Ingeniería Investigación y Tecnología, volumen XIV (número 1), enero-marzo 2013: 43-51
ISSN en trámite, FI-UNAM
(artículo arbitrado)

Minimización del *makespan* en máquinas paralelas idénticas con tiempos de preparación dependientes de la secuencia utilizando un algoritmo genético

Makespan Minimization for The Identical Machine Parallel Shop with Sequence Dependent Setup Times Using a Genetic Algorithm

Salazar-Hornig E.
Facultad de Ingeniería
Universidad de Concepción, Concepción, Chile
Correo: esalazar@udec.cl

Medina-S.J.C.
Facultad de Administración y Negocios
Universidad de las Américas, Santiago, Chile
Correo: jmedinas@udla.cl

Información del artículo: recibido: mayo de 2010, aceptado: abril de 2012

Resumen

Se presenta un algoritmo genético para la programación de trabajos en un sistema de máquinas paralelas idénticas, con tiempos de preparación dependientes de la secuencia, con el objetivo de minimizar el *makespan* (C_{max}). El algoritmo genético se compara con otros métodos heurísticos sobre un conjunto de problemas de prueba generados aleatoriamente. Posteriormente se introduce un procedimiento de mejora en el proceso evolutivo del algoritmo genético que mejora significativamente su desempeño.

Descriptores:

- máquinas paralelas idénticas
- algoritmos genéticos
- heurísticas

Abstract

A genetic algorithm for the parallel shop with identical machines scheduling problem with sequence dependent setup times and makespan (C_{max}) minimization is presented. The genetic algorithm is compared with other heuristic methods using a randomly generated test problem set. A local improvement procedure in the evolutionary process of the genetic algorithm is introduced, which significantly improves its performance.

Keywords:

- *identical parallel machines*
- *genetic algorithms*
- *heuristics*

Introducción

En todo sistema de producción las necesidades de los clientes se traducen en órdenes de producción que se liberan y "transforman" en trabajos con fecha de entrega asociada. La programación de producción que asigna estos trabajos a recursos productivos limitados, debe realizarse de manera detallada y eficiente para permitir un mejor control de las operaciones dentro del sistema productivo y constituir una ventaja competitiva difícil de imitar.

Los diferentes productos requieren en su fabricación distintas operaciones, las cuales se realizan en un orden y configuración productiva determinada, que depende del tipo de producto, el volumen de producción, la variedad de productos que se producen en el sistema, etcétera.

El *taller de máquinas paralelas* consiste en un sistema de *m* máquinas dispuestas en paralelo (multicapacidad), que procesan trabajos que requieren una operación, la cual puede realizarse en cualesquiera de las *m* máquinas.

Para resolver la programación de un *taller de máquinas paralelas* existen diferentes métodos exactos y heurísticos, constructivos o de mejora. La *heurística LPT* (Baker, 1974; Baker y Trietsch, 2009; Blazewicz *et al*., 1996; Pinedo, 2008) es una de las heurísticas clásicas de buen desempeño que resuelve el problema sin tiempos de preparación dependientes de la secuencia y minimización de *makespan*.

Esta heurística puede adaptarse para resolver el caso de tiempos de preparación dependientes de la secuencia (ver estudio experimental).

Se han tratado diferentes problemas de *máquinas paralelas idénticas* con *setup* aplicando diferentes metaheurísticas y métodos de búsqueda local. Mendes *et al.* (2002) plantean un algoritmo que combina *tabu search* y *algoritmos meméticos* para minimizar C_{max}. Radhakrishnan y Ventura (2000) tratan el problema de minimizar la suma total de la prontitud y tardanza a través de un algoritmo *simulated annealing* que incorpora una *heurística de búsqueda local* para construir la solución inicial. Lin y Liao (2004) tratan el problema de minimizar C_{max} sujeto al tiempo mínimo total de flujo basado en un método de programación matemática.

Anglani *et al*. (2005) plantean un modelo *fuzzy* de programación matemática que considera la imprecisión en los tiempos de proceso, balanceando los costos de *setup* con la satisfacción de la demanda. Behnamian *et al.* (2009) proponen un algoritmo híbrido para minimizar C_{max} utilizando *ant colony optimization, simulated annealing y variable neighborhood search*.

El taller de máquinas paralelas

En este trabajo se trata el problema del *taller de máquinas paralelas idénticas con tiempos de preparación dependientes de la secuencia*, que consiste en resolver la programación de trabajos en un sistema de capacidad múltiple con *m* máquinas que realizan operaciones iguales, dispuestas en paralelo y *n* trabajos a procesar en una, y sólo una, de las máquinas. El concepto de máquinas idénticas significa que cada trabajo puede ser procesado en cada una de las máquinas con igual tiempo de proceso. El tiempo de preparación en el que se incurre al procesar un trabajo en una máquina depende del trabajo previamente procesado en la misma. Este tipo de configuración está presente en diferentes ambientes de manufactura como en la industria textil, industria de la madera, etcétera.

