Ingeniare. Revista Chilena de Ingeniería
ISSN: 0718-3291
facing@uta.cl
Universidad de Tarapacá
Chile

Salazar Hornig, Eduardo; Ávila Thieme, Claudia
Heurística GRASP para la minimización del makespan en máquinas paralelas no relacionadas con tiempos de preparación dependientes de la secuencia
Ingeniare. Revista Chilena de Ingeniería, vol. 25, núm. 3, septiembre, 2017, pp. 524-534
Universidad de Tarapacá
Arica, Chile

Disponible en: http://www.redalyc.org/articulo.oa?id=77252700014

- Cómo citar el artículo
- Número completo
- Más información del artículo
- Página de la revista en redalyc.org

redalyc.org
Sistema de Información Científica
Red de Revistas Científicas de América Latina, el Caribe, España y Portugal
Proyecto académico sin fines de lucro, desarrollado bajo la iniciativa de acceso abierto

Heurística *GRASP* para la minimización del *makespan* en máquinas paralelas no relacionadas con tiempos de preparación dependientes de la secuencia

GRASP *approach for the unrelated parallel machines scheduling problem with* makespan *minimization and sequence dependent setup times*

Eduardo Salazar Hornig[1]* Claudia Ávila Thieme[1]

Recibido 7 de marzo de 2016, aceptado 26 de julio de 2016
Received: March 7, 2016 Accepted: July 26, 2016

RESUMEN

Se propone un algoritmo *GRASP* (*Greedy Randomized Adaptative Search Procedures*) para resolver el problema de la programación de trabajos en un sistema de máquinas paralelas no relacionadas con tiempos de preparación dependientes de la secuencia y minimización del *makespan*. Se evalúan cuatro procedimientos en la fase de búsqueda local de *GRASP* utilizando una representación secuencial y matricial de las soluciones. La efectividad y eficiencia de las alternativas propuestas se comparan con otras heurísticas de la literature sobre un conjunto de problemas de prueba, superando uno de los procedimientos propuestos el rendimiento promedio.

Palabras clave: *GRASP*, máquinas paralelas no relacionadas, *makespan*, tiempos de preparación dependientes de la secuencia.

ABSTRACT

A GRASP (Greedy Randomized Adaptive Search Procedures) *algorithm to solve the unrelated parallel machines scheduling problem with sequence dependent setup times and* makespan *minimization is proposed. Four neighborhoods in the local search phase of the* GRASP *algorithm using a sequence and matrix representation of solutions, are proposed. The effectiveness and efficiency of the proposed heuristics are compared on a benchmark problem dataset with other metaheuristics of the literature used in earlier studies. One of the proposed heuristics outperforms the average makespan.*

Keywords: GRASP*, unrelated parallel machines, makespan, sequence dependent setup times.*

INTRODUCCIÓN

En este estudio se considera el problema de la programación de *n* trabajos en un sistema de *m* máquinas paralelas no relacionadas con tiempos de *setup* dependientes de la secuencia y minimización del *makespan*. En un problema de programación de trabajos, el *makespan* corresponde al intervalo de tiempo en el que se procesan todos los trabajos a programar. Disponer de recursos productivos en paralelo otorga flexibilidad para el procesamiento de trabajos y permite mejorar la capacidad de respuesta de un sistema productivo. Por otro lado, las actividades de preparación de máquinas (tiempos de *setup*) que usualmente son requeridas cuando se pasa del proceso de un trabajo al siguiente, pueden

[1] Departamento de Ingeniería Industrial. Universidad de Concepción. Casilla 160-C. Concepción, Chile. E-mail: esalazar@udec.cl; deisytorres@udec.cl
* Autor de correspondencia.

llegar a ser un problema crítico en la programación de trabajos afectando la productividad del sistema, en particular, cuando estos son tiempos que dependen de la secuencia de proceso. Problemas de este tipo son comunes en la industria de la producción de pintura y de plástico (donde se requiere una completa limpieza entre la producción de una orden de producción y otra), como también en la industria de la producción de textiles, de vidrios, de químicos, de papel, y de servicios [1].

La importancia de considerar los tiempos de *setup* dependientes de la secuencia en problemas de programación de producción se ha hecho notoria en las últimas décadas [2-3].

