Un Algoritmo evolutivo híbrido para el problema de programación del taller de flujo permutado con restricciones de turno

Hybrid evolutionary algorithm for the permutation flow shop scheduling problem with shifts constraints

Manuel Eduardo García Jiménez[1] Omar Alexis Becerra Sierra[2] Juan Carlos Rivera[3]

Recibido 6 de Enero de 2021, aceptado 17 de Mayo de 2021
Received: January 6, 2021 Accepted: May 17, 2021

RESUMEN

Un Flow Shop es un sistema de producción en el cual una serie de trabajos debe seguir un flujo unidireccional para ser procesada en varias estaciones de trabajo. En este artículo se introduce el Permutation Flow Shop Scheduling Problem with Shifts Constraints (PFSSPSC), una variante del Permutation Flow Shop Scheduling Problem (PFSSP) en donde se busca minimizar el tiempo en que se terminan de procesar todos los trabajos y se imponen restricciones sobre los turnos de procesamiento de los trabajos. Estas restricciones consisten en que la ejecución de cada trabajo siempre debe terminar en el mismo turno en que empieza para evitar interrupciones. Además, en este artículo se introduce un algoritmo híbrido para hallar buenas soluciones al PFSSPSC. Éste consiste en un metaheurístico compuesto por un algoritmo genético y un algoritmo VNS (Variable Neighborhood Search). El algoritmo propuesto es probado con 120 instancias del PFSSP disponibles en la literatura adaptadas para el PFSSPSC. Las soluciones obtenidas son comparadas con aquellas ofrecidas por tres algoritmos heurísticos en cuanto a tiempo de ejecución y calidad de las soluciones obtenidas: un método constructivo, un método constructivo aleatorizado y un VND. La comparación entre los cuatro algoritmos muestra una superioridad de los algoritmos híbrido y VND en cuanto a la calidad de la solución, aunque el método constructivo y la búsqueda aleatoria son más rápidos. Entre el algoritmo híbrido y el VND no hay una gran diferencia en la calidad de la solución, pero el algoritmo híbrido es significativamente más rápido que el VND.

Palabras clave: Algoritmo evolutivo, Algoritmo híbrido, Programación del taller de flujo permutado, Restricciones de turno.

ABSTRACT

A Flow Shop is a production system in which a series of jobs must follow a one-directional flow to be processed in several workstations. This article introduces the Permutation Flow Shop Scheduling Problem with Shifts Constraints (PFSSPSC), a variant of the Permutation Flow Shop Scheduling Problem (PFSSP) where it is sought to minimize the time in which all jobs are finished and shift constraints are imposed. These constraints consist of forcing the execution of each job to always end in the same shift in which they begin to avoid interruptions. In addition, this article introduces a hybrid algorithm to find good solutions to the PFSSPSC. This consists of a metaheuristic composed by a Genetic Algorithm

[1] Universidad EAFIT. Departamento de Ciencias Matemáticas. Medellín, Colombia. E-mail: megarciaj@eafit.edu.co
[2] Universidad EAFIT. Departamento de Ciencias Matemáticas. Medellín, Colombia. E-mail: oabecerras@eafit.edu.co
[3] Universidad EAFIT. Departamento de Ciencias Matemáticas. Medellín, Colombia. E-mail: jrivera6@eafit.edu.co
* Autor de correspondencia: megarciaj@eafit.edu.co

and a Variable Neighborhood Search (VNS). The proposed algorithm is tested with 120 instances for the PFSSP available in literature adapted for the PFSSPSC. The obtained solutions are compared with those offered by three heuristic algorithms in terms of execution time and solution quality: a constructive algorithm, a randomized constructive method and a VND. The comparison among the four algorithms shows a superiority of hybrid and VND algorithms in terms of the solution quality, even if constructive and random search are faster. Among the hybrid algorithm and the VND there are not a great difference on solution quality, but the hybrid algorithm is significantly faster than VND.

Keywords: Evolutionary algorithm, Hybrid algorithm, Permutation flow shop scheduling, Shift constraints.

INTRODUCCIÓN

Un sistema productivo tipo *"Flow Shop"* es un ambiente en el que un conjunto de tareas debe seguir un flujo unidireccional para ser procesadas en múltiples estaciones de trabajo, donde cada estación consta de una sola máquina. En términos simples, se tienen *n* trabajos, los cuales han de pasar cada uno por todas las *m* máquinas del flujo, estación por estación. Cada máquina puede procesar un solo trabajo a la vez y no se puede interrumpir una tarea después de iniciada, de manera que no puede comenzar a realizar una tarea sin haber acabado la anterior. Los trabajos solo pueden pasar por una máquina a la vez, de modo que, para que una máquina inicie la ejecución de un trabajo se debe esperar a que la máquina anterior haya terminado el procesamiento de dicho trabajo. La ejecución de cada trabajo en cada máquina es caracterizada por su tiempo de procesamiento. La operación del Flow Shop implica elegir el orden en que se procesan los trabajos, más no sobre el orden en que éstos pasan por las máquinas. Así, el "Flow Shop Scheduling Problem" (FSSP) es el problema de optimización que busca encontrar un orden en el cual realizar los trabajos que minimice el tiempo de terminación de todos los trabajos.