El tiempo de proceso de cada trabajo está fijo y existen tiempos de preparación de máquinas que dependen del orden en el que se procesan los trabajos en cada una. El objetivo considerado en este trabajo es minimizar el *makespan* (C_{max}), que consiste en minimizar el intervalo de tiempo entre el inicio del procesamiento del primer trabajo (tiempo de referencia 0) y el tiempo de terminación del procesamiento del último trabajo, es decir, el intervalo de tiempo en el que se procesa completamente la totalidad de los trabajos (órdenes de producción). Se consideran los siguientes supuestos:

1. Cada trabajo debe ser procesado en una, y sólo una, máquina k, $k = 1, 2,..., m$.
2. El tiempo de proceso del trabajo i, independiente de la máquina, está dado por p_i ($i = 1,..., n$).
3. Los tiempos de preparación (*setup*) para procesar el trabajo j después del trabajo i, independiente de la máquina, está dado por s_{ij} ($i = 1,..., n$; $j = 1,\ldots, n$), donde s_{ii} representa la preparación inicial cuando el trabajo i es el primer trabajo procesado en una máquina.
4. Cada máquina puede procesar sólo un trabajo a la vez.
5. El proceso de un trabajo en una máquina no se puede interrumpir (*nonpreemption*).
6. Todos los trabajos son independientes entre sí y se encuentran disponibles en el instante inicial.
7. Las máquinas operan sin fallas en el horizonte de programación.
8. El objetivo es minimizar C_{max}.

Bajo la notación introducida por Graham *et al*. (1979) el problema de máquinas paralelas caracterizado por los supuestos mencionados se denota por $Pm / s_{ij} / C_{max}$, y es un conocido problema NP-Hard (Blazewicz *et al*.,

1996; Pinedo, 2008), lo que hace impracticable la obtención de la solución óptima para problemas de mediano a gran tamaño.

En este trabajo se resuelve el problema Pm / s_{ij} / C_{max} mediante un *algoritmo genético*, comparándolo con otras heurísticas. 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 (Allahverdi *et al*., 2008).

Un esquema simple para resolver la programación para este problema es definir un ordenamiento de los trabajos de acuerdo a un criterio determinado, y luego continuar de acuerdo al procedimiento de asignación de trabajos presentado en la figura 1.

procedure *Asignación de Tabajos*
Definir ListaTrabajos ordenada por un criterio.
while *(ListaTrabajos no vacía)* ***do***
Asignar primer trabajo de la lista a la máquina donde finaliza antes.
Eliminar primer trabajo de ListaTrabajos.
endwhile
endprocedure

Figura 1. Pseudocódigo de procedimiento de asignación de trabajos a máquinas

A modo de ilustración, consideremos un problema de 2 máquinas paralelas idénticas y 6 trabajos a programar, cuyos parámetros se presentan en las tablas 1 y 2.

Tabla 1. Tiempos de proceso de los trabajos (p_i)

Trabajo	1	2	3	4	5	6
p_i	9	12	7	15	10	8

Tabla 2. Matriz de tiempos de *setup* (s_{ij})

Trabajo	1	2	3	4	5	6
1	3	8	6	7	2	4
2	9	6	5	4	2	7
3	6	7	4	8	4	9
4	7	8	7	4	6	6
5	8	4	3	9	2	3
6	6	5	4	9	3	2

Utilizaremos la regla LPT (*largest processing time*) para generar la *ListaTrabajos* inicial del procedimiento de *Asignación de Trabajos* de la figura 1. La lista obtenida ordena los trabajos de mayor a menor tiempo de proceso, resultando: 4 – 2 – 5 – 1 – 6 – 3, para luego asignarlos a las máquinas de acuerdo al procedimiento de la figura 1. Este procedimiento de asignación genera la programación que se muestra en la carta Gantt de la figura 2.

Para el trabajo 4, tanto en la máquina 1 (M1) como en la máquina 2 (M2) el tiempo de terminación es 0 (disponibilidad de máquina) + 4 (setup inicial) + 15 (tiempo de proceso) = 19; se asigna a M1 utilizando el criterio de desempate al asignar a la máquina de menor índice. Luego el trabajo 2 se asigna a M2 terminando en el tiempo 0 + 6 + 12 = 18 (en M1 habría finalizado en el tiempo 19 + 8 + 12 = 39). A continuación, el trabajo 5 se asigna a M2 con tiempo de terminación 18 + 2 + 10 = 30 (en M1 habría finalizado en el tiempo 19 + 6 + 10 = 35) y así hasta obtener una programación con C_{max} = 48 (el trabajo 3, último trabajo que se procesa, termina en el tiempo 48).

