Metaheurísticas Basadas en Trayectoria para Resolver una Variante del Problema de Flowshop

Gabriela Minetti[1]- Carolina Salto[2]
[1,2]Facultad de Ingeniería - Facultad de Ingeniería, CONICET
Universidad Nacional de La Pampa
[1]minettig@ing.unlpam.edu.ar - [2]saltoc@ing.unlpam.edu.ar

Resumen: *El problema de flowshop flexible híbrido con secuencias dependientes del tiempo de puesta a punto es un problema que se puede encontrar en muchos ambientes industriales En este trabajo proponemos algoritmos metaheurísticos basados en trayectoria para resolver dicha variante. Las dos primeras propuestas son algoritmos de enfriamiento simulado que utilizan diferentes operadores de movimiento: intercambio e inserción. La tercera y última propuesta modifica un algoritmo de búsqueda local iterada propuesto en la literatura al reemplazar el operador de inserción por el de intercambio. Los experimentos numéricos comparan el rendimiento de las distintas propuestas, utilizando un amplio conjunto de datos de prueba de uso actual en la literatura. Los resultados muestran que las variantes algorítmicas que aplican el operador de intercambio son más eficientes que las que usan el de inserción para solucionar el problema en estudio.*

Palabras Claves: *metaheurísticas, enfriamiento simulado, búsqueda local iterada, flowshop flexible híbrido.*

Abstract: *The hybrid flexible flowshop problem with sequence dependent setup times is a problem found in many industrial environments. We propose trajectory-based metaheuristic algorithms to solve this problem. The first two proposals are simulated annealing using different movement operators: insertion and exchange. The third and last proposal modifies an iterated local search algorithm proposed in the literature by changing the insertion operator for the exchange one. Numerical experiments compare the performance of the different proposals using a comprehensive benchmark from the literature. The results show that algorithmic approaches applying the exchange operator are more efficient than the ones using the insertion operator to solve the variant of the flowshop problem.*

Keywords: *metaheuristics, simulated annealing, iterated local search, hybrid flexible flowshop.*

INTRODUCCIÓN

En el ámbito industrial se presentan con frecuencia problemas de planificación de la producción. Se pueden definir como la asignación de tareas a recursos de producción disponibles y luego secuenciarlas en cada recurso en el tiempo de una manera eficiente. Tales problemas son conocidos en su forma más simple como flowshop, el cual es difícil de resolver debido a su naturaleza combinatoria [3], [7].

En este trabajo nos centramos en una variante del flowshop donde existen varias etapas en la producción con múltiples máquinas (idénticas o no) por etapa, además cabe la posibilidad de que algún trabajo no necesite ser procesado en algunas de las etapas. Esta variante se conoce como problema de flowshop flexible híbrido (hybrid flexible flowshop problem, HFFSP). Además, en varias industrias (farmacéutica, metalúrgica, automotriz, entre otras) se consideran tiempos de puesta a punto (setup) de las máquinas entre dos trabajos diferentes, los cuales aportan mayor dificultad a los problemas de planificación de la producción. Estos tiempos de

setup pueden o no depender de la secuencia. Considerando estas restricciones, el problema considerado en este trabajo es el HSSFP con tiempos de setup dependientes de la secuencia (sequence dependent setup time, SDTS/FSSFP), cuyo objetivo es minimizar el tiempo de finalización.

Como veremos, el problema SDST/HFFS no ha sido ampliamente estudiado en la literatura. Sin embargo, se han propuesto algunas heurísticas [10] y metaheurísticas tal como algoritmos genéticos [11], [20], un sistema inmune [23], un algoritmo de búsqueda local iterada [16], entre otros.

En este trabajo, extensión del artículo [14], proponemos algoritmos metaheurísticos basados en trayectoria para resolver el problema SDST/HFFS. En una primera etapa se presentan dos variantes del algoritmo de enfriamiento simulado, donde se analizarán dos operadores de movimiento para generar el vecindario del estado actual, a fin de determinar cuál de los dos permite una mejor exploración del espacio de soluciones del SDST/HFFS ya que tiene un fuerte impacto en la efectividad del método. Luego, se formula una mejora al algoritmo de búsqueda local iterada propuesto por Naderi et al. [16]. Con el fin de evaluar nuestro aporte, estas propuestas algorítmicas se compararán con algoritmos que representen el estado del arte para el SDST/HFFS.

El artículo se organiza de la siguiente manera. En la Sección 2 se describe el problema y se detallan los algoritmos que lo tratan en la literatura. La Sección 3 explica los algoritmos propuestos en este artículo. La Sección 4 especifica la parametrización utilizada en la experimentación mientras que en la Sección 5 se presentan y analizan los resultados de la experimentación realizada. Finalmente, la Sección6 resume nuestras conclusiones y esboza el trabajo futuro.

DESCRIPCIÓN DEL PROBLEMA

En esta sección se realizará una descripción detallada del problema en cuestión: flowshop flexible híbrido con tiempos de puesta a punto (setup) (SDST/HFFS). La sección concluye con un breve estado del arte sobre el mismo.

El problema HFFS se caracteriza por un conjunto de N trabajos, $N=\{1,\cdots,n\}$, disponibles para su procesamiento en tiempo cero en una serie de M etapas, $M= \{1,\cdots,m\}$. En cada etapa $i,\in M$, se tiene un conjunto de $M_i=\{1,\cdots,m_i\}$ máquinas paralelas idénticas. Cada máquina en cada etapa puede procesar cualquier trabajo.

Cada trabajo tiene que ser procesado por sólo una de las Mi máquinas paralelas idénticas. Sin embargo, algunos trabajos pueden saltar algunas etapas. F_j indica el conjunto de etapas que el trabajo j ($j\in N$) tiene que visitar ($1 \leq |F_j| \leq m$). Sólo se permite saltar etapas pero no es posible alterar el orden de visita de las mismas. El tiempo de procesamiento del trabajo j en la etapa i se indica por p_{ij} . Finalmente, S_{ijk} es el tiempo de puesta a punto entre el trabajo j y el k ($k\in N$) en la etapa i. El tiempo de finalización del trabajo j se indica por C_j , y es el tiempo de finalización del trabajo en la última etapa visitada. El criterio de optimización es minimizar del tiempo de terminación máximo o makespan, el cual se calcula como $C_{max} = \max_{j\in N} C_j$.