El *"Permutation Flow Shop"* es un sistema de producción en el cual, además de las condiciones antes mencionadas, los trabajos son realizados por todas las máquinas en el mismo orden. Esto quiere decir que, si la primera máquina realiza un trabajo primero que todos los demás, este trabajo debe ser procesado de primero por todas las máquinas subsecuentes. Lo mismo ocurre con el segundo trabajo y así se sigue hasta que se procesen todos los trabajos.

En este artículo se introduce una nueva variante del problema donde se tiene en cuenta que los operarios de las máquinas trabajan por turnos, y el cambio de operarios implicaría la interrupción del procesamiento de los trabajos. Así, el *"Permutation Flow Shop Scheduling Problem with Shifts Constraints"* (PFSSPSC) consiste en un PFSSP donde el personal opera por turnos de una duración dada. Esto implica que los trabajos iniciados en un turno determinado deben terminar en el mismo turno para evitar interrupciones en el procesamiento de los trabajos debido al cambio de personal o incluso cambio de jornada en algunas aplicaciones. El objetivo es hallar un orden en el que se minimice el tiempo de finalización del último trabajo.

El PFSSP se ha convertido en un tema de interés desde que aparecieron los primeros trabajos en 1970, dado que es un problema de mucha complejidad y con aplicación práctica. En la actualidad este problema es estudiado por muchos investigadores para proponer algoritmos exactos y metaheurísticos. Buenas soluciones pueden generar ahorros en tiempo, personal y energía en fábricas con este flujo de trabajo. Debido a la complejidad asintótica del problema, éste se clasifica como NP-Hard [1], por lo tanto, se hace necesario el diseño de algoritmos metaheurísticos que obtengan soluciones de buena calidad en tiempos razonables.

Los algoritmos genéticos son métodos de optimización y búsqueda de soluciones basados en los postulados de la evolución biológica. Las ideas de estos algoritmos provienen de la imitación de la selección natural observada en la naturaleza [2], en las cuales entidades se mezclan y compiten entre sí, de tal manera que solo las más aptas son capaces de prevalecer frente a las demás entidades. Solo aquellos individuos que sobreviven son los que tienen la posibilidad de reproducirse, lo cual promueve el cruce entre las soluciones de mejor calidad. Debido a esto, con el tiempo la población va mejorando y nuevas soluciones mejores van emergiendo.

Otra clase de algoritmo para resolver problemas de optimización computacionalmente difíciles es la familia de métodos basados en búsqueda local. Estos métodos se basan en la premisa que la mayoría de los problemas se pueden formular en términos de espacios de búsqueda; iniciando en una solución candidata, se mueven iterativamente a una solución vecina. Típicamente, cada solución candidata tiene más de una solución vecina y la elección se toma empleando aquella información sobre las soluciones cercanas a la actual, de ahí el nombre búsqueda local.

El Flow Shop ha sido abordado desde varios enfoques y algunas variantes del problema han sido solucionadas utilizando diferentes estrategias heurísticas. Un algoritmo constructivo es presentado por Muhammad *et al.* [4], el cual funciona especialmente bien en grandes problemas de flujo en los entornos de secuenciación estática y dinámica. Jian y Rong [5] proponen otro algoritmo constructivo el cual inserta un grupo de trabajos en la solución al mismo tiempo, en lugar de insertar un trabajo a la vez como las reglas de despacho tradicionales. Rajkumar *et al.* [6] desarrollan un algoritmo GRASP para la integración de la planificación de procesos con la programación de producción en un entorno de trabajo flexible. El metaheurístico GRASP es un algoritmo iterativo que genera una solución inicial diferente en cada iteración y la mejora utilizando procedimientos basados en búsqueda local.

Para este trabajo se utilizan las ventajas que provee la construcción de un algoritmo híbrido. Una vez realizadas varias generaciones del algoritmo genético, la mejor solución es mejorada utilizando un VNS.

Los resultados se comparan y discuten para identificar si los algoritmos propuestos son adecuados para resolver el PFSSPSC. Dado que encontrar la solución óptima al PFSSPSC es un desafío, es importante realizar un análisis de los algoritmos y su comportamiento en este contexto e identificar las ventajas y desventajas de estos métodos en esta área.

El resto del artículo está compuesto por cinco secciones. La sección 2 presenta una revisión de la literatura sobre algunos métodos de solución heurística a problemas relacionados. Una descripción formal del PFSSPSC y una formulación matemática son descritas en la sección 3. Los métodos de solución propuestos son introducidos en la sección 4. En la sección 5 se presentan los experimentos computaciones realizados. Finalmente, se presentan las conclusiones y trabajos futuros en la sección 6.

TRABAJOS RELACIONADOS

En la Tabla 1 se presenta un resumen de la revisión de literatura clasificada de acuerdo con el tipo de

Tabla 1. Tabla de referencias.