El trabajo [4] fue uno de los primeros en abordar el problema de máquinas paralelas no relacionadas con tiempos de *setup* dependientes de la secuencia y minimización del *makespan*, proponiendo la heurística *PH* (heurística de partición). Luego, en [1] se propone para su resolución un algoritmo de búsqueda tabú que usa dos fases en los esquemas de perturbación, uno que actúa dentro de una sola máquina y otro que lo hace entre las máquinas, superando los resultados obtenidos por la heurística *PH*.

En [5] el problema se resuelve utilizando una metaheurística para la búsqueda aleatoria priorizada llamada *Meta-Raps*, que es una estrategia que usa tanto una heurística de construcción como una heurística de mejora para generar soluciones de alta calidad, mostrando también mejores resultados que la heurística *PH*. Más adelante, Arnaout, Rabadi y Musa [11] introducen un algoritmo *ACO* de optimización de colonia de hormigas (denominado *ACO I*) de dos etapas: la asignación de trabajos y la secuenciación de los trabajos asignados. El algoritmo muestra superioridad al comparar instancias del problema con búsqueda tabú, *Meta-Raps* y la heurística *PH*.

En [6] los autores proponen y evalúan un algoritmo genético que incluye una búsqueda local limitada de mejora en el operador de cruzamiento, una búsqueda local rápida basada en una vecindad de inserción y la aceptación de movimientos durante la búsqueda local. Mientras que, en [7] se desarrolla un conjunto de propiedades dominantes como condiciones necesarias en el orden de la secuencia de trabajos en el programa óptimo e introducen un algoritmo genético híbrido derivado de estas propiedades.

En [8] se presenta un algoritmo de recocido simulado (denominado *RSA*) que incorpora una estrategia de búsqueda restringida para minimizar el *makespan*, El algoritmo reduce el esfuerzo computacional para encontrar la mejor solución en la vecindad eliminando movimientos no efectivos de trabajos. La eficiencia y efectividad de este algoritmo es comparada con un algoritmo de recocido simulado básico y metaheurísticas existentes en la literatura. En [9] se propone un algoritmo de búsqueda en vecindad variable descendente hibridizado con elementos de programación matemática, mostrando un elevado desempeño. Finalmente, [10] introduce una mejora al algoritmo *ACO I* [11], denominado *ACO II*, para la minimización del *makespan*. Entre los cambios se incluyen cambios en la forma de actualización de la feromona, y se amplían los casos de estudio, demostrando obtener mejores resultados en todas las instancias al ser comparada con *ACO I*, presentando superioridad frente a las heurísticas *RSA*, recocido simulado, *Meta-Raps* y búsqueda tabú.

EL PROBLEMA DE MÁQUINAS PARALELAS NO RELACIONADAS

El problema de la programación de trabajos en un sistema de múltiple capacidad de m máquinas que realizan igual operación en paralelo no relacionadas consiste en programar n trabajos (disponibles en el tiempo cero) que deben ser procesados en una (y solo una) de las m máquinas. Al ser máquinas no relacionadas, el tiempo de procesamiento del trabajo depende de la máquina en la que se procesa. Una máquina puede procesar un trabajo a la vez y lo hace sin interrupción, es decir, una vez iniciado el proceso de un trabajo continúa hasta terminarlo. Se consideran también tiempos de preparación dependientes de la secuencia y de las máquinas, es decir, el tiempo de preparación en la máquina k del trabajo j depende del trabajo i procesado previamente, el que a su vez es diferente para la misma situación en otra máquina.

La medida de desempeño que se utiliza para evaluar los programas de producción es el mayor tiempo de finalización entre todas las máquinas, conocido como *makespan* (C_{max}), el que debe ser minimizado. El *makespan* consiste en el intervalo de tiempo entre el inicio del procesamiento del primer trabajo (tiempo de referencia 0) y el tiempo

de finalización del procesamiento del último trabajo, es decir, el intervalo de tiempo en el que se procesa completamente la totalidad de los trabajos. En términos de la notación de tres parámetros propuesta por [12] , este problema se denota como $R_m/s_{ijk}/C_{max}$, donde R_m identifica un sistema de m máquinas paralelas no relacionadas, s_{ijk} identifica la presencia de tiempos de *setup* dependientes de la secuencia, y el tercer elemento identifica el objetivo a optimizar (C_{max}). Un caso especial de este problema es el problema de máquinas paralelas idénticas sin tiempos de preparación dependientes de la secuencia ($P_m//C_{max}$), que es considerado un problema *NP-hard* (incluso para dos máquinas). Dado que el problema $R_m/s_{ijk}/C_{max}$ es una generalización de este último, éste también es *NP-hard* [1].