Un caso especial del SDST/HFFS es el problema de flowshop con múltiples procesadores con sólo dos etapas (m=2). Gupta [4] demostró que este último es NP-duro, lo cual sugiere que el SDST/HFFS pertenece a la misma clase de problemas [9].

Los primeros pasos en la resolución del SDST/HFFS fue el trabajo de Kurz y Askin [10] al introducir reglas de despacho basadas en métodos greedy, heurística de múltiple inserción y una adaptación de la regla de Johnson. Los mismos autores propusieron un algoritmo genético (Genetic Algorithm,

GA) con representación random keys (RKGA) [11] cuyos resultados mejoraron los publicados en [10]. Zandieh et al. [23] propusieron un sistema inmune artificial (artificial inmune system, IAS) utilizando una representación real de los individuos y el operador de cruce tradicional order crossover (OX). Los resultados muestran que el IAS mejora los resultados del RKGA. Mirsanei et al. [15] propusieron un algoritmo de enfriamiento simulado (Simulated Anneling, SA) que utiliza una combinación de dos operadores de movimiento (inversión e intercambio), superando los resultados del RKGA [11] y del IAS [23]. Naderi et al. [16] presentaron una regla heurística de despacho dinámica y un búsqueda local iterada (iterated local search, ILS). Estos algoritmos se compararon con otros siete algoritmos existentes, siendo la metaheurística búsqueda local iterada (Iterated Local Search, ILS) la que mejores resultados obtuvo. Choong et al. [2] propusieron un algoritmo híbrido basado en ejecutar la técnica de optimización por enjambre de partículas (Particle Swarm Optimization, PSO) y luego mejorar el resultado obtenido con SA o búsqueda tabú. La mejor combinación híbrida resultó ser PSO con SA, pero sin mejorar el estado del arte en la resolución del problema. Un trabajo reciente es el de [20] que plantearon un GA y analizaron su comportamiento utilizando varios operadores de cruce específicos del problema. Los autores muestran que su GA mejora al ILS [16]. Este GA no se incluirá en la comparación con los algoritmos propuestos en este trabajo, debido a la falta de detalle en su descripción al momento de replicar su funcionamiento.

METAHEURÍSTICAS PARA SDST/HFFSP

En esta sección describiremos los algoritmos basados en trayectoria, SA e ILS, propuestos para resolver el problema SDST/HFFS. Para ello es necesario explicar cómo determinar la secuencia de trabajos al comienzo de cada etapa y la forma de asignar trabajos a cada máquina en cada etapa.

En el problema de flowshop tradicional, sólo es necesaria una permutación de trabajos por cada máquina, al menos para los criterios basados en el tiempo de finalización. En cambio en el problema HFFS, mantener una única permutación de trabajos en todas las etapas resulta en una planificación poco eficiente. Además, en el flowshop tradicional no es necesario asignar trabajos a las máquinas dado que existe una sola máquina por etapa. En cambio, HFFS con más de una máquina por etapa puede ser visto como una serie de problemas de máquinas paralelas.

Esto nos lleva a adoptar una regla de despacho que contemple estas dos características, por ende elegimos la regla heurística modificada de despacho dinámico (Modified dynamic dispatching rule, MDDR), propuesta por Naderi et al. en [16]. MDDR se trata de un método muy rápido y efectivo que no separa las decisiones de secuenciamiento de trabajos y asignación de máquinas. Comienza con la etapa 1, considerando todos los trabajos que visitan esta primera etapa. Para estos trabajos se calcula el mínimo tiempo de finalización más temprano (earliest completion time, ECT) para todas las máquinas paralelas en esta etapa. El trabajo con el mínimo ECT se planifica y asigna a la máquina que produzca el valor mínimo de ECT. Este proceso se repite hasta que se asignen todos los trabajos en la etapa uno. Luego, el proceso continúa con la segunda etapa, pero esta vez considerando los tiempos de finalización de los trabajos en la etapa anterior. Esto se repite en las restantes etapas. Esta regla de despacho se considera dinámica ya que el valor de ECT puede variar luego de planificar cada trabajo.

Para una mejor comprensión de los algoritmos propuestos, es necesario explicar previamente los métodos de codificación de soluciones y de generación de la solución inicial utilizados. Si bien durante la última década se han empleado muchos métodos

de codificación, hemos adoptado uno de los más utilizados denominado representación basada en trabajos (job- based representation, JBR) [5]. En JBR, la secuencia de trabajos en la etapa uno se representa por una simple permutación de trabajos que indica el orden en el cual se ejecutan los mismos. En tanto que, la secuencia para las siguientes etapas se determina usando la regla de despacho MDDR.

Habitualmente, la generación de las soluciones con las cuales una metaheurística inicia su búsqueda es aleatoria. Sin embargo, considerar "buenas" soluciones iniciales en los primeros pasos del método ayuda a aprovechar las regiones prometedoras del espacio de búsqueda [13], [19], [22]. Para ello empleamos una adaptación de la bien conocida técnica NEH [17], denominada NEHH, ya que se trata de la heurística con mejor desempeño en el tratamiento de HFFS [18].

ENFRIAMIENTO SIMULADO

Enfriamiento simulado (Simulated Annealing, SA) es un método Monte-Carlo simple y de propósito general desarrollado para optimización combinatoria en la década de 1980 por Kirkpatrick et al. en [8]. SA es un algoritmo metaheurístico basado en una analogía con los sistemas físicos de enfriamiento. Este proceso térmico consiste en la obtención de estados de baja energía de un sólido en un baño de calor.