Algoritmo genético

Los *algoritmos genéticos* fueron introducidos por Holland (1975), utilizan un lenguaje de genética natural modelando, en forma artificial, mecanismos de la evolución natural aplicados a la optimización de problemas. Asocian el concepto de *individuo* a una solución factible del problema y el de *población* a un conjunto de *individuos* (soluciones factibles). Los *individuos* están formados por *genes* (elementos ordenados en una sucesión lineal), que se evalúan a través de una función de aptitud denominada *fitness*, que corresponde a una medida de la calidad del *individuo* como solución del problema (Goldberg, 1989; Davis, 1991 y Michalewicz, 1999). Otros enfoques evolutivos actuales e inspirados en procesos de la naturaleza, extendidos también a más

procedure *Algoritmo Genético*
$t \leftarrow 0$
inicializar P_t
evaluar P_t
while $(t < N_g)$ ***do***
$t \leftarrow t+1$
seleccionar padres de P_{t-1}
formar población P_t
evaluar P_t
endwhile
endprocedure

Figura 3. Pseudocódigo de un algoritmo genético (Michalewicz, 1999)

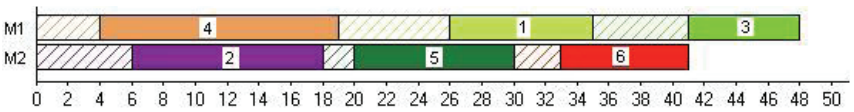

Figura 2. Carta Gantt–Procedimiento de asignación

de un objetivo pueden revisarse en Coello *et al*. (2010). La figura 3 muestra una estructura general de un algoritmo genético.

La población inicial de soluciones (P_0) se determina de manera aleatoria y el proceso de selección durante el proceso evolutivo se realiza de acuerdo con una distribución de probabilidades, que determina que un individuo tiene una probabilidad de ser seleccionado proporcional a su *fitness* (Michalewicz, 1999). Esto significa que en el proceso de selección los individuos de mejor *fitness* tienen mayor probabilidad de ser seleccionados.

La formación de la población de la generación t (P_t), a partir de la población de la generación $t-1$ (P_{t-1}), se realiza de acuerdo con un proceso de cruzamiento de dos individuos, sujeto a una probabilidad de cruzamiento (p_c), generando descendencia (hijos). Sobre la descendencia opera una mutación de acuerdo con una probabilidad de mutación (P_m). La evaluación de la población P_t evalúa a cada individuo determinando su *fitness*. El proceso evolutivo utiliza un operador de cruzamiento y un operador de mutación, y se realiza hasta que se evalúan Ng generaciones.

El *algoritmo genético* definido utiliza *individuos* con estructura de *cromosoma* de *n* elementos, que representa una lista ordenada de los trabajos a programar. El *individuo* es evaluado obteniendo el *makespan* de la asignación de trabajos a las máquinas de acuerdo al procedimiento indicado en la figura 1. Se utilizó esta estructura del cromosoma, dado que el *algoritmo genético* se comparó con otras heurísticas que generan la programación basada en una secuencia de trabajos; el mismo argumento anterior explica también el por qué tampoco se adoptó incluir elitismo en el proceso de evolución.

Para el problema de programar un conjunto de 6 trabajos, como en el ejemplo de la sección 2, la secuencia: 4 – 2 – 5 – 1 – 6 – 3 representa un individuo, que define una lista ordenada de trabajos asignados a las 2 máquinas del ejemplo, siguiendo el procedimiento de la figura 1, lo que produce la programación con C_{max} = 48 presentada en la figura 2. El valor del *makespan* igual a 48 que origina la mencionada secuencia corresponde al valor del individuo. Como se trata de un problema de minimización, un individuo representa una mejor solución mientras este valor sea menor (individuo de mejor *fitness*).

Se utilizan los operadores genéticos *PMX* propuestos por Goldberg y Lingle (1985) (Michalewicz, 1999) como operador de cruzamiento y *swap* (o *exchange*) como operador de mutación. Estos operadores son clásicos y frecuentemente se utilizan en la literatura en problemas de secuenciación. El operador *PMX* selecciona de manera aleatoria dos posiciones copiando la subsecuencia central de dos padres en dos descendientes (hijos). Las posiciones restantes de los hijos se llenan con los trabajos aún no asignados en la misma posición del padre que no aportó la subsecuencia central al hijo, esto es, si el trabajo no se encuentra en la subsecuencia central permanece en la misma posición, en caso contrario, se reemplaza por el trabajo que está en la misma posición de la subsecuencia central traspasada al otro hijo. El operador *swap* intercambia dos trabajos de un individuo en forma aleatoria.

