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

An ant colony algorithm for the permutation flowshop with sequence dependent setup times and makespan minimization

Eduardo Salazar Hornig[1] Natalia Pavón Weber[2]

Recibido 21 de diciembre de 2009, aceptado 30 de mayo de 2011
Received: December 21, 2009 Accepted: May 30, 2011

RESUMEN

En este trabajo se estudió el problema de secuenciamiento de trabajos en el taller de flujo de permutación con tiempos de preparación dependientes de la secuencia y minimización de makespan. Para ello se propuso un algoritmo de optimización mediante colonia de hormigas (ACO), llevando el problema original a una estructura semejante al problema del vendedor viajero TSP (*Traveling Salesman Problem*) asimétrico, utilizado para su evaluación de problemas propuestos en la literatura y se compara con una adaptación de la heurística NEH (*Nawaz-Enscore-Ham*). Posteriormente se aplica una búsqueda en vecindad a la solución obtenida tanto por ACO como NEH.

Palabras clave: Taller de flujo de permutación, tiempos de preparación, metaheurísticas, ACO, búsqueda local.

ABSTRACT

This paper studied the permutation flowshop with sequence dependent setup times and makespan minimization. An ant colony algorithm which turns the original problem into an asymmetric TSP (Traveling Salesman Problem) structure is presented, and applied to problems proposed in the literature and is compared with an adaptation of the NEH heuristic. Subsequently a neighborhood search was applied to the solution obtained by the ACO algorithm and the NEH heuristic.

Keywords: Permutation flowshop, sequence dependent setup times, metaheuristics, ACO, local search.

INTRODUCCIÓN

En un sistema de manufactura, la disposición de la maquinaria define el tipo de configuración productiva necesaria para realizar el proceso de transformación. Las necesidades de los clientes se traducen en órdenes de trabajo que se liberan y "transforman" en trabajos con fecha de entrega asociada. La programación de producción se preocupa de la asignación de recursos limitados a tareas productivas, y debe realizarse de manera detallada para mantener la eficiencia y el control de las operaciones dentro del sistema productivo y así constituir una ventaja competitiva difícil de imitar (ver, por ejemplo, [1-2]).

Los distintos productos requieren en su fabricación de distintas operaciones, las que se realizan en un orden y configuración productiva determinada, dependiendo del tipo de producto, su volumen de producción, la variedad de productos que se produce en la misma línea, etc. El *taller de flujo* consiste de *m* máquinas dispuestas en serie, donde se procesan trabajos de flujo unidireccional que requieren *m* operaciones,

[1] Departamento de Ingeniería Industrial. Universidad de Concepción. Casilla 160-C. Concepción, Chile. E-mail: esalazar@udec.cl
[2] Programa de Magíster en Ingeniería Industrial. Universidad de Concepción. Concepción, Chile.

cada una en una máquina distinta y siguiendo el mismo orden, esto es, primero en la máquina 1, después en la máquina 2, y así sucesivamente. Si la secuencia de trabajos permanece constante para todas las máquinas, el problema recibe el nombre de *taller de flujo de permutación*.

Para resolver el *taller de flujo de permutación* existen diferentes métodos exactos y heurísticos, constructivos o de mejora. Para el caso de dos máquinas, el clásico algoritmo de Johnson [3] obtiene la solución óptima del problema de *taller de flujo de permutación* con minimización de *makespan* (intervalo de tiempo en el que se procesa la totalidad de los trabajos) y denotado por C_{max}. Cuando se tienen tres o más máquinas se han desarrollado métodos heurísticos constructivos que entregan soluciones factibles, entre las que se destacan las conocidas heurísticas de Palmer, Gupta, MPS, CDS, RA y NEH diseñadas con el objetivo de minimización de *makespan* en el problema sin *setup*. Esta última, propuesta por Nawaz, Enscore y Ham en 1983 [4]. La heurística NEH, que construye una secuencia agregando trabajos de una lista de trabajos (ordenados de mayor a menor tiempo total de proceso) en forma sucesiva y evaluando múltiples inserciones de éstos en las secuencias parcialmente construidas, ha mostrado ser una buena heurística para este problema, existiendo un algoritmo eficiente desarrollado por [5]. Una revisión de este problema se presenta en [6-7].

También para este problema se han desarrollado metaheurísticas, incluyendo métodos de búsqueda en vecindad (búsqueda local), tales como *algoritmos genéticos*, *simulated annealing*, *tabu search* [5, 8-10]. Los trabajos que utilizan *ant colony optimization* se han concentrado principalmente en el *taller de flujo de permutación* sin tiempos de preparación con objetivos como la minimización de makespan [11-14], y objetivos relacionadas con la minimización de tiempo total de flujo [15-17]. El problema del *taller de flujo de permutación con tiempos de preparación dependientes de la secuencia* utilizando ACO ha sido poco tratado en la literatura, un ejemplo es el trabajo desarrollado en [18].

En las siguientes secciones se caracteriza el problema del taller de flujo de permutación, se describen las principales características de algoritmos *Ant Colony Optimization (ACO)*, en particular la versión de *Ant Colony System (ACS)* y se describe la aplicación ACS tratada en este trabajo para el problema del *taller de flujo de permutación con setup*. A continuación se describe la heurística NEH y el procedimiento de Búsqueda en Vecindad IP utilizados, los resultados obtenidos y las conclusiones.

EL TALLER DE FLUJO DE PERMUTACIÓN

El *problema de taller de flujo de permutación con tiempos de preparación dependientes de la secuencia* consiste en secuenciar un conjunto de n trabajos de flujo unidireccional en un sistema con m máquinas dispuestas en serie, trabajos que requieren m operaciones, cada una en una máquina distinta y siguiendo el mismo orden, siendo la secuencia de trabajos la misma para todas las máquinas, y el tiempo de preparación en el que se incurre al procesar cada nuevo trabajo depende del trabajo previamente procesado. Este tipo de configuración está presente en ambientes de manufactura de la industria química, farmacéutica, alimenticia, maderera, etc.