En los primeros pasos (a alta temperatura), SA acepta soluciones con mayores costos en virtud de una cierta probabilidad, con el fin de explorar el espacio de búsqueda y para escapar de óptimos locales. Durante el proceso de enfriamiento esta probabilidad disminuye de acuerdo a la temperatura de enfriamiento, la intensificación de la búsqueda y la reducción de la exploración con el fin de explotar una zona restringida de un espacio de búsqueda. En los últimos pasos la exploración es nula (o casi nula) y la explotación es total (o casi total). SA evoluciona a través de una secuencia de transiciones entre estados; las cuales son generadas por probabilidades de transición. En consecuencia, SA puede ser matemáticamente modelado por cadenas de Markov, donde se genera una secuencia de cadenas usando una probabilidad de transición cuyo cálculo involucra a la temperatura actual.

En otras palabras, SA inicia con una solución S0 y a continuación genera una solución vecina S1 mediante la aplicación de un operador de movimiento. Si S1 mejora el valor de la función objetivo se acepta con probabilidad 1; en caso contrario se calcula la probabilidad para aceptarla mediante el criterio de Boltzman ($\exp((E1 - E0)/T)$) que involucra el parámetro correspondiente a la temperatura actual T y a las evaluaciones de S0 y S1, E1 y E0 respectivamente. Este parámetro se actualiza cuando la longitud de la cadena de Markov es nula siguiendo algún esquema de decaimiento.

SA es una metaheurística fácilmente adaptable a problemas de planificación de la producción [1], [6], ya que es posible adecuar su diseño a una representación por permutación. Esto significa, por un lado, poder usar como función de evaluación la regla de despacho MDDR, y por otro lado, utilizar operadores especialmente diseñados y ampliamente usados, para este tipo de representación tales como el operador de intercambio y el de inserción. Sin embargo, la facilidad de adaptación al problema no es la única razón por la que se elige SA, sino que también se lo escoge por su capacidad de explorar el espacio de búsqueda en etapas tempranas localizando las regiones más prometedoras para luego explotarlas. De esta forma, surgen SAint y SAins, dos variantes basadas en SA que utilizan el operador de intercambio y el de inserción, respectivamente, para generar las soluciones vecinas. En el algoritmo 1, se muestra un pseudo-código común a SAint y a SAins.

Algoritmo 1 Pseudo-código de Saint y SAins:

```
k = 0;
inicializar T ; {inicializar la temperatura, T }
NEHH(S0 ); {generar la solución inicial, S0 , usando NEHH [17]}
evaluar S0 en E0 ;
repeat
    repeat
        k = k + 1;
        generar S1 desde S0 ; {generar la solución vecina, S1 , usando los
operadores de nterambio
        o inserción}
        evaluar S1 en E1 ;
        {si la nueva solución es mejor que la actual, S1 es aceptada}
        if (E1 − E0 ) ≤ 0 then
          S0 = S1 ;
          E0 = E1 ;
          {si la nueva solución es peor que la actual, S1 es aceptada bajo
la probabilidad:
          exp((E1 − E0 )/T )}
        else
          if exp((E1 − E0 )/T ) > random[0, 1) then
            S0 = S1 ;
            E0 = E1 ;
          end if
        end if
    until (k mod Long. Cadena de Markov) == 0
    actualizar T ;
until condición de corte sea satisfecha
return S0 ;
```

BÚSQUEDA LOCAL ITERADA

La búsqueda local iterada (Iterated Local Search, ILS) [12], [21] es una metaheurística basada en un concepto simple pero muy efectivo. En cada iteración, la solución actual es perturbada y a esta nueva solución se le aplica un método de búsqueda local para mejorarla. Este nuevo mínimo local obtenido por el método de mejora puede ser aceptado como nueva solución actual si pasa un test de aceptación. La importancia del proceso de perturbación es obvia: si es demasiado pequeño puede que el algoritmo no sea capaz de escapar del mínimo local; por otro lado, si es demasiado grande, la perturbación puede hacer que el algoritmo sea como un método de búsqueda local con un reinicio aleatorio. Por lo tanto, el método de perturbación debe generar una nueva solución que sirva como inicio a la búsqueda local, pero que no debe estar muy lejos de la actual para que no sea una solución aleatoria. El criterio de aceptación actúa como contra balance, ya que filtra la aceptación de nuevas soluciones dependiendo de la historia de búsqueda y de las características del nuevo mínimo local.

En [16], Naderi et al. proponen una ILS para SDST/HFFSP que resulta ser efectiva y eficiente y, que de ahora en adelante denominaremos ILSins. Este algoritmo comienza la búsqueda con una solución inicial generada por NEHH, luego una búsqueda local (Local Search, LS) basada en el operador de inserción se aplica a la solución candidata x. Esta búsqueda se detiene cuando se encuentra una primera mejora. Cuando no se encuentra ninguna mejora, se incrementa un contador. Este contador vuelve a cero cada vez que la búsqueda local mejora la solución actual. Si este contador supera un determinado umbral, se aplica un mecanismo de perturbación (Giant Leap, GL) basado en una colección de movimientos de inserción, con el fin de redireccionar la búsqueda hacia regiones más prometedoras del espacio de soluciones.

Con el objetivo de incrementar la calidad de las soluciones, en este trabajo se propone una variante de la metaheurística propuesta por Naderi, denominada ILSint. Dicha variante consiste en modificar el proceso de búsqueda local. Esta nueva LS se basa en el operador de intercambio y procede de la siguiente forma: el trabajo en la primera posición de x, denominado x1 se intercambia por el trabajo ubicado en una posición diferente elegida al azar en la permutación. Si esta nueva permutación x' resulta con un mejor makespan, la solución actual x es reemplazada por x' y la búsqueda local termina. De lo contrario, el procedimiento continúa con x2 . La búsqueda itera en la mayoría de los trabajos, sin repetición. El fundamento de este procedimiento es realizar una búsqueda local muy rápida. En el algoritmo 2 se describe un pseudo-código común a ILSint y a ILSins, y en el algoritmo 3 se detalla el esquema general de la búsqueda local basada en intercambio.