Titulo	Tipo de Algoritmo	Año
Hybrid genetic algorithms for solving reentrant flow-shop scheduling with time window [1]	Algoritmo Genético	2013
Multi-objective optimization using evolutionary algorithms [2]	Algoritmo Genético	2001
A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem [4]	Constructivo	1983
An NEH-based heuristic algorithm for distributed permutation flow-shop scheduling problems [5]	Constructivo	2001
A GRASP algorithm for the integration of process planning and scheduling in a flexible job-shop [6]	GRASP	2010
Evolutionary algorithms for scheduling a flow-shop manufacturing cell with sequence dependent family setups [7]	Algoritmo Genético	2005
Evolutionary algorithms for job-shop scheduling [8]	Algoritmo Genético	2004
Aplicación de un algoritmo ACO al problema de taller de flujo de permutación con tiempos de preparación dependientes de la secuencia y minimización de makespan [11]	ACO	2011
An improved tabu search approach for solving the job shop scheduling problem with tooling constraints [12]	Búsqueda tabú	1996
Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs [13]	ACO	2004
GRASP for set packing problems [14]	GRASP	2004
An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion [15]	Búsqueda local iterada	2009
Genetic Algorithms, Path Relinking, and the Flowshop Sequencing Problem [16]	Algoritmo Genético	1998

métodos de solución empleado. Ahí se mencionan los trabajos que se consideran relevantes para la solución del problema, además de otras variantes.

Vale la pena mencionar que todos estos artículos solo dan con soluciones permutadas, es decir, con el mismo orden de ejecución de los trabajos para cada máquina. Dicha característica también es considerada en el problema abordado en este artículo.

Entre los trabajos mencionados en la Tabla 1, hay algunos que sobresalen por sus enfoques particulares. Francca *et al.* [7] proponen un algoritmo evolutivo en el cual la población está conformada por un árbol ternario donde los nodos hojas son variaciones de su nodo raíz, el cual es un orden creado aleatoriamente. Cuando se van a cruzar elementos, se escogen al azar un nodo raíz y una de sus tres hojas para formar un nuevo individuo. Todos los nuevos nodos se vuelven parte de la población y con movimientos locales se crean sus tres hojas. Este algoritmo, aunque ingenioso, solo realiza cruces entre nodos relacionados, las nuevas soluciones tienden a parecerse mucho a sus padres, lo cual puede llevar a una baja diversidad en la población.

Mesghouni *et al.* [8] propone un algoritmo evolutivo en el cual cada solución es representada por una lista de trabajos para cada máquina y se generan nuevas soluciones (hijos) por medio de un operador de cruce en donde para cada posición en cada máquina en la nueva solución, se escoge entre los trabajos que los padres realizan en esa misma posición. Entre los dos trabajos elegibles, se escoge al trabajo que tarde menos en ser realizado en la máquina escogida. Luego se asegura factibilidad de la solución reemplazando trabajos repetidos con los trabajos faltantes. Este mecanismo para recuperar factibilidad es utilizado en el algoritmo propuesto en este artículo, debido a su eficiencia.

En cuanto a los métodos basados en búsqueda local sobresale el metaheurístico ILS *(iterated local search)*. El ILS propuesto por Dong *et al.* [15] consiste en tomar una solución representada como una lista ordenada de trabajos, y recorrer la totalidad de las soluciones vecinas creadas a partir del intercambio de las posiciones de dos trabajos.

Hasta donde los autores conocen, no hay ningún trabajo reportado en la literatura para resolver el PFSSPSC. Sin embargo, varios de los métodos antes mencionados fueron adaptados para crear soluciones al PFSSPSC con el fin de compararlos con el metaheurístico propuesto.

METODOLOGÍA

Descripción de problema

El FSSPSC consiste en un entorno de producción donde hay *N* trabajos a ser ejecutados y *M* máquinas que los deben procesar. Cada trabajo debe ser realizado por cada máquina exactamente una vez y en el orden de las máquinas. Cada trabajo *j* tiene un tiempo de ejecución distinto en cada máquina i denotado p_{ij}. Adicionalmente, el horizonte de planeación está dividido en turnos de trabajo de longitud *L*. Con el objetivo de evitar interrupciones en la ejecución de los trabajos, se debe garantizar que cada trabajo inicie y termine en el mismo turno de trabajo. Se asume por lo tanto, sin pérdida de generalidad, que $L \geq pij$ para todo trabajo *j* y toda máquina i. El objetivo es encontrar un orden π en el cual realizar los trabajos, de manera que se minimice el tiempo total de la operación, makespan. El makespan es el tiempo después del inicio en el cual la última máquina termina con el último trabajo de la secuencia π. Una secuencia π solo es tomada como factible si bajo ella todos los trabajos se realizan exactamente una vez [9]. Denotando a la secuencia de trabajos escogida como $\pi = (\pi_1, \pi_2,\ldots,\pi_N)$ y a $C(\pi_j, i)$ como el tiempo en el que se completa el trabajo j del orden en la máquina i, lo que se busca minimizar es $C(\pi n, M)$, lo cual es igual al tiempo de inicio del último trabajo de π en la última máquina, más el tiempo de ejecución de éste [10].