Desde el trabajo pionero de Johnson [3], que introduce el problema del taller de flujo de permutación para dos y tres máquinas sin tiempos de preparación dependientes de la secuencia, múltiples heurísticas se han desarrollado para resolver este problema y su generalización; en el año 1974 con el trabajo de Corwin y Esogbue [19] se trata un problema de dos máquinas con tiempos de preparación dependientes de la secuencia en la segunda máquina, el que es generalizado en 1975 con una propuesta de Gupta [20]. Sin embargo, este problema ha sido poco tratado con métodos metaheurísticos y computacionalmente eficientes para resolver problemas de gran tamaño [8].

El problema de secuenciamiento en un *taller de flujo* es un problema relacionado con la programación de producción: n trabajos deben ser procesados en m máquinas (etapas), y el tiempo de proceso del trabajo i en la máquina k está dado por p_{ik} (i = 1,...,n; k = 1,..., m). Estos tiempos son fijos, no negativos, y algunos de ellos pueden ser cero si algún trabajo no es procesado en alguna máquina. Los tiempos de preparación dependientes de la secuencia se denotan por s_{ijk}, representando la preparación de la máquina k cuando el trabajo j se procesa inmediatamente a continuación del trabajo i (i = 1, …, n; j = 1, … , n; k = 1, … , m), s_{iik} representa

el *setup* inicial si el trabajo i es el primer trabajo procesado en la máquina k. La minimización del *makespan* (C_{max}), objetivo considerado en este trabajo, consiste en minimizar el tiempo entre el inicio del procesamiento del primer trabajo en la primera máquina (incluyendo su tiempo de preparación) y la finalización del procesamiento del último trabajo en la última máquina, es decir, el intervalo de tiempo en el que se procesa completamente la totalidad de las órdenes de producción. Siguiendo la notación de Graham et al [21] este problema se denota por Fm/s_{ijk}, $prmu/C_{max}$.

Se consideran los siguientes supuestos:

- Cada trabajo debe ser procesado, a lo más, una vez en las máquinas 1, 2, ... , *m* (en ese orden).
- Previo al proceso de un trabajo la máquina debe prepararse, siendo el tiempo de preparación dependiente del trabajo procesado previamente.
- Cada máquina puede procesar sólo un trabajo a la vez.
- Un trabajo es procesado en una máquina a la vez.
- La secuencia de proceso para los trabajos es la misma en todas las máquinas. La secuencia común debe ser determinada, y define al problema como un *taller de flujo de permutación*.
- El proceso de un trabajo en una máquina no se puede interrumpir.
- Todos los trabajos son independientes entre sí y se encuentran disponibles en el instante cero.
- Las máquinas se encuentran continuamente disponibles.
- Se permite almacenar producto en curso (es posible que los trabajos esperen en cola por su proceso si es que la máquina requerida se encuentra ocupada.

En el problema de taller de flujo, en cada una de las *m* máquinas es posible secuenciar de *n*! posibles maneras, por lo que para encontrar una secuencia óptima sería necesario examinar $(n!)^m$ secuencias diferentes. Para el caso del *taller de flujo de permutación* (la misma secuencia de trabajos en todas las máquinas) el espacio de soluciones factibles, a pesar de que se reduce a *n*!, hace impracticable la obtención de la solución óptima para problemas de mediano a gran tamaño.

En este estudio se resolvió el problema de *taller de flujo de permutación con tiempos de preparación dependientes de la secuencia y minimización de makespan* mediante un algoritmo de colonia de hormigas tipo ACS (*ant colony system*). La relevancia de los problemas de programación con tiempos y/o costos de preparación dependientes de la secuencia queda de manifiesto en una amplia gama de configuraciones productivas [22].

El algoritmo propuesto reduce el problema de *m* máquinas a uno de una máquina, que es equivalente a un TSP asimétrico cuya resolución se realiza mediante un algoritmo ACS. La comparación del método se realiza contra las soluciones de la extensión del *benchmark* de Taillard [23], propuesta por Ruiz, Maroto y Alcaraz [8].

A modo de ilustración, consideremos un taller de flujo de tres máquinas en serie y cuatro trabajos a programar, cuyos tiempos de proceso se presentan en la Tabla 1, donde p_{ik} representa el tiempo de proceso del trabajo i en la máquina k. Los tiempos de *setup* en cada máquina se muestran en la Tabla 2, donde s_{ijk} representa el *setup* si a continuación del trabajo i se procesa el trabajo j en la máquina k.

Tabla 1. Tiempos de proceso de los trabajos (p_{ik}).

Trabajo	**1**	**2**	**3**	**4**
$\mathbf{p_{i1}}$	5	4	8	7
$\mathbf{p_{i2}}$	8	5	6	4
$\mathbf{p_{i3}}$	2	7	12	9

Tabla 2. Tiempos de *setup* (s_{ijk}).

M1 / s_{ij1}	**1**	**2**	**3**	**4**
1	**2**	2	3	1
2	4	**1**	1	2
3	2	3	**2**	5
4	5	1	2	**3**
M2 / s_{ij2}	**1**	**2**	**3**	**4**
1	**2**	2	1	1
2	4	**1**	2	3
3	2	5	**3**	4
4	4	2	3	**2**
M3 / s_{ij3}	**1**	**2**	**3**	**4**
1	**3**	3	5	6
2	1	**2**	6	3
3	3	2	**4**	5
4	2	4	3	**2**

Utilizaremos la heurística SPT (*shortest processing time*) para generar la secuencia de proceso en la máquina M_1 (por lo tanto en idéntica secuencia se procesan los trabajos en las máquinas M_2 y M_3): 2-1-4-3, generándose la programación presentada en la carta Gantt de la Figura 1. En esta figura, la barra sólida representa el tiempo de proceso de un trabajo y la barra achurada representa el tiempo de *setup* previo al proceso del respectivo trabajo; el color identifica a un trabajo, así el trabajo 1 (color violeta) inicia su proceso en M_1 en el tiempo 9 previa preparación de M_1 por 4 unidades de tiempo desde el momento en que finaliza el trabajo 2 en M_1, finalizando su proceso en M_1 en el tiempo 14; dado el carácter anticipatorio del *setup*, la preparación para procesar el trabajo 1 en M_2 parte una vez finalizado el proceso del trabajo 2 en M_2, por $s_{122} = 4$ unidades de tiempo, es decir, el trabajo 1 inicia su proceso en M_2 en el tiempo 14 y finaliza en el tiempo 22 dado que $p_{12} = 8$.

