt del problema vdata mt06.

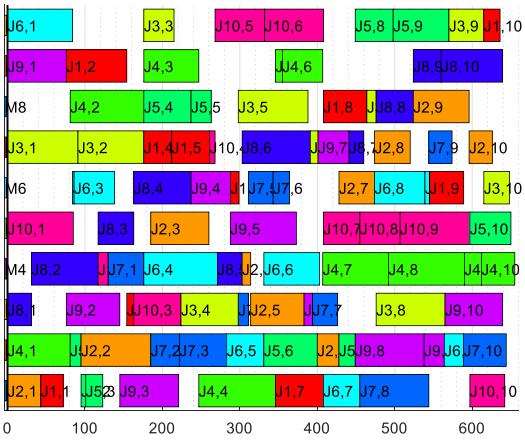

Figura 10. Diagrama de Gantt del problema vdata mt10.

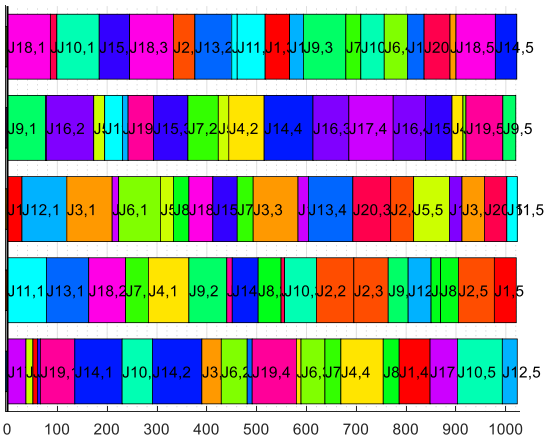

Figura 11. Diagrama de Gantt del problema vdata mt20.

7. Conclusiones

La implementación de nuevos algoritmos metaheurísticos para el FJSSP sigue siendo de interés ya que cada vez los sistemas de producción implican más tareas, más operaciones y una mayor flexibilidad para encontrar una programación eficiente. Al no existir un método general de solución, se requieren nuevas técnicas que hagan frente a este tipo de problemas.

Este trabajo propuso un algoritmo híbrido (AG-ECR) utilizando como búsqueda global a los AG por su buena capacidad de búsqueda en el espacio de soluciones, complementado por la ECR como método de búsqueda local para encontrar una mejor asignación de máquinas para cada operación, con la ventaja que tiene poco costo computacional. Con el AG-ECR se mejoraron los resultados de un 18.6% de los problemas en comparación con los otros métodos tomados como referencia, y se igualaron los resultados en un 16.27%, lo que demuestra que el algoritmo propuesto es competitivo con algoritmos de última generación.

Como trabajo futuro se plantea utilizar la ruta crítica y seguir trabajando con una vecindad simplificada en donde se seleccionen ya sea de forma aleatoria operaciones críticas o una operación del trabajo que se repita más en la ruta, para tratar de alcanzar los valores óptimos en todos los problemas de prueba.

Agradecimientos

Este estudio ha sido realizado gracias al apoyo del CONACYT con números de proyecto CB- 2017-2018-A1-S-43008 y FOP16-2021-01-320109, al respaldo de la Universidad Autónoma del Estado de Hidalgo, del Área Académica de Ingeniería y Arquitectura y del Doctorado en Ciencias en Ingeniería con Énfasis en Análisis y Modelación de Sistemas. Nayeli J. Escamilla Serna agradece el apoyo recibido como becaria CONACYT con número de registro 1013175.

Referencias

Aarts, E. H., & Korst, J. H. (1989). Simulated Annealing and Boltzmann Machines.

Adiri, I., Bruno, J., Frostig, E., & Rinnooy Kan, A. (1989). Single machine flow-time scheduling with a single breakdown. (Springer-Verlag, Ed.) 26(7), 679–696.

Alzaqebah, M., Jawarneh, S., Alwohaibi, M., Alsmadi, M. K., Almarashdeh, I., & Mohammad, R. M. (2020). Hybrid Brain Storm Optimization algorithm and Late Acceptance Hill Climbing to solve the Flexible Job-Shop Scheduling Problem. Journal of King Saud University – Computer and Information Sciences, 1-12.

Amjad, M. K., Butt, S. I., Kousar, R., Riaz, A., Agha, M., Faping, Z., . . . Asghe, U. (2018). Recent Research Trends in Genetic Algorithm Based Flexible Job Shop Scheduling Problems. Mathematical Problems en Engineering, 32.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. Annals of operations research, 41(3), 157-183.

Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. Computing, 45(4), 369-375.

Chen, R., Gen, M., & Tsujimura, Y. (1996). A tutorial Survey of job-shop scheduling. Elsevier Science Ltd, 983-997.

Dai, M., Tang, D., Giret, A., & Salido, M. (2019). Multi-objective optimization for energy-efficient flexible job shop scheduling problem with transportation constraints. Robotics and Computer Integrated Manufacturing, 59, 143-157.