Se consideran las siguientes características y supuestos:

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

A modo de ilustración, se considera un problema de 2 máquinas 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_{i1}	9	12	7	15	10	6
p_{i2}	11	8	10	11	13	8

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

s_{ij1}	**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
s_{ij2}	**1**	**2**	**3**	**4**	**5**	**6**
1	2	7	8	5	6	8
2	4	8	4	8	7	5
3	8	6	5	9	7	3
4	3	4	6	7	5	4
5	4	9	8	5	9	7
6	9	3	7	7	8	6

Utilizando la heurística *setupECT* (ver Figura 9), propuesta en este trabajo como una extensión de la heurística *ECT*, se obtiene la programación que se muestra en la carta Gantt de la Figura 1 (en las Tablas 1 y 2 se muestran en subrayado los tiempos de proceso y *setup* involucrados).

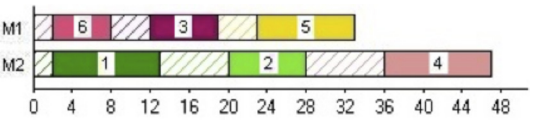

Figura 1. Carta Gantt-Programación.

El primer trabajo asignado a la máquina 1 (M1) es el trabajo 6, su tiempo de finalización es 0 (disponibilidad de M1) + 2 (*setup* inicial en M1) + 6 (tiempo de proceso de trabajo 6 en M1) = 8; a continuación se asigna el trabajo 3 finalizando en el tiempo 8 + 4 (*setup* en M1) + 7 (tiempo de proceso de trabajo 3 en M1) = 19. Finalmente se asigna el trabajo 5 cuyo tiempo de finalización es 19 + 4 (*setup* en M1) + 10 (tiempo de proceso trabajo 5 en M1) = 33. Análogamente, la programación sobre la máquina 2 (M2). M1 finaliza el proceso de su último trabajo en el tiempo 33, mientras que la M2 lo hace en el tiempo 47, así, el *makespan* resulta $C_{max} = 47$.

HEURÍSTICA GRASP

La heurística *GRASP* (*Greedy Randomized Adaptive Search Procedures*) fue propuesta por [13] y es un proceso iterativo en el que en cada iteración se

realizan una fase de construcción (utilizando un criterio *greedy*, adaptativo y aleatorizado) en la que se produce una solución factible, y una fase de búsqueda local en una vecindad de la solución construida en la fase de construcción. La mejor solución global encontrada se conserva como resultado.

La Figura 2 ilustra un pseudocódigo de *GRASP* de minimización, en donde *maxIterations* corresponde al número de iteraciones que se realizarán y *seed* a la semilla inicial para la generación de números pseudoaleatorios.

```
procedure GRASP(maxIterations, seed)
  Read_Input()
  for k=1, ... , maxIterations do
    Solution ← Greedy_Construction(seed)
    Solution ← Local_Search(Solution)
    Update(Solution, Best_Solution)
  end
  return Best_Solution
endprocedure
```

Figura 2. Pseudocódigo GRASP [14].

La solución factible generada en la fase de construcción se produce iterativamente agregando un elemento a la vez. En cada iteración de esta fase se forma una lista de elementos candidatos que pueden ser incorporados a la solución parcial sin destruir su factibilidad. La evaluación de los elementos candidatos se realiza mediante una función *greedy* que conduce a la lista restringida de candidatos (*RCL*) formada por los elementos candidatos mejor evaluados. El elemento de la lista *RCL* a ser incorporado a la solución es seleccionado en forma aleatoria. Una vez que el elemento seleccionado es agregado a la solución, se actualiza tanto la lista de candidatos como sus costos incrementales.

Las soluciones generadas en la fase de construcción no son necesariamente óptimas. La fase de búsqueda local usualmente mejora la solución generada en la fase de construcción realizando de manera iterativa una búsqueda en una vecindad de esta.