Formulación matemática

El problema de optimización descrito en la sección anterior se puede formular matemáticamente utilizando el modelo (1)–(6).

$$\min Z = S_{M,\pi_n} + p_{M,\pi_n} \tag{1}$$

$$S_{i,\pi_j} \geq S_{i-1,\pi_j} + p_{i-1,\pi_j} \quad \forall i \in \{2,\ldots,M\}, j \in \{1,\ldots,N\} \tag{2}$$

$$S_{i,\pi_j} \geq S_{i,\pi_{j-1}} + p_{i,\pi_{j-1}} \quad \forall i \in \{1,\ldots,M\}, j \in \{2,\ldots,N\} \tag{3}$$

$$\left\lfloor \frac{S_{i,\pi_j}}{L} \right\rfloor = \left\lfloor \frac{S_{i,\pi_j} + p_{i,\pi_j}}{L} \right\rfloor \quad (4)$$
$$\forall i \in \{1,\ldots,M\}, j \in \{1,\ldots,N\}$$

$$\pi \in \mathbb{R}^n, j \in \pi \quad \forall j \in \{1,\ldots,N\} \quad (5)$$

$$S_{j,j} \geq 0 \quad \forall i \in \{1,\ldots,M\}, j \in \{1,\ldots,N\} \quad (6)$$

donde Si, j es el tiempo de inicio del trabajo j en la máquina i, y x representa el mayor entero menor que x.

La ecuación (1) representa la función objetivo, la minimización de makespan, C (π_n, M). Las inecuaciones (2) indican que en cada máquina i el tiempo de inicio de cualquier trabajo j debe ser mayor o igual al tiempo de finalización del mismo trabajo en la máquina anterior. Las inecuaciones (3) implican que los tiempos de inicio de cualquier trabajo en cada máquina deben ser mayores o iguales al tiempo de finalización del trabajo anterior de π en la misma máquina. Las ecuaciones (4) obligan a que todos los trabajos tangan que terminar en el mismo turno en el que comienzan. Las expresiones (5) implican que nuestra variable de decisión π debe ser un vector de longitud N el cual contenga a todos los trabajos a ser realizados, mientras que las inecuaciones (6) indican que las variables de decisión Si, j deben ser no negativas.

METODOS Y MATERIALES

Algoritmo evolutivo propuesto

El algoritmo evolutivo propuesto está compuesto por dos algoritmos que actúan secuencialmente: un algoritmo genético y un VND (variable neighborhood descent). El algoritmo inicia con la generación de una población de soluciones factibles. A continuación, generación tras generación se busca el mejoramiento de este conjunto a través de operadores genéticos: selección, cruce, mutación y actualización de la población. Una vez terminadas todas las generaciones del algoritmo genético, la mejor solución de la población final es mejorada utilizando un algoritmo VND con dos vecindarios. La Figura 1 muestra el pseudocódigo del algoritmo propuesto.

En las siguientes secciones se describen en detalle los componentes del algoritmo.

Algoritmo Genético

Este algoritmo comienza con la generación de un conjunto de individuos (soluciones) iniciales generadas aleatoriamente (población inicial) de tamaño pob_tam. Cada individuo de esta población consiste en una lista ordenada de trabajos. A cada lista ordenada le corresponde un único programa con tiempos de inicio y finalización de cada trabajo en cada máquina y un valor de la función objetivo (tiempo de finalización del último trabajo ejecutado en la última máquina.

En cada iteración se generan num_hijos hijos. Cada hijo es generado por un proceso de cruce de dos individuos. Dichos individuos son seleccionados utilizando una variante del método de la ruleta. Así, cada individuo πk tiene una probabilidad de ser seleccionado $P(\pi k)$. Dicha probabilidad es calculada utilizando la ecuación (7)

$$P(\pi_k) = \frac{\frac{1}{(1+r(\pi_k))}}{\sum_{h=1}^{num\_hijos} \frac{1}{(1+r(\pi_h))}} \quad (7)$$

```
Datos: pop_tam,gen. Ambos enteros
Resultado: Orden. Lista de enteros
Función: Gen_VNS(pob_tam,gen):
Inicializa p_mut, nh, pob
para todas las gen hacer
    probs←Asignar_Probabilidades(pob)
    para todos los nh hacer
        padres←seleccionar_padres(pob)
        hijo←cruce(padres)
        hijo←correccion(hijo)
        si aleatorio < p_mut entonces
        hijo←mutacion(hijo)
        pob←pob+hijo
    pob←seleccionar_mejores(pob, pob_tam)
mejor←seleccionar_mejor(pob)
vecindario1←Crear_vecindario1(mejor)
mejorv1←escoger_mejor(vecindario1)
vecindario2←Crear_vecindario2(mejorv1)
mejorv2←escoger_mejor(vecindario2)
retorna mejorv2
```

Figura 1. Pseudocódigo del algoritmo propuesto.

donde $r(\pi_k)$ es el ranking del individuo π_k, es decir, la posición del individuo π_k al ordenar la población de menor (mejor) a mayor (peor) valor de la función objetivo.