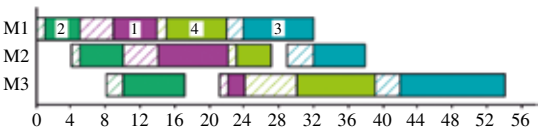

Figura 1. Carta Gantt-Programación Taller de Flujo.

Así, hasta finalizar el trabajo 3 en M_3 en el tiempo 54, obteniendo por lo tanto una programación con $C_{max} = 54$ (*makespan*).

ANT COLONY OPTIMIZATION (ACO)

La idea principal en este tipo de algoritmos es imitar el rastro de feromona que usan las hormigas en la naturaleza para comunicarse cuando transitan en la búsqueda de su alimento. Un procedimiento que simula este comportamiento fue desarrollado por Dorigo y otros [24], y se conoce como *optimización basada en colonias de hormigas* ACO (*ant colony optimization*), inicialmente utilizado para resolver el problema del vendedor viajero (TSP) para encontrar el tour más corto que recorre un conjunto de ciudades [25, 26].

Las colonias de hormigas artificiales corresponden a un conjunto de agentes computacionales simples que trabajan de manera cooperativa y se comunican mediante rastros de feromona artificiales. Por tanto, una hormiga artificial es un agente que se mueve de una ciudad a otra en un grafo TSP [27].

Los algoritmos ACO funcionan básicamente de la siguiente forma: *h* hormigas de la colonia transitan en forma independiente y de manera concurrente, siguiendo una regla de transición basada en la información local disponible en los arcos del grafo que representa el problema. Esta información local incluye la información heurística y los rastros de feromona (también llamada información memorística) para guiar la búsqueda. Opcionalmente, las hormigas pueden depositar feromona cada vez que recorren un arco (actualización local del rastro de feromona). Cuando todas las hormigas han completado un tour, se realiza la actualización global del rastro de feromona agregando una cantidad de feromona inversamente proporcional al largo del tour a los arcos de la mejor solución encontrada. La estructura genérica de un algoritmo ACO es [28]:

Procedimiento metaheurística ACO
Establecimiento de parámetro
Inicialización de rastros de feromona
Mientras (criterio de terminación no satisfecho) hacer
Construir hormigas y sus soluciones
Aplicar acciones adicionales (opcional)
Actualizar rastros de feromona
Fin mientras
Fin procedimiento

Además de lo explicado en el párrafo anterior, el modo de operación genérico de un algoritmo ACO puede incluir procedimientos adicionales, como acciones opcionales sin contrapunto natural (*daemon actions*) que se utilizan para implementar tareas desde una perspectiva global, como, por ejemplo, observar la calidad de todas las soluciones generadas y depositar una cantidad de feromona adicional en las transiciones asociadas a algunas de ellas, o aplicar un procedimiento de búsqueda local a las soluciones generadas por las hormigas antes de actualizar los rastros de feromona. En ambos casos, estas acciones reemplazan la actualización en línea a posteriori de feromona y el proceso pasa a llamarse actualización fuera de línea de rastros de feromona.

Sistema de Colonia de Hormigas (ACS)

El algoritmo ACO utilizado es el ACS (*ant colony system*) [26, 28], ya que es conocido como una de las versiones eficientes de ACO, su regla de transición, conocida como regla proporcional pseudo-aleatoria, permite balancear la intensificación y exploración en el espacio de soluciones [13] para resolver un problema TSP. El problema TSP tiene

un rol fundamental en ACO, dado que fue el primer problema tratado con este método [24, 25, 26, 28] existiendo una natural asociación entre este problema y la definición de la metaheurística ACO.

Cada una de las *h* hormigas consideradas es un agente simple, que encontrándose en la ciudad i elegirá ir a la ciudad j (aún no visitada por la hormiga) de acuerdo a una regla de transición que es función del nivel de feromona depositada en el arco (i,j) y de la visibilidad del arco (i,j). La notación siguiente se basa en la descripción del algoritmo [28].

Por τ_{ij} se representa el nivel de feromona depositada en el arco (i,j), y por η_{ij} a la visibilidad (información heurística) del arco (i,j), esta última definida como el inverso de la distancia d_{ij} que separa a la ciudad i de la ciudad j, esto es, $\eta_{ij} = 1/d_{ij}$.

La regla de transición permite balancear entre la exploración de nuevos caminos, representados por aquellos arcos con alto nivel de feromona y alta visibilidad (menor longitud), y la explotación de los conocimientos acumulados hasta el momento.

Una iteración del algoritmo corresponde al movimiento que agrega una ciudad al tour en construcción de cada una de las *h* hormigas, por lo que al cabo de *n* iteraciones se completa un ciclo, habiendo construido cada hormiga una solución factible.

La *regla de transición* (1) permite a una hormiga k posicionada en el nodo i elegir el nodo j_0 como siguiente nodo en su tour, maximizando la razón $\tau_{ij} \cdot \eta_{ij}^{\beta}$ con probabilidad q_0, o explotando el conocimiento acumulado en la red con probabilidad $(1\text{-}q_0)$.

$$j_0 = \begin{cases} \arg\left\{ \max_{j \in F_k(i)} \left[\tau_{ij} \cdot \eta_{ij}^{\beta} \right] \right\} & c.p. \quad q_0 \\ J & c.p. \quad (1 - q_0) \end{cases} \tag{1}$$

donde q_0 es un parámetro tal que $0 \leq q_0 \leq 1$, que determina la importancia relativa entre intensificación y explotación, y $F_k(i)$ es el conjunto de nodos no secuenciados para una hormiga k posicionada en el nodo i. J es una variable aleatoria con función de probabilidades dada por (2).

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij} \cdot \eta_{ij}^{\beta}}{\sum_{l \in F_k(i)} \tau_{il} \cdot \eta_{il}^{\beta}} & si \quad j \in F_k(i) \\ 0 & e.o.c. \end{cases} \tag{2}$$

Existen dos tipos de actualización del nivel de feromona de cada arco, una *actualización global* y una *actualización local*.