Una solución del problema en estudio es representada mediante una secuencia de n trabajos (orden en el que los trabajos son sucesivamente asignados a la máquina en la que finaliza antes). De esta manera, si denotamos por M_k al *makespan* parcial de la máquina k mientras se construye la solución, el tiempo de finalización del trabajo j quedará determinado por:

$$c_j = \min_{k=1,\ldots,m} \left\{ M_k + p_{jk} + s_{ijk} \right\} \qquad (1)$$

Una vez asignado un trabajo, se procede con el siguiente trabajo de la secuencia, hasta que no queden trabajos por asignar.

En las heurísticas *GRASP* propuestas en este trabajo, se utiliza la función *greedy* correspondiente al tiempo de finalización del trabajo j (asignado a la máquina donde finaliza antes), es decir:

$$g(j) = c_j \qquad (2)$$

Se define g_{min} y g_{max} como el menor y mayor costo de entre los trabajos aún no incorporados a la solución en construcción. Luego, para la definición de lista *RCL*, se ordenan los trabajos candidatos de menor a mayor de acuerdo a la función *greedy* y se seleccionan aquellos de menor costo restringidos por el parámetro α, $\alpha \in [0, 1]$:

$$RCL = \left\{ j / g(j) \le g_{\min} + \alpha\left(g_{\max} - g_{\min}\right) \right\} \qquad (3)$$

Una vez que se ha obtenido la lista *RCL* se selecciona aleatoriamente un trabajo de esta, el que es agregado a la solución parcial, repitiéndose el proceso hasta que no queden trabajos por asignar.

Para la fase de búsqueda local se utilizaron cuatro vecindades diferentes, las dos primeras trabajan sobre una representación de secuencia de la solución, mientras que las dos últimas lo hacen sobre una representación matricial, la que se representa mediante una matriz con filas de largo variable que contienen la secuencia de trabajos asignados a las respectivas máquinas (filas). La Figura 3 ilustra gráficamente la estructura secuencial y matricial de una solución para un problema de $n = 9$ trabajos y $m = 3$ máquinas.

La programación de los trabajos con una representación secuencial se realiza asignando los trabajos sucesivamente en el orden dado por la secuencia a la máquina 1, 2 ó 3, en la que el trabajo finaliza antes. En la representación matricial la

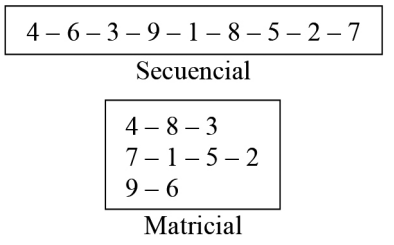

Figura 3. Representación de soluciones.

secuencia de trabajos en cada máquina está dada en forma explícita (fila *k* representa la secuencia de trabajos asignados a la máquina *k*). En ambos casos la programación considera los tiempos de proceso y *setup* del problema.

Vecindad 1 - IP (Intercambio a pares): En esta vecindad una solución del problema se representa mediante una secuencia de *n* trabajos que genera la programación mediante el procedimiento de asignación explicado anteriormente. La vecindad de una solución se construye realizando en forma sistemática todos los intercambios de dos trabajos hasta lograr una solución que mejora la semilla (criterio *first improvement*), iniciándose una nueva búsqueda a partir de la solución encontrada hasta que ya no es posible mejorar.

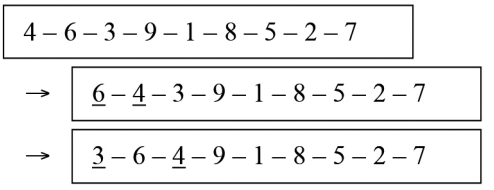

Figura 4. Intercambio a pares.

El máximo de intercambios en esta vecindad es $n\cdot(n+1)/2$, que corresponde a todos los posibles intercambios de dos trabajos en una secuencia de *n* trabajos. La Figura 4 ilustra el intercambio a pares en una solución de $n = 9$ trabajos con representación secuencial, mostrando los dos primeros intercambios sistemáticos. La heurística que utiliza esta vecindad se denomina *GRASP-1*.

Vecindad 2 - Intercambio aleatorio (S): Esta vecindad utiliza idéntica estructura que la Vecindad 1, pero se diferencia de esta en que los intercambios de dos trabajos se realizan en forma aleatoria un número controlado de veces. La heurística que utiliza esta vecindad se denomina *GRASP-2*.