La información genética de los dos individuos seleccionados, π^{p1} y π^{p2}, es entonces cruzada para la creación de un nuevo individuo, π^{h}. El cruce consiste en seleccionar aleatoriamente, para cada posición j del vector π^{h}, un valor entre el j-ésimo trabajo en el individuo π^{p1} y el j-ésimo trabajo en el individuo π^{p2}. Este método de cruce con alta frecuencia genera soluciones infactibles, soluciones con trabajos repetidos y trabajos no programados. Para reparar la solución se utiliza el procedimiento denominado corrección, en el cual se enlistan los trabajos que no han sido programados y los que se repiten en la solución. Después de esto se recorre el vector π^{h} reemplazando cada trabajo repetido en el vector π^{h} por otro de la lista de trabajos no programados. Este procedimiento convierte cualquier solución obtenida producto de un cruce, en una solución factible.

Una vez que se obtiene una solución factible, cada solución hija, π^{h}, puede ser mutada con una probabilidad prob_mutacion. La mutación consiste en elegir aleatoriamente dos posiciones, j_1 y j_2, del vector π^{h} e intercambiar sus valores. Dicha mutación no afecta la factibilidad de la solución πh.

Una vez que se generan num_hijos hijos, se hace una actualización de la población en la cual sobreviven los pob_tam individuos más aptos, aquellos con mayor valor en la función objetivo.

El proceso se repite por número determinado de *generaciones*. Una vez terminadas todas las generaciones, el mejor individuo obtenido en la población final es mejorado utilizando un algoritmo VND.

VND (Variable Neighborhood Descent)
El VND propuesto consiste en exploración de dos tipos de vecindarios diferentes, los cuales son alternados sistemáticamente para mejorar la mejor solución obtenida por el algoritmo genético.

Los dos vecindarios utilizados se describen a continuación:

- Vecindario 1: *Interchange of two adjacent jobs.* Este vecindario consiste en el intercambio de las posiciones de dos trabajos adyacentes, π_i y π_{i+1}. Por ejemplo, dada la secuencia de trabajos π = [1, 2, 3, 4, 5], se pueden obtener cuatro soluciones vecinas: [2, 1, 3, 4, 5], [1, 3, 2, 4, 5], [1, 2, 4, 3, 5] y [1, 2, 3, 5, 4].
- Vecindario 2: *Interchange of two pairs of adjacent jobs*. Este vecindario funciona de manera similar al vecindario anterior: se toman cuatro trabajos consecutivos en una solución y se intercambian las posiciones de los dos primeros con las posiciones de los dos finales. Por ejemplo, dada la secuencia de trabajos π = [1, 2, 3, 4, 5], se pueden obtener dos soluciones vecinas: [3, 4, 1, 2, 5] y [1, 4, 5, 2, 3].

EXPERIMENTACIÓN COMPUTACIONAL

Para realizar la experimentación y análisis del desempeño del algoritmo propuesto se realiza la implementación en la versión 3.8.3 de Python, con un computador con procesador Intel CORE i5-4210U a 1.70GHz.

En esta sección se presentan las instancias de prueba, se discute la selección de parámetros, se evalúa el desempeño de los diferentes componentes del algoritmo y se comparan los resultados con otros algoritmos.

Instancias de prueba
Para la experimentación y prueba del algoritmo se utilizaron 120 instancias. Dichas instancias corresponden a 10 muestras de 12 combinaciones diferentes de cantidades de trabajos y máquinas la cuales fueron propuestas por *Taillard* [18] para el Flow Shop Scheduling Problem. La cantidad de trabajos están entre 20 y 500, mientras que el número de máquinas está entre 5 y 20. Para la longitud de los turnos se utilizó un valor de 100 unidades de tiempo. Taillard [18] no propone instancias con longitudes de turnos, éstas fueron agregadas en esta investigación con el fin de probar los algoritmos para el FSSPSC. El valor de 100 unidades de tiempo fue utilizado debido a que los tiempos de procesamiento de los trabajos oscilan entre 1 y 100 unidades de tiempo.

Selección de parámetros
Es importante una buena parametrización de los algoritmos para lograr soluciones más robustas, además que ayudan a que el algoritmo converja más rápido a una buena solución. Teniendo esto en

mente se buscan entre los posibles parámetros del algoritmo a los que tengan mayor impacto sobre las funciones objetivo. Los parámetros a analizar están relacionados con el algoritmo genético debido a que el VND no considera parámetros.

Se identificaron cuatro parámetros: el tamaño de la población *(pob_tam)*, el número de generaciones *(generaciones)*, el número de hijos a generar en cada generación *(num_hijos)* y la probabilidad de mutación *(prob_mutacion)*.

La experimentación preliminar mostró poca variación en los resultados ante diferentes valores de los parámetros *num_hijos* y *prob_mutacion*.