Algoritmo 2 Pseudo-código de ILSins e ILSint

```
contador = 0;
Inicializar x; {ILSins e ILSint usan NEHH durante la inicialización}
xbest = x;
while condición de corte no sea satisfecha do
    x' = LS(x); {LS basada en inserción o intercambio según corresponda}
    if f (x' ) < f (x) then
      x = x';
    if f (x'') < f (xbest ) then
      xbest = x'' ;
    end if
else
    contador = contador + 1;
    {Aplicación del operador de perturbación, GL}
    if contador > umbral then
      contador = 0;
      for r=1 to nu_move do
        s~(r) = GL(x);
      end for
      x = s~best ;
    end if
  end if
end while
return x;
```

Algoritmo 3 Esquema general de la LS propuesta para ILSint

```
i = 0;
while i ≤ n do
    x' = la secuencia producida al intercambiar xi con el trabajo ubicado en
una posición
    diferente elegida al azar;
    if Cmax (x' ) < Cmax (x) then
      x = x' ;
      break;
    end if
    i = i + 1;
end while
return x ;
```

DISEÑO EXPERIMENTAL

En esta sección se muestran los valores paramétricos utilizados para evaluar el rendimiento de las metaheurísticas basadas en trayectoria descriptas en la Sección 3 para resolver el problema SDTS/FSSFP. Las variantes propuestas son:

SAint: SA con operador de intercambio para generar la solución vecina.

SAins: SA con operador de inserción para generar la solución vecina.

ILSint: ILS con una LS basada en el operador de intercambio.

ILSins: ILS con una LS basada en el operador de inserción (originalmente propuesto en [15]).

Los parámetros específicos del SA se fijaron en los siguientes valores. La temperatura inicia con un valor de 0.99 y el decaimiento se planifica usando el enfriamiento proporcional. En tanto que el número de iteraciones entre dos cambios consecutivos de temperatura (longitud de la cadena de Markov) se fija en 30. Este valor surgió de una experimentación previa realizada tomando valores de iteraciones en el conjunto 10, 30 y 60, resultando 30 el número de iteraciones más adecuado para el proceso, con el fin de alcanzar el cuasi equilibrio en cada nivel de temperatura.

Los valores de los parámetros para las variantes de ILS se describen a continuación. La cantidad máxima de iteraciones sin observar mejoras entre perturbaciones (umbral) se fija en 30. El operador de perturbación GL utiliza la operación de inserción para generar 15 (nu_move) soluciones perturbadas, siendo 2 la cantidad de trabajos reubicados. Estos parámetros son los adoptados en [16].

Todas las variantes algorítmicas planteadas utilizan la heurística NEHH para generar la solución inicial. A fin de que la comparación entre las distintas metaheurísticas resulte confiable, el criterio de parada fue configurado en $n^2 \times m \times 1{,}5$ms de tiempo de CPU. Este criterio es el adoptado en varios trabajos [16], [20]. El cuadro I muestra un resumen de los parámetros utilizados en este trabajo.

El conjunto de prueba está formado por 960 instancias (disponibles en http:\\soa.iti.es). Las instancias resultan de la combinación de varios valores de n y m, donde n={20, 50, 80, 120} y m={2, 4, 8}. Los tiempos de procesamiento están generados desde una distribución uniforme [1,99]. Los tiempos de puesta a punto pertenecen a cuatro distribuciones distintas ([1,25], [1,50], [1,99] y [1,125]) que permiten analizar los efectos que tiene el tiempo de setup en los resultados; se corresponden con un 25%, 50%, 100% y 125% del tiempo de procesamiento, respectivamente. Se consideran grupos de instancias con dos máquinas paralelas en cada etapa y otros grupos donde la cantidad de máquinas por etapas se obtiene de una distribución uniforme

en el rango [1,4]. La probabilidad de que un trabajo salte una etapa es de 0.1 y 0.4. En el cuadro II se presenta un resumen de estas instancias.

CUADRO 1
VALORES PARAMÉTRICOS USADOS POR LAS VARIANTES DE SA Y DE ILS

	PARÁMETRO	VALOR
SA	LONGITUD DE LA CADENA DE MARKOV	30
	TEMPERATURA INICIAL	0.99
ILS	Nº DE ITERACIONES SIN MEJORA (UMBRAL)	15
	Nº DE PERTURBACIONES (UN_MOVE)	30
ILS e SA	REPRESENTACIÓN DE LA SOLUCIÓN	PERMUTACIONES
	MÉTODO PARA GENERAR LA SOLUCIÓN INICIAL	NEHH
	CONDICIÓN DE TERMINACIÓN	$n^2 \times m \times 1{,}5ms$

CUADRO 2
VALORES PARAMÉTRICOS USADOS POR LAS VARIANTES DE SA Y DE ILS

PARÁMETRO	VALOR
n	20, 50, 80, 120
m	2, 4, 8
m_i	CONSTANTE: 2 VARIABLE: U(1,4)
P_{ij}	U(1,99)
S_{ijk}	U(1,25) U(1,50) U(1,100) U1,125)
S_p	0.1, 0.4

Los algoritmos fueron implementados en C++. Para la experimentación se utilizó un cluster de PCs con procesador Intel Core i7 a 3.90 GHz y 4GB de RAM. Cada instancia fue ejecutada 30 veces para cada configuración algorítmica. Para comparar los distintos algoritmos se utiliza la desviación porcentual relativa (DPR) de las mejores soluciones conocidas, esta métrica se describe en la Ecuación 1, donde Algsol es el mínimo Cmax obtenido por un determinado algoritmo para una instancia del problema en las 30 ejecuciones y minsol es el mínimo Cmax de los algoritmos bajo comparación para una determinada instancia. Obviamente, son deseables valores pequeños de DPR para un algoritmo, ya que indican una desviación porcentual de sus soluciones respecto del mínimo.

$$DPR = \frac{Alg_{sol} - min_{sol}}{mimin_{sol}n} \times 100 \qquad (1)$$

La experimentación realizada ha sido extensa: las cuatro variantes algorítmicas en estudio se evaluaron con cada una de las instancias, en total suman 3840 combinaciones (4 algoritmos x 960 instancias). Debido a la naturaleza estocástica de los algoritmos, cada una de ellas se ejecutan 30 veces para obtener una muestra confiable. En consecuencia se llevaron a cabo 30 x 3840 ejecuciones, totalizando 115200 ejecuciones.