En ACS se realiza sólo una actualización fuera de línea (actualización global), que opera solamente sobre la mejor solución encontrada hasta el momento, por lo que sólo los arcos de esta solución presentarán retroalimentación. La actualización global es realizada una vez que todas las hormigas han completado el tour.

El nivel de feromona es actualizado de acuerdo a (3), donde γ es un parámetro tal que $0 \leq \gamma \leq 1$, que representa el decaimiento en el nivel de feromona, y L_b es la distancia del mejor tour encontrado hasta el momento.

$$\tau_{ij} \leftarrow (1\text{-}\gamma) \cdot \tau_{ij} + \gamma / L_b \tag{3}$$

El primer factor de la actualización de feromona corresponde a la evaporación de los rastros en todas las conexiones utilizadas por la mejor hormiga global, y el segundo factor corresponde a la deposición.

Además, en ACS las hormigas aplican una actualización en línea de los rastros de feromona (actualización local), lo que favorece la generación de soluciones distintas a las ya encontradas. La actualización local se realiza según (4) inmediatamente después que la hormiga transita el arco (i,j).

$$\tau_{ij} \leftarrow (1\text{-}\rho) \cdot \tau_{ij} + \rho \cdot \tau_0 \tag{4}$$

En (4) $\rho \in (0,1]$ es un segundo parámetro de decremento de feromona, y τ_0 el valor del rastro de feromona inicial. La regla de actualización en línea paso a paso incluye tanto la evaporación de feromona como la deposición de la misma. Dado que la cantidad de feromona depositada es menor a la evaporada, esta actualización hace que los rastros de feromona en los arcos inmediatamente transitados disminuyan. Así, esto lleva a una técnica de exploración adicional del ACS, ya que los arcos

transitados por un gran número de hormigas son cada vez menos atractivos para el resto de hormigas que en la iteración actual, lo que ayuda a diversificar los caminos que construyen las hormigas.

Inicialmente, cada hormiga es puesta aleatoriamente en una ciudad y luego se aplica la regla de transición definida. Para construir una solución factible cada hormiga tiene una memoria, en la cual se almacena el viaje parcial actual. La memoria se utiliza para determinar en cada paso de la construcción de la ruta las ciudades que aún no han sido visitadas y así garantizar la construcción de una solución factible.

Los valores de los parámetros varían dependiendo del modelo ACO diseñado, de la aplicación y de la complejidad del problema.

MODELO ACS PARA EL TALLER DE FLUJO

Se presenta el modelo ACS para el *taller de flujo de permutación con tiempos de preparación dependientes de la secuencia* desarrollado en este trabajo. La evaluación del algoritmo se realiza utilizando instancias de la extensión por Ruiz, Maroto y Alcaraz [8] del *benchmark* de Taillard [23]. El *benchmark* de Taillard propone instancias de *flowshop de permutación* con tiempos de proceso de los trabajos distribuidos uniforme discreto entre 1 y 99 (p_{ij} ~ UD[1,99]). Se consideraron 10 instancias de problemas para cada combinación de n = 20, 50 y 100 trabajos con m = 5, 10 y 20 máquinas, en total 90 instancias (ver Tabla 3).

La extensión [8] agrega a cada instancia tiempos de preparación dependientes de la secuencia de a lo más 10%, 50%, 100% y 125% de los mayores tiempos de proceso de las instancias originales de Taillard, constituyendo así cuatro conjuntos de 90 problemas cada uno: SDST10, SDST50, SDST100 y SDST125, con tiempos de preparación distribuidos UD[1,9], UD[1,49], UD[1,99] y UD[1,125] respectivamente.

Se consideraron tiempos de preparación anticipatorios, es decir, la preparación de la máquina para procesar el siguiente trabajo puede comenzar tan pronto como la máquina queda libre. La evaluación de la heurística se realizó por medio de rutinas en lenguaje

Tabla 3. Instancias de Taillard.

Instancias	n	m
ta001-ta010	20	5
ta011-ta020	20	10
ta021-ta030	20	20
ta031-ta040	50	5
ta041-ta050	50	10
ta051-ta060	50	20
ta061-ta070	100	5
ta071-ta080	100	10
ta081-ta090	100	20

C/C++ adaptadas del software SPS_Optimizer [29], herramienta diseñada para la programación de operaciones en un computador Centrino Pro Intel Core 2 Duo T7500 de 2.2 GHz de 2 GB de RAM con Windows XP.

Para evaluar el desempeño se utilizó el *makespan* (C_{max}) como medida de desempeño, que corresponde al intervalo de tiempo en el que se procesan completamente todos los trabajos. El *makespan* entregado por el método se compara contra la mejor solución conocida (MSC). Para determinar el porcentaje de incremento sobre el mejor makespan conocido (%MSC), se utiliza la medida:

$$\%MSC = \frac{\text{Sol}_{\text{Método}} - \text{MSC}}{\text{MSC}} * 100 \qquad (5)$$

$\text{Sol}_{\text{Método}}$ es la solución obtenida con el método en estudio y MSC es la mejor solución conocida de la respectiva instancia en la extensión [8] del *benchmark* de Taillard. Los valores de MSC actualizados fueron proporcionados por uno de los autores de dicha extensión.

El problema del *taller de flujo de permutación con tiempos de preparación dependientes de la secuencia* de m máquinas con n trabajos se transforma a un problema de 1 máquina con tiempos de preparación dependientes de la secuencia y n trabajos, problema con similar estructura a la del TSP asimétrico (ATSP), que al ser resuelto mediante el algoritmo ACO traspasa luego su solución al problema original, enfoque que también ha sido utilizado en el mismo problema [30-31], y en otros problemas [32].