Padre 1 :	5 – 2 | 3 – 1 – 6 | 4	(58)	Padre 2 :	4 – 6 | 1 – 5 – 2 | 3	(45)
	x – x | 1 – 5 – 2 | 4			4 – x | 3 – 1 – 6 | x	
Hijo 1 :	3 – 6 | 1 – 5 – 2 | 4	(47)	Hijo 2 :	4 – 2 | 3 – 1 – 6 | 5	(47)

Figura 4. Operador de Cruzamiento PMX

Para el ejemplo de la sección 2, la figura 4 muestra el cruzamiento de dos individuos (padres) de una población con *makespan* 58 y 45, respectivamente, aplicando el operador de cruzamiento *PMX* seleccionando en forma aleatoria la subsecuencia que incluye las posiciones 3 a 5, proceso que genera dos hijos, ambos con *makespan* de 47.

Para el mismo ejemplo, la figura 5 muestra la mutación del hijo 1 intercambiando las posiciones 2 y 6, proceso que genera el mutante con *makespan* 46.

Individuo : 3 – **6** – 1 – 5 – 2 – **4** (47)

Mutante : 3 – **4** – 1 – 5 – 2 – **6** (46)

Figura 5. Operador de mutación *swap*

Posteriormente, se introduce un algoritmo de mejora que optimiza la asignación de los trabajos en cada máquina. El algoritmo de mejora que se introduce en la evaluación de cada individuo busca reducir los tiempos de terminación de cada máquina del sistema, para lograr una disminución del *makespan*. La reducción del tiempo de terminación en cada máquina se trata resolviendo problemas independientes de secuenciamiento de trabajos en una máquina con tiempos de preparación dependientes de la secuencia.

Este problema tiene estructura similar a una variante del problema del vendedor viajero asimétrico ATSP (*asymmetric traveling salesman problem*), en el que un vendedor que debe visitar *n* ciudades con distancias asimétricas entre ellas (esto es $d_{ij} \neq d_{ji}$), minimizando el total de la distancia recorrida sin retornar a la ciudad de origen.

Los trabajos se asocian a las ciudades y los tiempos de *setup* s_{ij} se asocian a las distancias d_{ij} (note que $d_{ij} \neq d_{ji}$

```
procedure Heurística del Mejor Vecino
    Definir ListaTrabajos
    while (ListaTrabajos no vacía) do
     Asignar primer trabajo de ListaTrabajos como primer trabajo de Secuencia
     Definir ListaSecuencia (todos los trabajos salvo primer trabajo de la Secuencia).
        while (ListaSecuencia no vacía) do
            Asignar trabajo de ListaSecuncia que genere menor setup a continuación.
            Eliminar trabajo asignado  de ListaSecuencia.
        endwhile
        Evaluar Secuencia.
        Eliminar primer trabajo de ListaTrabajos.
    endwhile
endprocedure
```

Figura 6. Pseudocódigo Procedimiento Heurística del Mejor Vecino (MV)

dado que se tienen tiempos de preparación dependientes de la secuencia $s_{ij} \neq s_{ji}$), así, minimizar la distancia total recorrida por el vendedor se asocia con minimizar la suma total de *setups*, que es equivalente a minimizar C_{max}.

Por lo anterior, se aplica para su resolución una adaptación de una heurística *greedy* clásica aplicada al problema del vendedor viajero, denominada *heurística del mejor vecino* (*nearest neighbor heuristic*), que construye una secuencia de ciudades a visitar, de manera que estando en una ciudad, el vendedor escoge la ciudad más cercana aún no visitada. El procedimiento aplicado en este trabajo se indica en la figura 6.

Para el ejemplo de la sección 2, el individuo 4 – 2 – 5 – 1 – 6 – 3 fue asignado mediante el procedimiento de asignación de la figura 1, obteniéndose una programación con $C_{max} = 48$ (figura 2). A M1 se asigna la secuencia de trabajos 4 – 1 – 3, mientras que a M2 se asigna la secuencia de trabajos 2 – 5 – 6, lo que produce tiempos de terminación de 48 y 41 en M1 y M2, respectivamente.

Así, para completar la evaluación del individuo 4 – 2 – 5 – 1 – 6 – 3 se resuelven dos problemas de secuenciamiento de una máquina, uno (en M1) que considera sólo los trabajos 1, 3 y 4, y otro (en M2) que considera sólo los trabajos 2, 5 y 6. Resolviendo estos problemas mediante la heurística del mejor vecino se obtienen nuevos ordenamientos de trabajos en M1 y M2, como se ilustra en la figura 7.