Hemos realizado un análisis estadístico de los resultados a fin de determinar si existen diferencias significativas entre los algoritmos comparados. Como paso previo, se determina si los datos siguen una distribución normal al aplicar la prueba de Shapiro-Wilks y homogeneidad de varianzas (homocedasticidad). En el caso de que los datos resulten con una distribución normal, se realiza la prueba de ANOVA. En caso contrario, se usa el test no paramétrico Kruskal Wallis. El valor de α se fija en 0.01, para indicar un nivel de confianza del 99 % en los resultados.

ANÁLISIS DE LOS RESULTADOS

Esta sección está dedicada al análisis de los resultados obtenidos en la experimentación llevada a cabo sobre los variantes de SA e ILS mencionadas en el apartado anterior.

En las siguientes subsecciones, se medirán en primer instancia parámetros básicos tales como DPR (ver ecuación 1), la cantidad de veces que un algoritmo encuentra la mejor solución (#min) y la distancia relativa (DR) entre la solución inicial y la final, cuyos objetivos son ofrecer un estudio de las variantes algorítmicas propuestas desde el punto de vista de la calidad de las soluciones encontradas. Por último, se realiza un análisis del esfuerzo numérico y tiempo empleado por los algoritmos para hallar una solución, esto permite estudiar la velocidad de convergencia de cada algoritmo para encontrar la mejor solución.

ANÁLISIS DE CALIDAD

Comenzaremos el análisis con la comparación del rendimiento de los algoritmos SA propuestos con dos métodos para generar el vecindario. El cuadro III muestra los valores promedios de DPR (columnas 2 y 3) y de #min (columnas 4 y 5) para cada grupo de instancias discriminados para las variantes de SA propuestas. Los resultados de los experimentos están promediados para cada combinación de n y m. A fin de verificar la validez estadística de los resultados, la última columna de este cuadro muestra los valores p obtenidos del estudio estadístico realizado. La última fila del cuadro presenta valores promedios de las dos métricas y se resaltan en negritas los mejores valores para cada una.

Como se puede observar en el cuadro III, los valores promedios de DPR para SAint son menores que los de SAins para todas las instancias, indicando que el primer algoritmo encuentra soluciones con valores de makespan menores que el segundo. Además, los valores de #min de SAint superan, en promedio, en más de cuatro veces el valor correspondiente a SAins. Los valores p son menores que α, entonces con un 99 % de nivel de confianza se puede afirmar que hay diferencias significativas entre la calidad de soluciones obtenidas por SAint y SAins. De esta manera, se puede inferir la influencia del operador de movimiento en la calidad final de los resultados. Es decir que, la operación de intercambio le permite al SA explotar con mayor eficacia el espacio de búsqueda que en el caso de usar el operador de inserción.

CUADRO III
VALORES PROMEDIOS DE DPR Y #MIN OBTENIDOS POR CADA SA EN CADA GRUPO DE INSTANCIAS

Inst.	DPR		#min		ANÁLISIS DE VARIANZA (P-VALUE)
	SAint	SAins	SAint	SAins	
20x2	**1,85**	6,36	**64**	16	1,4280E-09
20x4	**0,75**	5,36	**66**	14	1,9340E-15
20x8	**0,68**	4,73	**60**	20	5,6650E-12
50x2	**0,79**	6,86	**69**	11	2,2000E-16
50x4	**1,57**	5,23	**66**	14	2,1700E-15
50x8	**0,13**	5,25	**74**	06	2,2000E-16
80x2	**2,19**	5,75	**67**	13	4,0280E-15
80x4	**0,72**	4,11	**68**	12	2,2000E-16
80x8	**0,17**	3,70	**70**	10	2,2000E-16
120x2	**0,79**	6,02	**75**	05	2,2000E-16
120x4	**1,01**	4,14	**69**	11	2,2000E-16
120x8	**0,25**	3,10	**68**	12	2,2000E-16
PROMEDIO	**0,91**	5,05	**68**	12	

Dado que SAint resultó ser el algoritmo basado en SA que encuentra soluciones de mayor calidad, a continuación se comparan sus resultados con los obtenidos por las variantes ILS. En el cuadro IV se muestran los valores promedio de DPR y de #min obtenidos por estos tres algoritmos. De dicha comparación surge que los algoritmos basados en ILS obtienen soluciones con mejores makespan finales que SAint, debido a que los DPR son notablemente menores a los calculados para este último. Como muestra la última columna, el análisis de varianza avala las diferencias antes mencionadas, resultando ILSint el algoritmo que obtiene los valores mínimos de Cmax .

Al contrastar las dos versiones de ILS observamos que ILSint presenta valores de DPR inferiores a ILSins para los tres primeros grupos de instancias (n∈{20, 50, 80}), mientras que para las instancias con n = 120 los valores son similares, obteniendo ILSins una ventaja mínima. Para confirmar estas últimas observaciones hemos realizado pruebas estadísticas considerando sólo las dos versiones de ILS, las cuales indican que las diferencias entre ILSint y ILSins son significativas (valores p cercanos a 0) para n∈{20, 50, 80}, mientras que para n = 120 los resultados son estadísticamente similares (valores p mayores al nivel de significancia α).