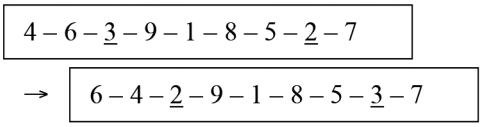

Figura 5. Intercambio aleatorio.

La Figura 5 ilustra el intercambio de los trabajos 3 y 2 seleccionados en forma aleatoria.

Vecindad 3 - Inserción max-min: En esta vecindad una solución es representada por una matriz. La vecindad se construye mediante la inserción de un trabajo seleccionado en forma aleatoria de la máquina con mayor tiempo de finalización, en la máquina con menor tiempo de finalización. Luego se aplica la heurística *MV* [15] del mejor vecino (ver Anexo), que reordena los trabajos en las máquinas involucradas en el proceso de inserción minimizando el tiempo total de proceso. La heurística que utiliza esta vecindad se denomina *GRASP-3*.

La heurística *MV* construye una secuencia de trabajos a procesar en una máquina, asignando sucesivamente, entre los trabajos no asignados, el que a continuación origina el menor *setup* para minimizar el tiempo total de proceso en la máquina.

La Figura 6 ilustra la inserción del trabajo 5 (seleccionado aleatoriamente) de la máquina 2 (mayor tiempo total de proceso) en la máquina 1 (menor tiempo total de proceso), asumiendo que al aplicar la heurística *MV*, en ambas máquinas la secuencia se reordena. Se ilustra el paso de una solución con $C_{max} = 53$ a una con $C_{max} = 52$.

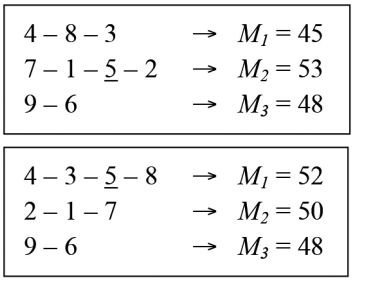

Figura 6. Inserción *max-min* (matriz).

Vecindad 4 - Intercambio aleatorio (M): Esta vecindad utiliza una estructura matricial, y se construye mediante intercambios aleatorios de dos trabajos seleccionados en forma aleatoria en dos máquinas distintas, también seleccionadas aleatoriamente. Luego se aplica la heurística *MV* que reasigna los trabajos en las máquinas seleccionadas para el intercambio minimizando el tiempo total de proceso. La heurística que utiliza esta vecindad se denomina *GRASP-4*.

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

Figura 7. Intercambio aleatorio (matriz).

La Figura 7 ilustra el intercambio del trabajo 8 con el trabajo 9 (ambos seleccionados aleatoriamente), asumiendo que al aplicar la heurística *MV*, en la máquina 1 la secuencia se reordena, mientras que en la máquina 3 se mantiene.

La heurística *ECT* (*earliest completion time*) y sus variantes, fueron propuestas por [16] para el problema de máquinas paralelas no relacionadas sin *setup*. En el presente trabajo los autores proponen una heurística constructiva (*setupECT*) que extiende la heurística *ECT* al caso con *setup*.

Manteniendo el sentido de la heurística original, el procedimiento de extensión consiste en programar primero aquel trabajo que tenga el menor tiempo de finalización considerando todas las máquinas y todos los trabajos no asignados, tomando en cuenta los *setup* involucrados. De esta manera, si se define U como el conjunto de trabajos no programados y M_k como el tiempo de finalización actual de la máquina k, su pseudocódigo se muestra en la Figura 8.

procedure *setupECT()*
$U = \{1, 2, \ldots, n\}$
for $k=1, \ldots, n$ **do** $M_k = 0$ **end**
while $U \neq \varnothing$ **do**
$t_{j(k)} = arg[\min_{k=1,\ldots,m} \{ M_k + p_{jk} + s_{ijk} \}; j \in U$
$j_0(k_0) \leftarrow arg[\min_{j=U} \{ t_{j(k)} \}]$
Asignar trabajo j_0 a máquina k_0
$M_{k0} \leftarrow M_{k0} + p_{jk0} + s_{ijk0}$
$U \leftarrow U - \{j_0\}$
end-while

Figura 8. Pseudocódigo heurística *setupECT*

EXPERIMENTO

Los resultados obtenidos mediante las heurísticas *GRASP* propuestas son comparados entre sí en términos de la calidad de la solución y de los tiempos computacionales requeridos. Luego se evalúa el desempeño de estas comparando sus resultados con heurísticas *Meta-Raps*, *ACO* y *búsqueda tabú* desarrolladas en trabajos de la literatura.