Para la elección de los valores de los parámetros *pob_tam* y generaciones se probaron varias combinaciones de dichos valores en diferentes instancias para determinar cuál era la mejor. En las Figuras 2 y 3 se pueden visualizar los valores promedio del tiempo de cómputo y de la diferencia porcentual entre el valor de la función objetivo y una cota inferior al solucionar las 120 instancias de prueba con los diferentes valores de los parámetros. El valor de la cota inferior para cada instancia (z') fue calculada como el máximo de la suma de tiempos de ejecución de todos los trabajos en cada máquina (ver ecuación (8).

$$z' = \max_{i \in M} \left(\sum_{j=1}^{N} p_{i,j} \right) \tag{8}$$

La distancia porcentual a la cota inferior, D, es calculado de acuerdo a la ecuación (9). Dicha distancia puede ser interpretada como una aproximación a la distancia a la solución óptima, de manera que entre menor sea dicha distancia, mejor es la solución obtenida.

$$D = \left(\frac{z - z'}{z'} \right) \times 100 \tag{9}$$

Con relación a los tiempos de cómputo, se nota un aumento de éstos a medida que aumenta el número de máquinas y de trabajos. Sin embargo, en términos absolutos, no hay mucha diferencia debido a que la máxima diferencia entre dos combinaciones de los valores de los parámetros es de aproximadamente 3 segundos. Por lo tanto, se dio mayor importancia a la desviación porcentual para la parametrización del algoritmo.

Figura 2. Comportamiento de los tiempos de cómputo promedios al variar el tamaño de la población y el número de generaciones.

Los valores de los parámetros elegidos, y con los que se realizaron los experimentos descritos en las siguientes secciones, son *pob_tam* igual a 20 y número de generaciones igual a 20.

Las Tablas 2 y 3 muestran los tiempos de promedio ejecución y distancias porcentual promedio a la cota inferior para las 120 instancias del problema.

En la Tabla 2 se puede ver que, como era de esperarse, los tiempos de cómputo aumentan proporcionalmente al número de trabajos y de máquinas. Es de notar que los operadores genéticos tienen una relativamente baja complejidad computacional. Por ejemplo, el operador de cruce tiene una complejidad $O(n)$. Sin embargo, la evaluación de la función objetivo implica una complejidad $O(n \cdot m)$.

Las distancias promedio a las cotas inferiores están entre 87,91% y 209,29%, con un promedio de 114,41%. En general, se nota un aumento de dicha desviación para las instancias con menor número de trabajos y aquellas con mayor número de máquinas.

También es importante tener en cuenta que la cota inferior utilizada es una cota inferior para el FSSP y su aproximación al valor óptimo del FSSPSC ofrece una sobrevaloración de la desviación real a la solución óptima.

Tabla 2. Tiempos promedio de corrida en segundos por combinación de número de trabajos y número de máquinas.

	Número de máquinas		
Número de Trabajos	5	10	20
20	0,2156	0,4201	0,8484
50	0,7624	1,5589	3,0203
100	2,2757	4,5891	9,1901
200		16,0240	30,7916
500			169,8675

Tabla 3. Desviación porcentual promedio a la cota inferior por combinación de número de trabajos y número de máquinas

	Número de Máquinas		
Número de trabajos	5	10	20
20	96,94	130,95	209,29
50	93,21	109,52	134,45
100	97,72	87,91	109,70
200		97,55	104,96
500			100,66

Evaluación de los componentes del algoritmo

En esta sección se describen varios experimentos orientados a definir la configuración del algoritmo.

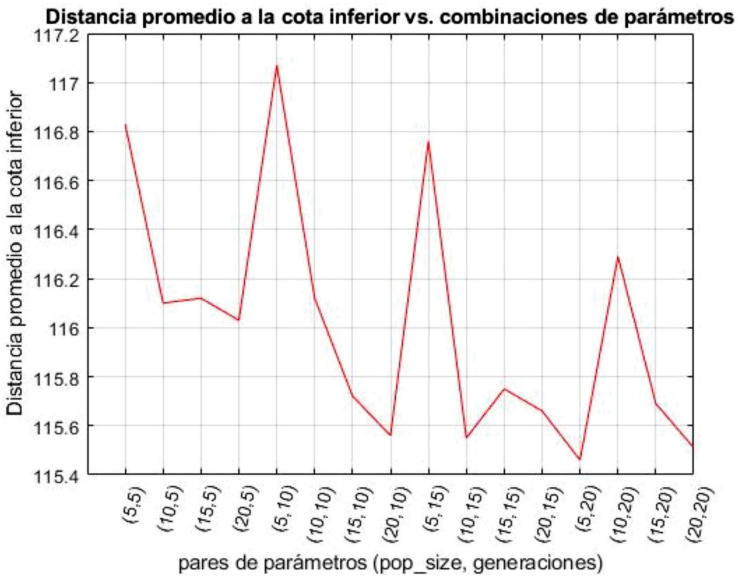

Figura 3. Comportamiento de la desviación porcentual promedio variar el tamaño de la población y el número de generaciones.

Inicialmente se utilizó el procedimiento VND como mecanismo de mutación; sin embargo, el tiempo de cómputo aumentó significativamente y el mejoramiento de la función objetivo no justificaba tal aumento. De manera alternativa, se cambió el operador de mutación por el descrito previamente y se utilizó un procedimiento de postoptimización aplicando el VND a cada individuo en la población final. Como resultado, no hubo una mejora significativa ni en el tiempo de cómputo ni en la calidad de la mejor solución encontrada. Finalmente, se decidió aplicarle el VND solo al mejor individuo obtenido por el algoritmo genético.