Analizando la cantidad de valores de #min de cada algoritmo por grupo de instancias, ILSint tiene valores considerablemente mayores (cercanos al total de instancias dentro de cada grupo) para los

CUADRO IV
VALORES PROMEDIOS DE DPR Y #MIN OBTENIDOS POR SAINT, ILSINT, ILSINS EN CADA GRUPO DE INSTANCIAS

Inst.	DPR			#min			ANÁLISIS DE VARIANZA (P-VALUE)
	SAint	ILSint	ILSins	SAint	ILSint	ILSins	
20x2	13,93	**0,00**	1,22	05	**74**	01	2,20E-16
20x4	8,24	**0,00**	1,01	04	**75**	01	2,20E-16
20x8	6,68	**0,06**	0,65	03	**68**	09	2,20E-16
50x2	6,30	**0,07**	2,30	09	**70**	01	2,20E-16
50x4	8,07	**0,05**	1,83	05	**73**	02	2,20E-16
50x8	5,75	**0,07**	1,64	02	**76**	02	2,20E-16
80x2	7,75	**0,17**	2,15	14	**62**	04	2,20E-16
80x4	6,32	**0,04**	1,69	07	**70**	03	2,20E-16
80x8	5,63	**0,03**	1,57	01	**77**	02	2,20E-16
120x2	4,42	**0,89**	0,88	33	**27**	20	6,24E-01
120x4	5,76	**0,71**	0,62	20	**27**	33	7,08E-10
120x8	3,57	**0,49**	0,45	12	**26**	42	2,20E-16
PROMEDIO	6,87	**0,21**	1,33	9,58	**60,42**	10	

grupos de instancias con n∈{20, 50, 80}, seguido por SAint y ILSins. Para las instancias con n = 120, los valores de #min son muy similares entre los distintos algoritmos.

Del análisis se desprende que Saint e ILSint fueron los mejores algoritmos dentro de cada tipo de metaheurística. Esto da soporte a la idea de que el operador de intercambio es el más adecuado para el problema en estudio. La razón de esto es que el operador de intercambio permite conservar más bloques sin modificaciones dentro de una solución que en el caso del operador de inserción. Esto sucede porque la inserción de un trabajo en una posición aleatoria desplaza todos los trabajos a partir de tal posición, ocasionando una disrupción muy importante en las soluciones y saltos aleatorios en el espacio de búsqueda con el consecuente deterioro de la calidad.

El cuadro V muestra las distancias relativas (DR) entre la calidad de la solución inicial y de la final para cada metaheurística en cada grupo de instancias. Un mayor valor de DR indica que el algoritmo ha logrado una importante mejora de la solución. Vale indicar que todos los algoritmos utilizaron la misma estrategia para generar las soluciones iniciales (NEHH). ILSint es el algoritmo que logra mayores valores de DR para todas las instancias, corroborando que ILSint es el que logra los mejores valores de makespan.

En resumen, los algoritmos basados en ILS presentan el mejor comportamiento en cuanto a la calidad de las soluciones, especialmente cuando se utiliza el operador de intercambio como operador de búsqueda local. Es decir, que ILSint es el que obtiene los valores mínimos de makespan.

CUADRO V
MEDIANAS DE LOS VALORES DE DR PARA SAINT, SAINS, ILSINT, ILSINS EN CADA GRUPO DE INSTANCIAS

Inst.	SAint	SAins	ILSint	ILSins
20x2	0,22	0,17	**0,26**	0,23
20x4	0,19	0,15	**0,25**	0,22
20x8	0,18	0,14	**0,21**	0,21
50x2	0,22	0,16	**0,24**	0,22
50x4	0,18	0,14	**0,21**	0,21
50x8	0,18	0,14	**0,20**	0,18
80x2	0,20	0,16	**0,22**	0,19
80x4	0,16	0,13	**0,19**	0,18
80x8	0,15	0,12	**0,18**	0,17
120x2	0,18	0,15	**0,19**	0,18
120x4	0,16	0,14	**0,17**	0,17
120x8	0,13	0,11	**0,15**	0,15

ANÁLISIS DE DESEMPEÑO

El análisis de desempeño se lleva a cabo considerando el esfuerzo numérico realizado y el tiempo empleado por SAint, SAins, ILSint e ILSins. El esfuerzo numérico se mide teniendo en cuenta el número promedio de evaluaciones realizadas para que cada algoritmo halle la mejor solución, y por el promedio del total de evaluaciones realizadas durante el proceso completo de búsqueda. Finalmente, se analiza el tiempo promedio empleado por cada opción algorítmica para encontrar la mejor solución, el cual es medido en segundos. Cabe aclarar que

no es necesario comparar el tiempo total dedicado por cada algoritmo, dado que se estableció como condición de corte el mismo tiempo de ejecución variando de acuerdo al tamaño de las instancias.

En primer lugar, procederemos con el estudio del esfuerzo para localizar la mejor solución. Es decir, la cantidad de evaluaciones promedio de la función objetivo necesarias para alcanzar las soluciones con menor makespan con cada uno de los algoritmos. El cuadro VI muestra los valores obtenidos para esta métrica. Se observa que las variantes de ILS tienden a necesitar una menor cantidad de evaluaciones que las variantes de SA. y el otro, el conformado por las variantes de ILS. Este último reduce aproximadamente un 50 % la cantidad de evaluaciones necesarias para alcanzar las mejores soluciones cuando se lo compara con el desempeño de SA. Siendo ILSint el algoritmo que necesita en promedio el menor esfuerzo para alcanzar las mejores soluciones en 8 de los 12 grupos de instancias.

Continuamos el estudio del esfuerzo numérico teniendo en cuenta también las evaluaciones realizadas durante todo el proceso se búsqueda. Como se muestra en la figura 2, las cuatro técnicas algorítmicas realizan un número total de evaluaciones estadísticamente similar. Si, además, consideramos que el tiempo de ejecución total es igual para todas ellas, deducimos que el esfuerzo computacional realizado en cada ciclo de la búsqueda es similar. La razón de esto es que todas trabajan con una única solución por iteración y utilizan operadores de movimiento con características similares. En tanto que, el proceso de perturbación en ILS es compensado por el número de iteraciones (longitud de la cadena de Markov) que realiza SA entre dos cambios consecutivos de temperatura. Esto, también, nos lleva a inferir que el esfuerzo computacional realizado por los operadores de intercambio y de inserción no presentan diferencias importantes.