En este problema el tiempo de proceso del trabajo i se define como la suma de los tiempos

de proceso de este trabajo en todas las máquinas, de acuerdo a:

$$p_i = \sum_{k=1}^{m} p_{ik} \quad (6)$$

Al tratarse la solución de una secuencia permutación, esto es, la misma secuencia de proceso en todas las máquinas, el tiempo de preparación del trabajo j al programarse a continuación del trabajo i queda definido como la suma de los tiempos de preparación para la secuencia parcial i $\rightarrow$ j en cada máquina:

$$s_{ij} = \sum_{k=1}^{m} s_{ijk} \quad (7)$$

Sin importar en qué orden se procesan los trabajos, los tiempos de proceso son siempre los mismos, por lo que la matriz de distancias de un TSP se asocia con la matriz de tiempos de preparación del problema de una máquina. Así, la información heurística o visibilidad del algoritmo ACO se define como:

$$\eta_{ij} = \frac{1}{s_{ij}}, \quad (8)$$

donde s_{ij} es el tiempo de preparación requerido para procesar el trabajo j luego del trabajo i en el problema de una máquina. Cabe recordar que η_{ij} es un indicador de la deseabilidad de recorrer el arco (i,j), por lo que tiene sentido asociarlo al tiempo de preparación: a mayor tiempo de preparación asociado al programar el trabajo j luego de i es menos deseable que la secuencia incluya los trabajos en ese orden.

El rastro inicial de feromona queda determinado por:

$$\tau_0 = \frac{1}{n \cdot L_0}, \quad (9)$$

donde el valor L_0 corresponde al mejor *makespan* obtenido de una muestra aleatoria de cinco secuencias. El valor *n* representa el número de trabajos a secuenciar, y la regla de transición está dada por las ecuaciones (1) y (2).

La actualización de feromona fuera de línea se realiza sobre los arcos de la mejor solución obtenida (S_b) de acuerdo a la ecuación (3), y la actualización en línea paso a paso de feromona es realizada cada vez que una hormiga transita por un arco (i,j) de acuerdo a la ecuación (4).

El valor del parámetro γ, factor de evaporación de feromona en la actualización global, se considera en este estudio igual a ρ. De acuerdo a lo recomendado en la literatura [26], el número de hormigas *h* se consideró igual a 10, y basado en referencias de la literatura [27] y de un análisis experimental previo se realizaron cinco réplicas por problema, combinando los aspectos de eficiencia computacional y calidad de solución.

Los restantes parámetros del ACS: ρ, β, q_0 y ciclos se calibraron de forma experimental, basado en valores de referencia de la literatura para el problema TSP. Los parámetros del análisis experimental se resumen en la Tabla 4.

Tabla 4. Parámetros utilizados en el estudio.

Parámetro	Valor
h	10
ρ	0,4-0,5
β	3-4-5
q_0	0,95
ciclos	3.500
réplicas	5

El gráfico de la Figura 2 presenta el %MSC promedio de las cinco réplicas (eje vertical) de cada instancia para la combinación de parámetros β=3 y ρ=0,4 para los cuatro grupos de problemas, en el eje horizontal se representan las 90 instancias para cada grupo de problemas (SDST10, SDST50, SDST100 y SDST125); cada subdivisión incluye 10 problemas en el orden indicado en la Tabla 3 (para efectos de claridad se omite en el eje horizontal la identificación explícita de cada problema). Se observa que a medida que aumenta el máximo del tiempo de preparación respecto del tiempo de proceso, la calidad de las soluciones aumenta, es decir, los valores de %MSC tienden a ser menores (la línea SDST10 tiende a estar sobre la línea SDST50, ésta a su vez tiende a estar sobre la línea SDST100, la que a su vez tiende a estar sobre la línea SDST125). Similar comportamiento se observó para todas las combinaciones de β y ρ consideradas. En general, se podría afirmar que el rendimiento del ACS mejora a medida que los tiempos de preparación aumentan respecto de los tiempos de proceso.

A través de un estudio experimental previo se descartaron valores de parámetros de β y ρ que mostraron evidente mal desempeño del algoritmo. Para las mejores combinaciones de estos parámetros se realizó un análisis de varianza para la variable %MSC obtenida por el algoritmo ACS, cuyos resultados se presentan en la Tabla 5. Al observar los valores-p se infiere que con un nivel de significancia α = 0,05 ninguno de los factores considerados incide en el valor de %MSC, es decir, no afectan el buen desempeño de la metaheurística para los niveles de ρ (0,4 y 0,5) y de β (3, 4 y 5) considerados, confirmando también para este problema rangos de valores de β y ρ recomendados en la literatura.

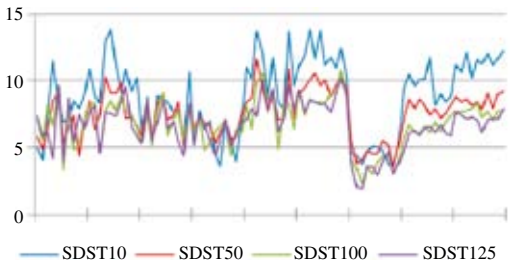

Figura 2. %MSC promedio para β = 3 y ρ = 0,4.

Tabla 5. ANOVA para %MSC utilizando ACS.

Fuente	**SC**	**gl**	**CM**	**F**	**p**
Efectos principales					
A (β)	0,855174	2	0,427587	0,07	0,9279
B (ρ)	2,17869	1	2,17869	0,38	0,5369
Interacciones					
AB	1,28351	2	0,641756	0,11	0,8938
Residuos	12307,5	2154	5,71377		
Total (corregido)	12311,8	2159			

En aquellos problemas en que los tiempos de preparación son anticipatorios y grandes en relación a los tiempos de proceso, una buena secuenciación implica pocos tiempos muertos, pues en el tiempo previo al proceso de un trabajo una máquina necesariamente debiera estar ejecutando la preparación sin esperas adicionales. En esta situación, cualquier modificación en la secuencia afecta directamente al *makespan*, por lo que es importante contar con algoritmos que consideren y/o exploten esta situación.

HEURÍSTICA NEH Y BÚSQUEDA EN VECINDAD

El algoritmo ACS se comparó con la heurística NEH adaptada al problema del *taller de flujo de permutación con tiempos de preparación dependientes de la secuencia*. La lista inicial se obtuvo realizando una estimación del tiempo de proceso de cada trabajo i en la etapa k como la suma entre el correspondiente tiempo de proceso y el promedio de los posibles tiempos de preparación para el trabajo i en la etapa k como una estimación de su tiempo de preparación antes de su proceso:

$$pe_{ik} = p_{ik} + \sum_{j=1}^{n} s_{jik} \,/\, n$$

Con el fin de mejorar los resultados obtenidos con el algoritmo ACS y NEH, se implementó una búsqueda en vecindad de tipo IP (intercambio de pares) aplicada a la solución entregada por ACS (ACS+BV) y a la solución entregada por NEH (NEH+BV).