La secuencia de trabajos 3 – 1 – 4 en M1, finaliza en M1 con tiempo 44, mientras que la secuencia de trabajos 6 – 5 – 2, finaliza en M2 con tiempo 39, por lo que el *makespan* resultante es 44, mejorando en este caso el *makespan* obtenido por el procedimiento de asignación.

Estudio experimental

Los métodos evaluados fueron los algoritmos genéticos AG (estándar con operador de cruzamiento *PMX* y operador de mutación *swap*) y AG+MV (AG incorporando la *heurística del mejor vecino* como algoritmo de mejora) presentados en la sección 3, LPT* (extensión de la *Heurística LPT* al problema de máquinas paralelas idénticas con *setup*) y SMC (simulación Montecarlo).

Los parámetros de probabilidad de cruzamiento (p_c) y mutación (p_m), y el tamaño de población (N_p) del algoritmo genético se calibraron mediante experimentación. En un estudio preliminar se analizaron 5 instancias del problema (distintas a las consideradas en el trabajo) con valores $p_c = 0.5$, 0.6 y 0.8; $p_m = 0.10$, 0.20 y 0.50; $N_p =$ 50, 100 y 200 procesando 10 réplicas de 200 generaciones por instancia. Se observó el *makespan* promedio para cada combinación de valores en cada instancia, obteniéndose con mayor frecuencia el menor promedio para los valores $p_c = 0.8$, $p_m = 0.5$ y $N_p = 200$. Finalmente, el número de generaciones para la experimentación se estableció en 500 para posibilitar una búsqueda exhaustiva y se procesaron 10 réplicas para cada instancia entregando como resultado del método la secuencia con menor *makespan*. Estos parámetros se utilizaron tanto para el algoritmo genético básico AG, como para la versión del algoritmo genético con mejora AG+MV.

Heurística LPT*

La *heurística LPT,* aplicada con buenos resultados al problema de máquinas paralelas sin *setup* (Baker, 1974; Baker y Trietsch, 2009), se extiende en este trabajo al

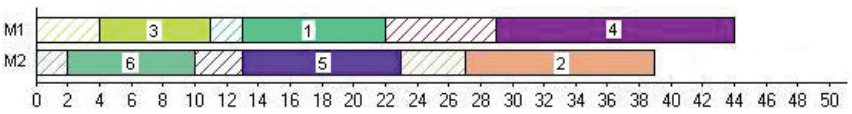

Figura 7. Carta Gantt–asignación con mejoramiento del mejor vecino

caso de máquinas paralelas idénticas con *setup*, denominándola *heurística LPT**. La heurística LPT* redefine los tiempos de proceso de cada trabajo *i* estimando su tiempo de ocupación de máquina (pe_i) como la suma de su tiempo de proceso y el promedio de los tiempos de *setup* que puede tener cada trabajo, esto es:

$$pe_i = p_i + \sum_{j=1}^{n} s_{ji} \Big/ n$$

En lugar de ordenar los trabajos de mayor a menor tiempo de proceso p_i como lo hace la heurística LPT, los trabajos se ordenan de mayor a menor tiempo estimado de ocupación de máquina pe_i, formando así la *ListaTrabajos* inicial para el procedimiento de *Asignación de Trabajos* de la figura 1.

Simulación Montecarlo

El método de simulación Montecarlo genera en forma (pseudo) aleatoria una muestra de tamaño N de secuencias de *n* trabajos; cada una de las N secuencias de la muestra se construye seleccionando con igual probabilidad el siguiente trabajo de entre los trabajos aún no seleccionados mediante un procedimiento de muestreo (pseudo) aleatorio. Cada secuencia de la muestra se evalúa mediante el procedimiento de asignación de trabajos de la figura 1, entregando como resultado del procedimiento de muestreo la secuencia de la muestra con menor *makespan*. En este trabajo se consideraron 10 réplicas (muestras) de tamaño N = 50.000 secuencias, cada una de *n* = 50 trabajos, entregando como solución la secuencia de menor *makespan* entre las 10 secuencias entregada por cada réplica.

La evaluación de los algoritmos se realizó utilizando instancias generadas en forma aleatoria de acuerdo a distribuciones de tiempos de proceso y de *setup* utilizadas en la literatura. Los tiempos de proceso de los trabajos se generaron de la distribución uniforme discreta entre 1 y 100 ($p_i \sim UD[1,100]$). Los tiempos de preparación dependientes de la secuencia de los trabajos se generaron de la distribución uniforme discreta entre 1 y 30 ($s_{ij} \sim UD[1,30]$). Se considera, por lo tanto, un entorno productivo donde los tiempos de preparación son esencialmente menores a los tiempos de proceso.