CUADRO VI

NÚMERO PROMEDIO DE EVALUACIONES REALIZADAS POR CADA ALGORITMO PARA ENCONTRAR LA MEJOR SOLUCIÓN EN CADA GRUPO DE INSTANCIAS

Inst.	ALGORITMOS				ANÁLISIS DE VARIANZA (P-VALUE)
	SAint	SAins	ILSint	ILSins	
20x2	714960	747799	**284359**	361750	2,20E-16
20x4	562998	587730	**277894**	281207	2,20E-16
20x8	490962	503395	**227554**	253201	2,20E-16
50x2	681551	718550	**339083**	375352	2,20E-16
50x4	672317	692112	**314303**	373033	2,20E-16
50x8	692937	704472	**373581**	352159	2,20E-16
80x2	783941	825918	**425485**	438931	2,20E-16
80x4	795645	817104	**437034**	469658	2,20E-16
80x8	804433	815757	**452669**	401275	2,20E-16
120x2	851031	889680	**438444**	423376	2,20E-16
120x4	870691	890372	**449977**	455684	2,20E-16
120x8	885035	895697	**518881**	515507	2,20E-16
PROMEDIO	733875	757382	**378272**	391761	

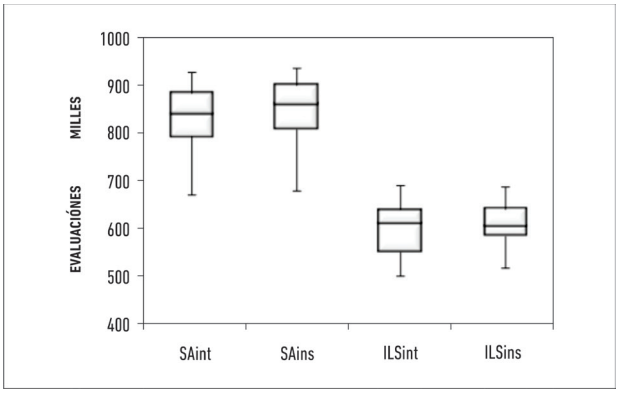

Figura 1. Boxplot de las evaluaciones para obtener las mejores soluciones para todos los algoritmos

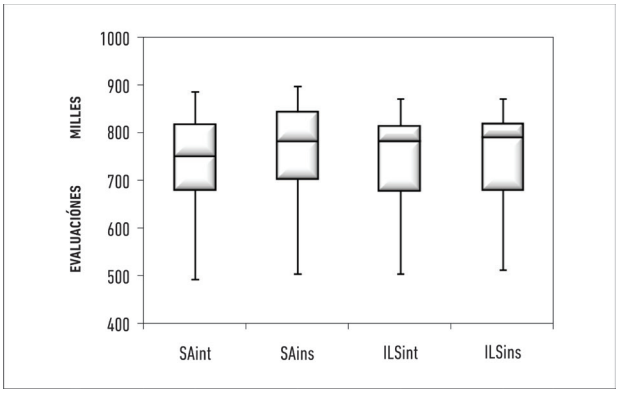

Figura 2. Boxplot del total de evaluaciones realizadas por los algoritmos durante todo el proceso de búsqueda.

Finalmente, analizamos la otra métrica interesante para medir el desempeño de una metaheurística que es el tiempo de ejecución necesario para alcanzar la mejor solución y se muestra en el cuadro VII. Como se puede observar, las variantes más rápidas son las correspondientes a ILS, con tiempos menores (de aproximadamente el 50 %) a los presentados por los algoritmos basados en SA, independientemente de la combinación de n y m que se analice. Esta es una situación esperada en función a lo observado previamente al analizar el esfuerzo numérico de cada variante. Cabe destacar que, todas las diferencias marcadas son estadísticamente significativas (valores p cercanos a cero en la última columna del cuadro VI).

CUADRO VII
TIEMPO PROMEDIO EMPLEADO POR CADA ALGORITMO PARA ENCONTRAR LA MEJOR SOLUCIÓN EN CADA GRUPO DE INSTANCIAS

Inst.	ALGORITMOS				ANÁLISIS DE VARIANZA (P-VALUE)
	SAint	SAins	ILSint	ILSins	
20x2	1,21	1,25	**0,31**	0,48	2,20E-16
20x4	2,10	0,93	**0,93**	0,88	2,20E-16
20x8	4,08	1,81	**1,81**	2,00	2,20E-16
50x2	7,05	7,11	**3,40**	3,78	2,20E-16
50x4	14,06	14,08	**6,58**	7,89	2,20E-16
50x8	29,05	29,11	**15,98**	15,01	2,20E-16
80x2	19,06	19,08	**10,18**	10,50	2,20E-16
80x4	38,04	38,13	**21,16**	22,66	2,20E-16
80x8	76,03	76,03	**43,78**	38,71	2,20E-16
120x2	43,03	43,05	**22,01**	21,21	2,20E-16
120x4	86,06	86,04	**44,78**	45,51	2,20E-16
120x8	172,06	172,06	**102,90**	102,09	2,20E-16
PROMEDIO	41,03	41,03	**22,82**	22,56	

Resumiendo, las técnicas basadas en ILS se desempeñan con casi un 50 % más de eficiencia que las variantes de SA para hallar los menores valores de makespan. Siendo nuestra propuesta, ILSint, la que presenta un mejor comportamiento en 7 de las 12 instancias.

CONCLUSIONES

En este trabajo se presentaron tres algoritmos para resolver el problema NP-duro SDST/HFFS: SAint, SAins e ILSint. En los dos primeros casos se trata de dos versiones de SA que utilizan los operadores de intercambio e inserción, respectivamente, para generar soluciones vecinas. En tanto que, ILSint resulta de una mejora realizada al ILS presentado en [16], renombrado en este trabajo como ILSins. Esta mejora consiste en modificar el operador de la búsqueda local. Es decir, se reemplazó el operador de inserción usado en ILSins por el de intercambio, surgiendo de esta manera ILSint.