La vecindad IP de una secuencia de *n* trabajos se define como el conjunto de todas las secuencias que se obtienen a partir de una secuencia inicial (denominada semilla) realizando los n·(n-1)/2 posibles intercambios de a pares entre elementos de la secuencia. La Figura 3 ilustra la vecindad IP obtenida a partir de una secuencia semilla para un problema con n = 4 trabajos; en negrita los elementos intercambiados respecto de la semilla.

Semilla 3 – 2 – 1 – 4

Vecindad
2 – **3** – 1 – 4
1 – 2 – **3** – 4
4 – 2 – 1 – **3**
3 – **1** – **2** – 4
3 – **4** – 1 – **2**
3 – 2 – **4** – **1**

Figura 3. Ilustración de vecindad IP.

En la implementación realizada para este estudio, la semilla corresponde a la solución entregada por el respectivo algoritmo, y el proceso iterativo realiza la búsqueda en cada vecindad, redefiniendo la semilla al momento de encontrar una solución que mejora la semilla actual. La búsqueda termina una vez que no se encuentra una mejor solución dentro de la vecindad explorada.

RESULTADOS

La Tabla 6 presenta el orden de magnitud de los tiempos CPU en segundos para los distintos tamaños de problema para ACS (por réplica), BV y NEH. Las instancias de mayor tamaño (100 trabajos y 20 máquinas) se resolvieron en un tiempo aproximado de un minuto para ACS, característica que indica competitividad de la heurística en relación al tiempo de ejecución.

La Tabla 6 presenta el orden de magnitud de los tiempos CPU en segundos para los distintos tamaños de problema para ACS (por réplica), BV y NEH. Las instancias de mayor tamaño (100 trabajos y 20 máquinas), se resolvieron en un tiempo aproximado de un minuto para ACS, característica que indica competitividad de la heurística en relación al tiempo de ejecución.

Tabla 6. Tiempos CPU [s] de heurística ACS y BV.

n	m	CPU-ACS	CPU-BV	CPU-NEH
20	5	3,50	0,00	0,00
	10	4,00	0,02	0,01
	20	4,20	0,02	0,02
50	5	17,00	0,20	0,05
	10	17,50	0,20	0,10
	20	18,50	0,50	0,20
100	5	60,00	2,00	0,30
	10	62,00	3,00	0,50
	20	65,00	5,00	0,50

Para cada instancia se considera la mejor solución obtenida entre las diferentes réplicas y combinaciones de parámetros β y ρ. La solución de ACS+BV en cada instancia resulta de la mejor solución obtenida al aplicar la búsqueda en vecindad a todas las soluciones generadas por ACS en cada instancia.

La Tabla 7 muestra estadísticas de %MSC promedio según %Setup para ACS y la heurística NEH sin y con búsqueda en vecindad (BV). Se destaca el efecto positivo de incluir la búsqueda en vecindad (ver también este efecto en las Tablas 8a hasta 8d).

El algoritmo ACS presenta un desempeño aceptable, considerando que los valores de MSC han sido obtenidos producto de variados estudios de optimización. Es posible afirmar que el algoritmo ACS propuesto es competitivo para aquellos problemas en los que los tiempos de preparación se generan en un mayor rango, es decir, en problemas donde existe alta variabilidad en los tiempos de preparación y en promedio mayores a los tiempos de proceso, sin embargo, esta característica desaparece al agregar la búsqueda en vecindad (ACS+BV) aplicada a la solución entregada por ACS.

Tabla 7. Estadísticas de %MSC.

%Setup	ACS	ACS+BV	NEH	NEH+BV	HGA
SDST10	6,82	2,28	4,22	2,96	0,42
SDST50	5,57	3,41	6,56	5,62	1,06
SDST100	4,49	3,36	9,08	8,18	1,55
SDST125	4,19	3,31	9,81	8,75	1,65
Promedio	5,27	3,09	7,42	6,38	1,17

La búsqueda en vecindad mejoró las soluciones de *makespan* obtenidas con ACS en promedio en un 2,18%, mientras que para la heurística NEH mejoró las soluciones en promedio en un 1,04%.

Tabla 8a. Estadísticas de %MSC para SDST10.

SDST10	ACS	ACS+BV	NEH	NEH+BV
20*05	4,99	1,44	3,79	2,26
20*10	7,14	2,15	4,49	3,06
20*20	5,99	1,78	3,60	3,05
50*05	4,14	1,85	3,36	2,08
50*10	8,81	3,19	5,36	3,58
50*20	9,99	3,42	5,76	4,69
100*05	3,27	1,39	3,26	1,99
100*10	7,36	2,22	3,68	2,44
100*20	9,70	3,08	4,64	3,46
Promedio	6,82	2,28	4,22	2,96

Por otro lado, la heurística NEH muestra un desempeño promedio superior o similar a ACS en problemas con baja variabilidad de tiempos de preparación (SDST10 y SDST50), pero es claramente superada por ACS en problemas con alta variabilidad de los tiempos de preparación (SDST100 y SDST125), manteniendo parcialmente esta característica al agregar la búsqueda en vecindad (NEH+BV). Esta situación se detalla en las Tablas 8a) a 8d) que muestran estadísticas de %MSC separadas por grupo de instancias (SDST10, SDST50, SDST100 y SDST125 respectivamente).

Tabla 8b. Estadísticas de %MSC para SDST50.

SDST50	ACS	ACS+BV	NEH	NEH+BV
20*05	4,66	2,70	6,41	5,09
20*10	5,47	2,76	6,05	4,33
20*20	5,40	2,41	4,21	3,58
50*05	4,73	4,00	8,76	7,46
50*10	6,53	4,82	7,27	6,23
50*20	7,62	4,57	6,00	5,32
100*05	3,00	2,05	8,19	7,64
100*10	5,77	3,51	6,78	5,95
100*20	6,97	3,88	5,41	4,97
Promedio	5,57	3,41	6,56	5,62

Tabla 8c. Estadísticas de %MSC para SDST100.