Se generaron 30 instancias de problemas de 5 máquinas (*m* = 5) y 50 trabajos (*n* = 50), esperando en promedio la asignación de 10 trabajos por máquina.

La evaluación de la heurística se realizó por medio de rutinas adaptadas del software SPS_Optimizer (Salazar, 2010), herramienta diseñada para la programación de operaciones.

Para evaluar el desempeño se utilizó el *makespan* (C_{max}) como medida de desempeño. El *makespan* entregado por el método se compara contra la mejor solución conocida (MSC), correspondiente a la mejor solución obtenida entre todos los métodos. Para determinar el porcentaje de incremento sobre el mejor *makespan* conocido (%MSC), se utiliza la medida:

$$\%MSC = \frac{Sol_{Método} - MSC}{MSC} * 100$$

$Sol_{Método}$ es el valor del makespan obtenido con el método en estudio y MSC es la mejor solución conocida de la respectiva instancia.

Los resultados del *makespan* obtenidos por cada algoritmo en cada instancia se muestran en la tabla 4 (columnas GA, LPT*, SMC y AG+MV). En la figura 8 se grafican estos resultados, en el eje horizontal se tienen las 30 instancias y en el eje vertical el valor del *makespan*.

En éste se aprecia que el algoritmo LPT* presenta el desempeño más bajo (prácticamente en todos los

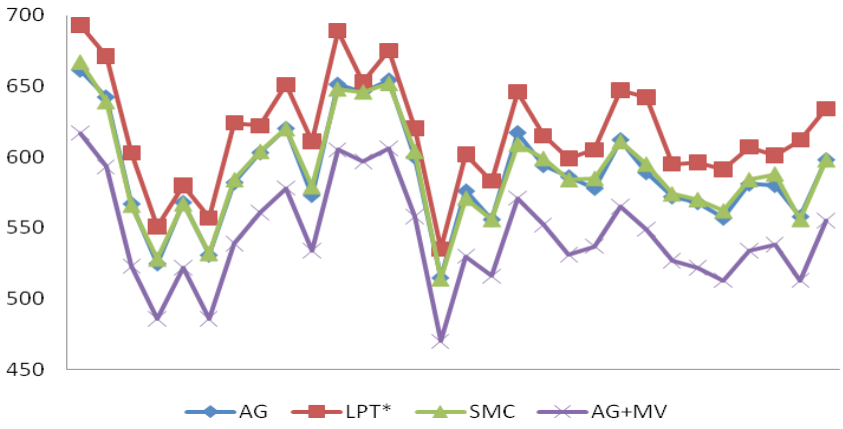

Figura 8. Comparación de métodos

problemas su gráfica está por sobre la de todos los otros métodos), mientras que los algoritmos AG y SMC presentan un rendimiento muy similar (las soluciones de estos métodos prácticamente se superponen). De la tabla 4 se obtiene que GA es mejor en el 63.33% (19 sobre 30) y LPT* en 50% (15 sobre 30) de las instancias, existiendo 4 instancias en las cuales estos algoritmos obtienen igual resultado.

Lo que se destaca en la figura 8 es el buen rendimiento que logra AG+MV, mejorando significativamente su desempeño respecto de AG, obteniendo la mejor solución en 100% de las instancias evaluadas. Por este motivo se realiza la comparación de todos los algoritmos con respecto a las soluciones obtenidas por AG+MV, detallando en la tabla 4 las diferencias porcentuales %MSC por instancia para cada método (columnas %AG, %LPT* y %SMC).

La tabla 3 resume las diferencias porcentuales promedio de los algoritmos AG, LPT* y SMC con respecto a AG+MV, lo que se interpreta de la siguiente forma: en promedio las soluciones del algoritmo AG+MV son, respectivamente, 8.18%, 14.41% y 8.39% mejores soluciones que los algoritmos AG, LPT* y SMC.

Tabla 3. Comparación de algoritmos

Algoritmo	AG	LPT*	SMC
%MSC promedio	8.18	13.41	8.39

También en la tabla 3 se verifica el similar comportamiento promedio entre los algoritmos AG y SMC con una leve ventaja de AG sobre SMC, presentando también poca variabilidad. La última columna de la tabla 4 (columna %AG/SMC) muestra la desviación porcentual de la diferencia absoluta entre AG y SMC por instancia, la que en promedio indica una desviación de 0.55% con una máxima diferencia de 1.49% en una instancia.