El conjunto de instancias utilizadas para la evaluación de los métodos propuestos y posterior comparación de resultados es obtenido del banco de problemas utilizados por [1], seleccionando problemas con tiempos de proceso dominantes y de seis tamaños *m*/*n* (*m* máquinas y *n* trabajos) diferentes: 2/20, 4/40, 6/60, 8/80, 10/100 y 12/120. Se seleccionó esta combinación de instancias para representar un ambiente de producción donde cada máquina en promedio debe procesar 10 trabajos. En estas instancias los tiempos de proceso de los trabajos se distribuyen uniformemente en el intervalo [125, 175], mientras que los tiempos de preparación de los trabajos se distribuyen uniformemente en el intervalo [50, 100]. Para cada tamaño de problema existen 15 instancias, lo que equivale a un conjunto de 90 problemas a evaluar.

La evaluación de las variantes *GRASP* y *setupECT* se realizó en un computador con procesador Intel Core i5 de 2.50 GHz y 4 GB de RAM con sistema operativo Windows 7 utilizando rutinas para para programar máquinas paralelas idénticas del programa *SPS_Optimizer* [17] adaptadas al caso de máquinas paralelas no relacionadas.

Para la comparación de resultados se realizó un análisis de la calidad de las soluciones mediante la evaluación del *makespan* (C_{max}). Como la mayoría de los problemas considerados son de gran tamaño, no es posible encontrar la solución óptima de estos en tiempos computacionales razonables. Por lo anterior, se utiliza una cota inferior para medir la calidad de las soluciones mediante el cálculo del máximo error relativo o la diferencia porcentual (DP) entre una solución factible respecto a la cota inferior:

$$DP = \frac{C_{\max_{método}} - CI}{CI} \times 100 \quad (4)$$

donde $C_{max_{método}}$ corresponde a la solución obtenida por el método y *CI* corresponde a la cota inferior del problema. La cota inferior utilizada en el estudio es una comúnmente utilizada en la literatura [11], y se calcula como CI=max(CI_1, CI_2), donde:

$$CI_1 = \frac{1}{m}\sum_{j=1}^{n} \min_{\substack{k=1,\ldots m \\ i=1,\ldots n}} \left[p_{jk} + s_{ijk} \right] \quad (5)$$

$$CI_2 = \max_{j=1,\ldots,n} \left\{ \min_{\substack{k=1,\ldots,m \\ k=1,\ldots,n}} \left[p_{jk} + s_{ijk} \right] \right. \quad (6)$$

Para la experimentación es necesario establecer algunos parámetros utilizados por las heurísticas *GRASP*, ya que tienen una influencia significativa en los resultados obtenidos. Estos parámetros corresponden al parámetro α y al número de iteraciones (parámetro *maxIiterations*). En un estudio experimental previo se analizó el rendimiento para valores de α = 0,0, 0,1, 0,2, 0,3, 0,4, 0,5, 0,6, 0,7, 0,8, 0,9 y 1,0, concluyéndose que los valores α = 0,0 y α = 0,1 son los que dominan absolutamente el rendimiento de la heurística *GRASP-1*, siendo adoptados en este trabajo para efecto de comparación entre las diferentes versiones de la heurística (se adopta igual parámetro α dado que este parámetro incide en la calidad de la solución generada por GRASP en la fase de construcción, que para todas las versiones de la heurística utiliza la misma función). Los valores α = 0,0 y α = 0,1 se ubican en el rango inferior de variación del parámetro α , lo que explica que las soluciones generadas en la fase de construcción lo hacen con un criterio más bien *greedy* que aleatorio.

El parámetro *maxIterations* se fijó en 500, determinado de manera que el tiempo computacional para una réplica de los problemas de mayor tamaño *n*/*m* = 12/120 no exceda, en promedio, los 10 minutos para *GRASP-1*. Se realizan 10 réplicas para cada instancia. Para las heurísticas *GRASP-2*, *GRASP-3* y *GRASP-4* se fija en 1000 el número de iteraciones en la fase de búsqueda local, igual para todas las vecindades de manera de que los resultados entre estas sean comparables.