SDST100	ACS	ACS+BV	NEH	NEH+BV
20*05	3,62	3,34	11,01	9,23
20*10	4,35	2,82	6,72	5,75
20*20	4,76	2,43	5,90	4,51
50*05	3,66	3,64	12,75	12,14
50*10	5,61	4,81	10,51	9,03
50*20	6,77	4,95	7,31	6,76
100*05	1,76	1,39	11,69	10,98
100*10	4,29	3,01	8,62	8,26
100*20	5,58	3,88	7,18	6,92
Promedio	4,49	3,36	9,08	8,18

Tabla 8d. Estadísticas de %MSC para SDST125.

SDST125	ACS	ACS+BV	NEH	NEH+BV
20*05	3,97	3,59	10,80	8,80
20*10	4,04	3,03	8,45	7,42
20*20	4,54	2,61	5,85	4,76
50*05	3,94	3,74	13,94	12,03
50*10	5,18	4,77	11,48	9,95
50*20	6,12	4,82	8,32	8,05
100*05	1,24	1,13	12,30	11,61
100*10	3,74	2,79	10,18	9,42
100*20	4,99	3,33	6,97	6,67
Promedio	4,19	3,31	9,81	8,75

Lo anterior permite indicar que para el caso del *taller de flujo de permutación con tiempos de preparación dependientes de la secuencia*, aplicar una búsqueda en vecindad utilizando como partida las soluciones entregadas por ACS, entrega soluciones competitivas agregando un tiempo no significativo (la implementación de BV no suma al orden de magnitud del tiempo CPU de ACS).

La Tabla 7 muestra también los resultados promedio del algoritmo HGA (*hybrid genetic algorithm*) desarrollado por Ruiz, Maroto y Alcaraz [8], con un rendimiento promedio de 1,17%, mejor que el rendimiento promedio de 3,09% obtenido por ACS+BV. En el mencionado estudio, el algoritmo HGA es comparado con siete algoritmos de la literatura, siendo HGA el de mejor rendimiento. En ese estudio ACS+BV sería el de segundo mejor rendimiento luego de HGA en los problemas SDST125 y SDST100, y el de tercer y quinto mejor rendimiento en los problemas SDST50 y SDST10 respectivamente.

CONCLUSIONES

El trabajo analizó el comportamiento de un algoritmo ACS de optimización de colonia de hormigas aplicado a la programación de trabajos en la configuración productiva de *taller de flujo de permutación con tiempos de preparación anticipatorios y dependientes de la secuencia* con minimización de makespan (C_{max}). Un análisis de varianza realizado para la variable %MSC permitió inferir que con un nivel de significancia $\alpha = 0{,}05$, ninguno de los factores considerados tenía una incidencia significativa en el desempeño del algoritmo ACS. Lo anterior permite afirmar que para los valores de los parámetros ρ = [0,4-0,5] y β = [3-4-5] el algoritmo ACS muestra estabilidad en su rendimiento, por lo que se recomienda su uso.

Para efecto de comparación se implementó una extensión de la heurística NEH al problema de *taller de flujo con tiempos de preparación dependientes de la secuencia*. Con el objetivo de mejorar la solución entregada tanto por el algoritmo ACS como por la heurística NEH se implementó una búsqueda en vecindad IP. Para la evaluación de los algoritmos se utilizó parte de la extensión del benchmark de Taillard [23] utilizado por Ruiz, Maroto y Alcaraz [8].

El algoritmo ACS mejora su rendimiento a medida que aumenta el porcentaje máximo de tiempo de preparación respecto del tiempo de proceso, a diferencia de la heurística NEH que muestra un comportamiento inverso. Esta característica no

se mantiene en forma tan marcada al aplicar la búsqueda en vecindad IP, mostrando ACS+BV mejor desempeño promedio que NEH+BV, en particular en problemas con mayor porcentaje máximo de tiempo de preparación respecto del tiempo de proceso.

La aplicación de búsqueda en vecindad IP a la solución obtenida con ACS mejora en promedio en un 2,38%, con resultados entre 0,90% y 7,65% promediando 4,48%. De acuerdo a esto, no se puede afirmar que el algoritmo presentado sea competitivo para todo tipo de problema de *taller de flujo de permutación con tiempos de preparación dependientes de la secuencia*.

Los tiempos de ejecución son del orden de un minuto por réplica para las instancias de mayor tamaño, sin que la búsqueda en vecindad influya en el orden de magnitud del tiempo de ejecución, por lo que se puede afirmar que complementar ACS con una búsqueda en vecindad mejora las soluciones con bajo costo computacional.

La aplicación de la búsqueda en vecindad mejora el desempeño de la heurística NEH en 1,04%. De la comparación de ACS+BV con la heurística constructiva NEH+BV se observa que a medida que aumenta el % del tiempo máximo de preparación respecto del tiempo de proceso esta última baja su rendimiento; sin embargo, esto puede ser compensado con el bajo tiempo de ejecución del algoritmo (del orden de 0,50 segundos en los problemas de mayor tamaño). ACS+BV muestra un rendimiento claramente superior a NEH+BV, especialmente en problemas con mayor % de tiempo de preparación máximo respecto del tiempo de proceso.

Posiblemente la simplificación que se hace del problema original, y el hecho de no considerar en forma explícita el tipo de configuración productiva tratada, puede incidir en la obtención de mejores resultados de ACS+BV. Al redefinir la matriz de costo/distancia que se genera para dar estructura de TSP al problema, de manera tal que considere explícitamente la estructura del sistema productivo en estudio podría mejorar su rendimiento.

Finalmente, se debe tener en cuenta que al momento de elegir qué método se utilizará para programar las operaciones, es necesario considerar el desempeño del método en términos de calidad de la solución y tiempo computacional requerido, ya que para un problema de programación de producción éste debe ser capaz de entregar buenas soluciones en un tiempo breve. En este sentido pueden servir heurísticas constructivas como la heurística NEH fáciles de comprender e implementar.