El procesamiento se realizó en un computador Intel Core 2 Duo T7500 de 2.2 GHz de 2 GB de RAM. El orden de magnitud del tiempo CPU para la ejecución de los algoritmos se presenta en la tabla 5. Los resultados presentados se obtuvieron con base en la ejecución de 10 réplicas para los algoritmos AG (15 s/réplica), SMC (3 s/réplica) y AG+MV (60 s/réplica).

Tabla 5. Tiempo CPU de algoritmos

Algoritmo	AG	LPT*	SMC	GA+MV
CPU [s]	150	0,00	30	600

Tabla 4. Resultados de algoritmos por instancia

Instancia	AG	LPT*	SMC	AG+MV	%AG	%LPT*	%SMC	%AG/SMC
1	661	693	667	617	7,13	12,32	8,10	0,97
2	642	671	639	593	8,26	13,15	7,76	0,51
3	567	603	566	523	8,41	15,30	8,22	0,19
4	525	551	528	486	8,02	13,37	8,64	0,62
5	568	580	567	522	8,81	11,11	8,62	0,19
6	531	557	532	486	9,26	14,61	9,47	0,21
7	582	624	584	539	7,98	15,77	8,35	0,37
8	603	622	604	561	7,49	10,87	7,66	0,18
9	620	651	620	578	7,27	12,63	7,27	0,00
10	573	611	579	534	7,30	14,42	8,43	1,12
11	651	689	648	605	7,60	13,88	7,11	0,50
12	646	653	646	597	8,21	9,38	8,21	0,00
13	654	675	652	606	7,92	11,39	7,59	0,33
14	600	620	604	558	7,53	11,11	8,24	0,72
15	515	535	514	470	9,57	13,83	9,36	0,21
16	576	602	571	530	8,68	13,58	7,74	0,94
17	556	583	556	516	7,75	12,98	7,75	0,00
18	617	646	609	571	8,06	13,13	6,65	1,40
19	594	615	599	552	7,61	11,41	8,51	0,91
20	586	599	584	531	10,36	12,81	9,98	0,38
21	578	605	585	537	7,64	12,66	8,94	1,30
22	612	647	611	565	8,32	14,51	8,14	0,18
23	589	642	595	549	7,29	16,94	8,38	1,09
24	572	595	574	527	8,54	12,90	8,92	0,38
25	568	596	570	522	8,81	14,18	9,20	0,38
26	557	591	562	513	8,58	15,20	9,55	0,97
27	581	607	584	534	8,80	13,67	9,36	0,56
28	580	601	588	538	7,81	11,71	9,29	1,49
29	558	612	556	513	8,77	19,30	8,38	0,39
30	598	634	598	555	7,75	14,23	7,75	0,00
				Promedio	8,18	13,41	8,39	0,55

La incorporación del algoritmo de mejora en el algoritmo genético estándar, mejora en torno a 8% la calidad de las soluciones obtenidas, cuadruplicando el costo en tiempo CPU. Los algoritmos AG y SMC obtienen similares resultados, pero este último lo hace en un quinto del tiempo del primero. Por otro lado, el tiempo computacional del algoritmo LPT* es prácticamente despreciable obteniendo soluciones que son sólo alrededor de 5% peor que los algoritmos AG y SMC, y se alejan en promedio aproximadamente un 13.5% del algoritmo AG+MV.

Conclusiones

Se utilizó un algoritmo genético estándar para resolver el problema de programación de trabajos en el sistema de máquinas paralelas idénticas con tiempos de preparación dependientes de la secuencia, al que se le introdujo la *heurística del mejor vecino* como algoritmo de mejora, que optimiza la asignación de los trabajos resolviendo un problema independiente de minimización de *makespan* en cada máquina.

Al introducir el algoritmo de mejora, el aumento en el desempeño del algoritmo genético es significativo, mejorando en promedio las soluciones obtenidas con el algoritmo genético estándar en cerca de un 8%, superando a todos los otros algoritmos en todas las instancias; sin embargo, su tiempo computacional es 4 veces mayor.

El algoritmo genético estándar y la simulación Montecarlo presentan un rendimiento similar, pero no una ventaja significativa por parte del algoritmo genético, ya que este último utiliza 5 veces más de tiempo computacional que la *simulación Montecarlo*.

El algoritmo LPT* es el algoritmo que presenta el rendimiento más bajo entre los algoritmos considerados, siendo el que obtiene la peor solución en todas las instancias evaluadas, sin embargo, su tiempo computacional es prácticamente despreciable, alejándose sólo alrededor de 5% del algoritmo genético estándar y simulación Montecarlo, y cerca de 13.5% del algoritmo genético con la mejora.