Para la configuración del VND se probaron tres tipos de vecindarios. Los dos primeros fueron descritos previamente en la sección Métodos y materiales. El tercer tipo de vecindario consiste en intercambiar el orden del primer trabajo en la solución actual con el de todos los demás trabajos en la solución. Después de experimentar con los tres vecindarios, se concluyó que este último vecindario tiene una probabilidad muy baja de obtener mejoras y un tiempo de cómputo elevado. Por lo tanto, se decidió utilizar solo los dos primeros vecindarios para el algoritmo híbrido.

Comparación con otros métodos

Con el fin de evaluar el desempeño del algoritmo propuesto, se compararon los resultados del metaheurístico con los de otros métodos heurísticos. Los tres algoritmos con los que se compara son:

1. *Método Constructivo:* este algoritmo, ordena los trabajos en orden no decreciente de los tiempos de procesamiento en la primera máquina, y luego aplica el algoritmo NEH [5].
2. *Búsqueda aleatoria:* Este algoritmo es una variante de los algoritmos con ruido (noising methods) en el cual se suma un valor aleatorio con distribución uniforme entre -1 y 1 (U(-1;1)) a los tiempos de procesamiento de cada trabajo en cada máquina. Luego, se ordenan los trabajos en orden no decreciente del tiempo de procesamiento promedio en todas las máquinas. La función objetivo se calcula con los datos originales para cada solución obtenida.
3. *VND:* Se utilizó el algoritmo VND utilizado con el método híbrido propuesto y el método constructivo como solución inicial.

Se debe tener en cuenta que, debido a que esta variante del FSSP no se encuentra en la literatura, no es posible comparar los resultados con otros algoritmos de la literatura.

Las Tablas 4, 5 y 6 presentan la comparación entre el algoritmo híbrido propuesto con cada uno de los tres algoritmos de comparación. En cada tabla se presenta el número de veces que cada algoritmo

Tabla 4. Comparación entre algoritmo Genético con VNS y un algoritmo Constructivo.

	Genético con VNS	**Constructivo**	**Empate**
Menor Tiempo	0	120	0
Menor distancia a z’	52	10	58

Tabla 5. Comparación entre Genético con VNS y un algoritmo de búsqueda aleatoria.

	Genético con VNS	**Búsqueda Aleatoria**	**Empate**
Menor Tiempo	0	120	0
Menor distancia z’	72	12	33

Tabla 6. Comparación entre Genético con VNS y un algoritmo de búsqueda local.

	Genético con VNS	**VNS**	**Empate**
Menor Tiempo	67	53	0
Menor distancia z’	55	2	63

obtiene el menor tiempo de cómputo y la menor desviación, así como el número de empates.

De las Tablas 4, 5 y 6 se puede concluir que, aunque tome más tiempo, la mayoría de las veces el metaheurístico híbrido propuesto obtiene soluciones mejores o iguales que los métodos de comparación. El método híbrido no solo obtiene mejores soluciones la mayoría de las veces con relación al algoritmo VND, sino que más de la mitad de las veces es más rápido. En cuanto al tiempo de cómputo, el algoritmo propuesto tiene un mejor promedio y es más estable que el *VND*.

La Tabla 7 presenta el tiempo de cómputo promedio (en segundos), la desviación estándar del tiempo de cómputo, la desviación porcentual promedio con respecto a la cota inferior y la desviación estándar de la desviación porcentual promedio, para cada uno de los cuatro algoritmos en comparación.

De la Tabla 7, como era de esperarse, el método constructivo es el más rápido, seguido del algoritmo de búsqueda local, el método híbrido y el VND. Por un lado, es claro que el procedimiento de mejora basado en VND tiene una complejidad computacional importante. Sin embargo, el algoritmo híbrido logra superar en eficiencia al algoritmo VND tradicional. Con relación a la desviación porcentual con respecto a la cota inferior, se nota que el comportamiento tiende a ser el opuesto, es decir, los algoritmos más rápidos son los que menor calidad ofrecen en la solución final. También es importante notar que no hay una gran diferencia entre el algoritmo híbrido y el VND.

CONCLUSIONES

En este artículo se introduce una nueva variante del FSSP nombrado como Flow Shop Scheduling Problem with Shift Constraints (FSSPSC). En este problema, además de las restricciones del FSSP clásico, se requiere que cada trabajo inicie en el mismo turno de trabajo en el que se inició su procesamiento. Dicha restricción es de gran relevancia en procesos en los que la calidad del producto puede verse afectada por interrupciones o cambios de personal.

Para este problema se propuso un algoritmo metaheurístico híbrido que combina un algoritmo genético con un *"variable neighborhood descent"* de manera secuencial y se evaluó su rendimiento. Este nuevo método permite obtener soluciones factibles del FSSPSC.

Este algoritmo híbrido balancea las ideas de explotación y exploración. La explotación se favorece gracias a los operadores de selección y cruce de individuos de la población, así como la aplicación del VND a la mejor solución de la población final. De otro modo, la generación aleatoria de la población inicial y el operador de mutación ayudan a diversificar las soluciones.