REFERENCIAS

[1] M. Pinedo. "Planning and Scheduling in Manufacturing and Services". Springer. 2005.

[2] J. Heizer y B. Render. "Dirección de la producción y de operaciones-Decisiones tácticas". Octava Edición. Prentice-Hall. 2007.

[3] S.M. Johnson. "Optimal two- and three-stage production schedules with setup times included". Naval Research Logistics Quarterly. Vol. 1, Issue 1, pp. 61-68. 1954.

[4] M. Nawaz, E. Enscore and I. Ham. "A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem". OMEGA. Vol. 11, Issue 1, pp. 91-95. 1983.

[5] E. Taillard. "Some efficient heuristic methods for the flow shop sequencing problem". European Journal of Operational Research. Vol. 47, Issue 1, pp. 65-74. 1990.

[6] S.R. Hejazi and S. Saghafian. "Flowshop-scheduling problems with makespan criterion: a review". Int. Journal of Production Research. Vol. 43, Issue 14, pp. 2895-2929. 2005.

[7] R. Ruiz and C. Maroto. "A comprehensive review and evaluation of permutation flowshop heuristics". European Journal of Operational Research. Vol. 165, Issue 2, pp. 479-494. 2005.

[8] R. Ruiz, C. Maroto and J. Alcaraz. "Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics". European Journal of Operational Research. Vol. 165, Issue 1, pp. 34-54. 2005.

[9] R. Ruiz and C. Maroto. "A genetic algorithm for hybrid flowshop with sequence dependent setup times and machine eligibility". European Journal of Operational Research. Vol. 169, pp. 781-800. 2006.

[10] R. Ruiz and T. Stützle. "An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives". European

Journal or Operational Research. Vol. 187, Issue 3, pp. 1143-1159. 2008.

[11] T. Stützle". An ant approach to the flow shop problem". In: Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing (EUFIT'98), pp. 1560-1564. Aachen, Germany. 1998.

[12] K-C Ying and C-J Liao. "An ant colony system for permutation flow-shop sequencing". Computers & Operations Research. Vol. 31, Issue 5, pp. 791-801. 2004.

[13] F. Ahmadizar, F. Barzinpour and J. Arkat. "Solving Permutation Flow Shop Sequencing using Ant Colony Optimization". IEEE International Conference Industrial Engineering and Engineering Management. 2007.

[14] B.M.T. Lin, C.Y. Lu, S.J. Shyu and C.Y. Tsai. "Development of new features of ant colony optimization for flowshop scheduling". International Journal of Production Economics. Vol. 112, Issue 2, pp. 742-755. 2008.

[15] Y. Gajpal and C. Rajendran. "An ant-colony optimization algorithm for minimizing the completion-time variance of jobs in flowshops". International Journal of Production Economics. Vol. 101, Issue 2, pp. 259-272. 2006.

[16] C. Rajendran and H. Ziegler. "Two ant-colony algorithms for minimizing total flowtime in permutation flowshops". Computers & Industrial Engineering. Vol. 48, Issue 4, pp. 789-797. 2005.

[17] 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.

[18] Y. Gajpal, C. Rajendran and H. Ziegler. "An ant colony algorithm for scheduling in flowshops with sequence-dependent setup times of jobs". International Journal of Advanced Manufacturing Technology. Vol. 30, Issue 5-6, pp. 416-424. 2006.

[19] B.D. Corwin and A.O. Esogbue. "Two machine flowshop scheduling problems with sequence dependent setup times: A dynamic programming approach". Naval Research Logistics Quarterly. Vol. 21, pp. 515-524. 1974.

[20] J.N.D. Gupta. "A search algorithm for the generalized flowshop scheduling problem". Computers & Operations Research. Vol. 2, Issue 2, pp. 83-90. 1975.

[21] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan. "Optimization and approximation in deterministic sequencing and scheduling theory: a survey". Annals of Discrete Mathematics. Vol. 5, pp. 287-326. 1979.

[22] A. Allahverdi, C.T. Ng, T.C.E Cheng and M. Kovalyov. "A survey of scheduling problems with setup times or costs". European Journal of Operational Research. Vol. 187, Issue 3, pp. 985-1032. 2008.

[23] E. Taillard. "Benchmarks for basic scheduling problems". European Journal of Operational Research. Vol. 64, Issue 2, pp. 278-285. 1993.

[24] M. Dorigo, V. Maniezzo and A. Colorni. "Possitive feedback as a search strategy". Technical Report Nº 91-016. Dipartimento di Elettronica Politecnico di Milano. Italy. 1991.

[25] M. Dorigo and L. Gambardella. "Ant Colonies for the Traveling Salesman Problem". BioSystems. Vol. 43, pp. 73-81. 1997a.

[26] M. Dorigo and L.M. Gambardella. "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". IEEE Transactions on Evolutionary Computation. Vol. 1, Issue 1, pp. 53-66, 1997b.

[27] M. Dorigo, V. Maniezzo and A. Colorni. "Ant System: Optimization by a Colony of Cooperative Agents". IEEE Transactions on System, Man and Cybernetics-Part B. Vol. 26, Issue 1, pp. 29-41. 1996.

[28] M. Dorigo, G. Di Caro and L. Gambardella. "Ant Algorithms for Discrete Optimization". Artificial Life. Vol. 5, Issue 2, pp. 137-172. 1999.

[29] E. Salazar. "Programación de Producción con SPS Optimizer". Actas (CD) Optima 2005. Valdivia, Chile. 25-27 Octubre 2005.

[30] J.V. Simons. "Heuristics in Flow Shop Scheduling with Sequence Dependent Set-up Times". Omega. Vol. 20, Issue 2, pp. 215-225. 1992.

[31] R. Ruiz-Mercado and J. Bard. "An Enhanced TSP-Based Heuristic for Makespan Minimization in a Flow Shop with Setup Times". Journal of Heuristics. Vol. 5, Issue 1, pp. 53-70. 1999.

[32] E. Salazar y N. Ruiz. "Modelo ACO para la Recolección de Residuos por Contenedores". Ingeniare. Revista chilena de ingeniería. Vol. 17 Nº 2, pp. 236-243. 2009.