Para un trabajo futuro, el enfoque del procedimiento AG+MV puede mejorarse en varios aspectos, por un lado, respecto a la composición del algoritmo genético es posible explorar el efecto que tendría que considerar cromosomas de diferente estructura, como también la inclusión del concepto de elitismo; por otro lado, tanto para el procedimiento de búsqueda (AG) como para el procedimiento de mejora (MV) utilizados en este trabajo, es posible considerar otras metaheurísticas, pudiendo ser extendido también al caso de máquinas paralelas no idénticas.

Referencias

Allahverdi A., Ng C.T., Cheng T.C.E., Kovalyov M. A Survey of Scheduling Problems with Setup Times or Costs. *European Journal of Operational Research*, volumen 187 (número 3), 2008: 985-1032.

Anglani A., Grieco A., Gerriero E., Musmanno R. Robust Scheduling of Parallel Machines with Sequence-Dependent Set-up Costs. *European Journal of Operational Research*, volumen 161 (número 3), 2005: 704-720.

Baker K.R., Trietsch D. *Principles of Sequencing and Scheduling*, New York, John Wiley and Sons, 2009.

Baker K.R. *Introduction to Sequencing and Scheduling*, New York, John Wiley and Sons, 1974.

Blazewicz J., Ecker K., Pesch E. Schmidt G., Weglarz J. *Scheduling Computer and Manufacturing Processes*, Springer, 1996.

Coello C., Dhaenens C., Jourdan L. *Advances in Multi-Objective Nature Inspired Computing*, Springer, Berlín/Heidelberg, 2010.

Davis L. *Handbook of Genetic Algorithms*, New York, Van Nostrand Reinhold, 1991.

Behnamian J., Zandieh M., Fatemi-Ghomi S.M.T. Parallel-Machine Scheduling Problems with Sequence-Dependent Setup Times Using an ACO, SA and VNS Hybrid Algorithm. *Expert Systems with Applications,* volumen 36 (número 6), 2009: 9637-9644.

Goldberg D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*, Massaschussets, Addison Wesley, Reading, 1989.

Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy-Kan A.H.G. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. *Annals of Discrete Mathematics*, volumen 5, 1979: 287-326.

Holland J.H. *Adaptation in Natural and Artificial Systems*, Michigan, University of Michigan Press, Ann Arbor, 1975.

Lin C.H. y Liao C.J. Makespan Minimization Subject to Flowtime Optimality on Identical Parallel Machines. *Computers & Operations Research*, volumen 31 (número 10), 2004: 1655-1666.

Mendes A., Müller F., Franca P., Moscato P. Comparing Meta-Heurístic Approaches for Parallel Machine Scheduling Problems with Sequence Dependent Setup Times. *Production Planning and Control*, volumen 13 (número 2), 2002: 143-154.

Michalewicz Z. *Genetic Algorithms+Data Structures=Evolution Programs*, 3th ed., Springer, 1999.

Pinedo M. *Scheduling–Theory, Algorithms and Systems*, 3th ed., Springer, 2008.

Radhakrishnan S., Ventura J. Simulated Annealing for Parallel Machine Scheduling with Earliness-Tardiness Penalties and Sequence-Dependent Set-Up Times. *International Journal of Production Research*, volumen 38 (número 10), 2000: 2233-2252.

Salazar E. Programación de sistemas de producción con SPS_Optimizer. *Revista ICHIO*, volumen 1 (número 2), 2010: 33-46.

Este artículo se cita:

Citación Chicago

Salazar-Hornig Eduardo, Juan Carlo Medina-S. Minimización del *makespan* en máquinas paralelas idénticas con tiempos de preparación dependientes de la secuencia utilizando un algoritmo genético. *Ingeniería Investigación y Tecnología* XIV, 01 (2013): 43-51.

Citación ISO 690

Salazar-Hornig E., Medina S.J.C. Minimización del *makespan* en máquinas paralelas idénticas con tiempos de preparación dependientes de la secuencia utilizando un algoritmo genético. *Ingeniería Investigación y Tecnología*, volumen XIV (número 1), enero-marzo 2013: 43-51.

Semblanza de los autores

Eduardo Salazar-Hornig. Es ingeniero matemático por la Universidad de Concepción (1984), obtuvo el grado de magíster en investigación de operaciones en la RWTH University of Aachen, Alemania en 1992. Su línea de investigación incluye sistemas de producción, planificación y programación de producción y simulación. Es profesor de tiempo completo en el Departamento de Ingeniería Industrial y Programa de Magíster en Ingeniería Industrial de la Universidad de Concepción, Chile.

Juan Carlo Medina-S. Es ingeniero civil industrial por la Universidad Católica de la Santísima Concepción (2006) y candidato a magíster del programa de magíster en ingeniería industrial de la Universidad de Concepción. Actualmente se desempeña como profesor de la Universidad de las Américas en Santiago de Chile.