La configuración del algoritmo híbrido fue definida principalmente por la complejidad computacional resultante, sin descuidar significativamente la calidad de las soluciones obtenidas. Luego de una experimentación inicial, se optó por aplicar el procedimiento VND una sola vez, y utilizar solo dos vecindarios. Esto resultó beneficioso ya que el nuevo algoritmo logra mejores tiempos de cómputo que el VND con una calidad equivalente a la del VND.

Como trabajo futuro, se propone evaluar otras estrategias para mejorar las soluciones, así como la realización de nuevos experimentos orientados a evaluar el desempeño de los algoritmos. Entre las alternativas de mejoramiento, se propone

Tabla 7. Comparación entre el desempeño del algoritmo híbrido propuesto y los tres algoritmos de comparación.

	Tiempo de cómputo promedio (s)	**Desviación estándar del tiempo de cómputo**	**Desviación porcentual promedio**	**Desviación estándar de la desviación**
Híbrido	22,14	49,585	115,501	31,950
Constructivo	0,074	0,098	174,260	79,755
B. aleatoria	0,432	0,628	117,770	33,991
VND	33,30	81,620	115,006	31,940

explorar nuevas estructuras de vecindarios, y nuevas configuraciones de algoritmos híbridos.

REFERENCIAS

[1] C. Chettha, S. Kanchana, C. Chen-Fu and G. Mitsuo. "Hybrid genetic algorithms for solving reentrant flow-shop scheduling with time windows". International Journal of Industrial Engineering and Management Systems. Vol. 12 Nº 4, pp. 306-316. 2013.

[2] D. Kalyanmoy. "Multi-objective optimization using evolutionary algorithms". International Journal of John Wiley & Sons. Vol. 16. 2001.

[3] J.S. David, P.H. Christos and Y. Mihalis. "How easy is local search?". Journal of computer and system sciences. Vol. 37 Nº 1, pp. 79-100. 1988.

[4] N. Muhammad, E. Jr Emory and H. Inyong. "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem". The International Journal of Management Science. Vol. 11 Nº 1, pp. 91-95. 1983.

[5] G. Jian and C. Rong. "An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems". Scientific Research and Essays. Vol. 6 Nº 14, pp. 3094-3100. 2011.

[6] M. Rajkumar, P. Asokan, T. Page, and S. Arunachalam. "A GRASP algorithm for the integration of process planning and scheduling in a flexible job-shop". International Journal of Manufacturing Research. Vol. 5 Nº 2, pp. 230-251. 2010.

[7] P.M. Francca, J.N.D. Gupta, A.S. Mendes, P. Moscato and K.J. Veltink. "Evolutionary algorithms for scheduling a flowshop manufacturing cell with sequence dependent family setups". Computers & Industrial Engineering. Vol. 48 Nº 3, pp. 491-506. 2005.

[8] K. Mesghouni, S. Hammadi and P. Borne. "Evolutionary algorithms for job-shop scheduling". International Journal of Applied Mathematics and Computer Science. Vol. 14 Nº 1, pp. 91-104. 2004.

[9] R. Ruiz and T. Stutzle. "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem". European journal of operational research. Vol. 177 Nº 3, pp. 2033-2049. 2007.

[10] Q. Pan, M.F. Tasgetiren and Y. Liang. "A discrete differential evolution algorithm for the permutation flowshop scheduling problem". Computers & Industrial Engineering. Vol. 55 Nº 4, pp. 795-816. 2008.

[11] E. Salazar and N. Pavón. "Aplicación de un algoritmo ACO al problema de taller de flujo de permutación con tiempos de preparación dependientes de la secuencia y minimización de makespan". Ingeniare Revista chilena de ingeniería. Vol. 19 Nº 2, pp. 253-264. 2011

[12] A. Hertz and M. Widmer. "An improved tabu search approach for solving the job shop scheduling problem with tooling constraints". Discrete applied mathematics. Vol. 65, pp. 319-345. 1996.

[13] C. Rajendran and H. Ziegler. "Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs". European Journal of Operational Research. Vol. 155, pp. 426-438. 2004.

[14] X. Delorme, X. Gandibleux and J. Rodriguez. "GRASP for set packing problems". European Journal of Operational Research. Vol. 153, pp. 564-580. 2004.

[15] X. Dong, H. Huang and P. Chen. "An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion". Computers & Operations Research. Vol. 36, pp. 1664-1669. 2009.

[16] C.R. Reeves and T. Yamada. "Genetic algorithms, path relinking and the flowshop sequencing problem. Evolutionary computation. Vol. 6, pp. 45-60. 1998.

[17] T. Murata, H. Ishibuchi and H. Tanaka. "Genetic algorithms for flowshop scheduling problems". Computers & Industrial Engineering. Vol. 30 Nº 4, pp. 1061-1071. 1996.

[18] E. Taillard. "Benchmarks for basic scheduling problems". European Journal of Operational Research. Vol. 64 Nº 2, pp. 278-285. 1993.