Del análisis de los resultados obtenidos por estos cuatro algoritmos, en primer lugar se desprendió que las técnicas que usan el operador de intercambio realizan una explotación mucho más eficaz del espacio de búsqueda que al aplicar el operador de inserción. La razón principal de esto es la gran disrupción que genera el operador de inserción en las soluciones. En segundo lugar, se determinó con sustento estadístico, que los algoritmos basados en ILS obtuvieron soluciones de mejor calidad que los basados en SA y el esfuerzo computacional para lograrlo fue significativamente menor. Por último, es importante destacar que la versión de ILS propuesta en este trabajo obtuvo los mejores resultados empleando el menor esfuerzo computacional en un alto porcentaje de las instancias. Como futuras líneas de investigación, intentaremos en primer lugar diseñar e implementar operadores para generar soluciones vecinas que utilicen información heurística del problema en cuestión. Estos operadores se podrán utilizar luego en metaheurísticas tales como SA, ILS y GA con el fin de desarrollar algoritmos aún más eficientes que los actuales en la resolución de SDST/HFFS. Como una segunda línea de investigación, pretendemos incursionar en otras variantes del problema HFFS, dado que resulta necesario en diferentes actividades económicas desarrolladas en la región.

AGRADECIMIENTOS

El trabajo ha sido financiado por la UNLPam (proyecto 09/F059), a ANPCYT y PICTO-UNLPam-0278. La Dra Salto agradece a CONICET.

REFERENCIAS

U.K. Chakraborty. Minimizing total flow time in permutation flow shop scheduling with improved simulated annealing. In Evolutionary Computation, 2009. CEC '09. IEEE Congress on, pages 158–165, May 2009.

F. Choong, S. Phon-Amnuaisuk, and M.Y. Alias. Metaheuristic methods in hybrid flow shop scheduling problem. Expert Systems with Applications, 38:10787–10793, 2011.

E. Coffman and J. Bruno. Computer and job-shop scheduling theory. John Wiley & Sons, New York, 1976.

J. Gupta. Two-stage, hybrid flowshop scheduling problem. Journal of the Operational Research Society, 39(4):359–364, 1988.

C.W. Holsapple, V.S. Jacob, R. Pakath, and J.S. Zaveri. A genetics-based hybrid scheduler for generating static schedules in flexible manufacturing contexts. Systems, Man and Cybernetics, IEEE Transactions on, 23(4):953–972, Jul 1993.

P. Jarosław, S. Czesław, and Z˙ . Dominik. Optimizing bicriteria flow shop scheduling problem by simulated annealing algorithm. Procedia Computer Science, 18(0):936–945, 2013. 2013 International Conference on Computational Science.

A. Rinnooy Kan. Machine Scheduling Problems: Classification, complexity and computations. Martinus Nijhoff, The Hague, 1976.

S. Kirkpatrick, C.D. Gelatt Jr, and M.P. Vecchi. Optimization by simulated annealing. Science, (220):671–680, 1983.

T. Kis and E. Pesch. A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. European Journal of Operational Research, 164(3):592–608, 2005.

M. Kurz and R. Askin. Comparing scheduling rules for flexible flow lines. International Journal of Production Economics, 85(3):371–388, 2003.

M. Kurz and R. Askn. Scheduling flexible flow lines with sequence- dependent setup times. European Journal of Operational Research, 159(1):66–82, 2004.

H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In Handbook of Metaheuristics, pages 321–353. Kluwer Academic Publishers, 2002.

G. Minetti, E. Alba, and G. Luque. Seeding strategies and recombination operators for solving the dna fragment assembly problem. Inf. Process. Lett., 108(3):94–100, 2008.

G. Minetti, and C. Salto. Técnicas metaheurísticas para resolver una variante de aplicación industrial del problema de flowshop. En Actas del 2° Congreso Nacional de Ingeniería Informática/Sistemas de Información (CoNaIISI 2014); ISSN: 2346-992; pag. 247-254, 2014.

H. Mirsanei, M. Zandieh, M. Moayed, and M. Khabbazi. A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times. Journal of Intelligent Manufacturing, 22(965-978), 2011.

B. Naderi, R. Ruiz, and M. Zandieh. Algorithms for a realistic variant of flowshop scheduling. Computers & Operations Research, 37(2):236 – 246, 2010.

M. Nawaz, E. Enscore Jr, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega, 11(1):91–95, 1983.

R. Ruiz and C. Maroto. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research, 169(3):781–800, March 2006.

C. Salto, E. Alba, and J.M. Molina. Optimization Techniques for Solving Complex Problems, chapter Greedy Seeding and Problem- Specific Operators for GAs Solving Strip Packing Problems, pages 361–378. John Wiley & Sons, Inc., 2009.

A Sioud, M. Gravel, and C. Gagne. A genetic algorithm for solving a hybrid flexible flowshop with sequence dependent setup times. In Evolutionary

Computation (CEC), 2013 IEEE Congress on, pages 2512–2516, June 2013.

T. Stützle. Local search algorithms for combinatorial problems analysis, algorithms and new applications. Technical report, DISKI Dissertationen zur KÄunstliken Intelligenz, Sankt Augustin, Germany, 1999.

C.H. Westerberg and J. Levine. Investigation of different seeding strategies in a genetic planner. In Proceedings of the EvoWorkshops on Applications of Evolutionary Computing, pages 505–514, London, UK, UK, 2001. Springer-Verlag.

M. Zandieh, S. Fatemi Ghomi, and S. Moattar Husseini. An immune algorithm approach to hybrid flow shops scheduling with sequence dependent setup times. Applied Mathematics and Computation, 180(1):111–127, 2006.

Fecha de Presentación: 15/05/15
Fecha de Aprobación: 08/06/15