Defersha, F. M., & Rooyani, D. (2020). An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time. Computers & Industrial Engineering, 147, 19.

Deng, Q., Gong, G., Gong, X., Zhang, L., Liu, W., & Ren, Q. (2017). A bee evolutionary guiding nondominated sorting genetic algorithm II for multiobjetive flexible job shop scheduling. 1-20.

Ding, H., & Gu, X. (2020). Hybrid of human learning optimization algorithm and particle swarm optimization algorithm with scheduling strategies for the flexible job-shop scheduling problem. Neurocomputing, 414, 313-332.

Dong, X., Chen, P., Huang, H., & Nowak, M. (2013). A multi restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time. Computers & Operations Research, 627-632.

Dorigo, M. (1992). Optimization, Learning and Natural Algorithms,. PhD thesis. Italie.

Fisher, H., & Thompson, G. (1963). Probabilistic learning combinations of local job-shop scheduling rules. Englewood Cliffs, 225-51.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). rtificial Intelligence through Simulated Evolution. New York, USA: AWiley.

Gao, L., Peng, C., Zhou, C., & Li, P. (2016.). Solving flexible job shop scheduling problem using general particle swarm optimization. In:Proceedings of the 36th CIE Conference on Computers and Industrial Engineering, (págs. pp. 3018-3027).

Garey, M., Johnson, D., & Sethi, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. Mathematics of Operations Research, 1(2), 117-129.

Glover, F. (1989). Tabu seach - Part I. ORSA Journal on Computing 1, 190-206.

Goldberg, D. E. (1989). Genetic algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA.

Holland, H. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, 208.

Hurink, E., Jurisch, B., & Thole, M. (1994). Tabu search for the job shop scheduling problem with multi-purpose machine. Operations Research Spektrum 1, 15, 205–215.

Kennedy, J., & Eberhart, R. (1995). Particle Swarm Optimization. Purdue School of Engineering and Technology Indianapolis.

Kern, W. (1989). A probabilistic analysis of the switching algorithm for the Euclidean TSP. Mathematical Programming, 44, 213-219.

Koza, J. R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. Mitt.

Lawrence, S. (1984). esource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement). Pittsburgh, Pennsylvania, Carnegie-Mellon University: Graduate School of Industrial Administration.

Li, X., & Gao, L. (2016). An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. Int. J. Production Economics, 174, 93-110.

Mailing, G. (2003). Algortitmos heurísticos y el problema de job shop scheduling. Buenos Aires: Facultad de Ciencias Exactos y Naturales.

Mastrolilli, M., & Gambardella, L. (2000). Effective neighbourhood functions for the flexible job shop problem. Journal of Scheduling, 3(1), 3-20.

Michallet, J., Prins, C., Amodeo, L., Yalaoui, F., & Vitry, G. (2014). Multi-Start iterated local search for the periodic vehicle routing problem with time windows and time spread constraints on services. Computers & Operacions Research, 41, 196-207.

Rechenberg, I. (1973). Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. Frommann-Holzboog, Stuttgart, Germany.

Rodriguez, K. E., de Aguilar, A. G., & Hideaki, T. R. (2018). A new approach to solve the flexible job shop problem based on a hybrid particle swarm optimization and Random- Restart Hill Climbing. Computers and Industrial Engineering, 178-189.

Salido, M. A., Escamilla, J., Giret, A., & Barber, F. (2016). A genetic algorithm for energy- efficiency in job-shop scheduling. Springer- Verlag London, 85, 1303-1314.

Shen, L., Dauzère-Pérès, S., & Neufeld, J. (2018). Solving the flexible job shop scheduling problem with sequence-dependent setup times. European Journal of Operational Research, 503-516.

Sreekara, M., Ratnam, C., Rajyalakshmi, G., & Manupati, V. (2018). An effective hybrid multi objective evolutionary algorithm for solving real time event in flexible job shop scheduling problem. Measurement, 114, 78-90.

Xia, W., & Wu, Z. (m de 2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. Computers & Industrial Engineering, 48(2), 409-425.

Xie, N., & Chen, N. (2018). Flexible job shop scheduling problem with interval grey processingtime. Applied Soft Computing, 70, 513–524.

Yang, Y., Huang, M., Yu, Z., & Bing, Q. (2020). Robust scheduling based on extreme learning machine for bi-objective flexible job-shop problems with machine breakdowns. Expert Systems with Applications, 158, 1-12.

Zhang, G., Shao, X., Li, P., & Gao, L. (2009). An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. Computers & Industrial Engineering(56), 1309–1318.

Zobolas, G. I., Tarantilis, C., & Ioannou, G. (2008). Exact, heuristic and meta-heuristic algorithms for solving shop scheduling problems. En F. Xhafa, & A. Abraham, Metaheuristics for Scheduling in Industrial and Manufacturing Applications (Vol. 128, págs. 1-40). Studies in Computational Int