Si bien, teóricamente, para α = 0,0 la fase de construcción *GRASP* debería obtener siempre la misma solución, en general, para este problema se producen empates, por lo que, la lista de candidatos contiene frecuentemente más de un trabajo resolviéndose los empates en forma aleatoria; esto se manifiesta en forma más marcada a medida que aumenta el tamaño de las instancias. Por otro lado, la búsqueda en vecindad en el caso de las heurísticas *GRASP-2*, *GRASP-3* y *GRASP-4* es una búsqueda en vecindad aleatoria, lo que ayudará a producir soluciones diferentes, aun cuando se parta de una misma solución.

RESULTADOS

Los resultados obtenidos con las heurísticas *GRASP* se presentan en la Tabla 3, donde se muestra la diferencia porcentual promedio, mínima y máxima sobre la cota inferior para α = 0,0 y α = 0,1, además del porcentaje de veces que se obtiene la mejor solución (% MS). En algunos casos % MS es mayor a 100 dado que para algunas instancias la mejor solución se obtiene tanto con α = 0,0 como con α = 0,1.

Para la heurística *GRASP-1* se observa que consigue la mejor solución un mayor porcentaje de veces con el parámetro α = 0,1, con excepción del caso *n*/*m* = 12/120. Las diferencias porcentuales promedios respecto de la cota inferior para esta heurística, oscilan entre 3,07% (α = 0,1) y 5,49% (α = 0,0) que se presentan para el caso *n*/*m* = 2/20 y para el caso *n* /*m* = 6/60, respectivamente. La menor diferencia porcentual promedio (sobre todas las instancias) se produce con α = 0,1. Para las variantes *GRASP-2* y GRASP-3 la menor diferencia porcentual promedio (sobre todas las instancias) también se produce con α = 0,1.

A diferencia de las anteriores, con la variante *GRASP-4* las mejores soluciones se obtienen en su mayoría con el parámetro α = 0,0, a excepción de los problemas tamaño *n*/*m* = 6/60. Las diferencias porcentuales promedios respecto de la cota inferior para esta heurística oscilan entre 2,99% (α = 0,1) y

Tabla 3. Diferencia porcentual, promedio, mínimo y máximo sobre cota inferior heurísticas *GRASP*.

	m/n	**2/20**		**4/40**		**6/60**		**8/80**		**10/100**		**12 /120**		**Total**	
	α	**0,0**	**0,1**	**0,0**	**0,1**	**0,0**	**0,1**	**0,0**	**0,1**	**0,0**	**0,1**	**0,0**	**0,1**	**0,0**	**0,1**
GRASP-1	Promedio	4,60	3,07	5,26	3,98	5,49	4,54	5,01	4,59	4,66	4,60	4,46	4,57	4,91	4,23
	Mínimo	2,77	1,87	3,60	3,37	4,44	3,96	4,41	3,99	4,17	4,22	3,98	4,19		
	Máximo	6,02	4,15	6,68	4,54	6,39	5,34	5,75	5,24	5,31	5,03	4,94	4,81		
	% MS	0	100	0	100	7	100	13	93	47	53	67	53		
GRASP-2	Promedio	5,02	3,69	5,75	4,29	6,08	4,98	5,49	5,01	5,07	5,06	4,92	5,04	5,39	4,68
	Mínimo	2,87	2,47	3,76	3,44	4,48	4,44	4,52	4,31	4,39	4,57	4,15	4,53		
	Máximo	6,46	4,92	7,18	4,97	7,21	5,48	6,44	5,74	5,64	5,42	5,61	5,59		
	% MS	0	100	0	100	7	93	20	93	47	53	73	27		
GRASP-3	Promedio	6,40	4,30	6,90	4,57	7,21	5,35	6,43	5,35	5,59	5,48	5,24	5,44	6,30	5,08
	Mínimo	4,51	2,77	4,40	3,47	6,08	4,44	5,19	4,33	4,56	4,91	4,64	4,83		
	Máximo	8,62	5,47	8,76	5,28	8,85	6,16	8,49	5,93	6,76	6,04	6,13	6,03		
	% MS	7%	100	0	100	0	100	0	100	53	53	73	33		
GRASP -4	Promedio	3,01	2,99	3,18	3,25	4,10	4,00	4,31	4,35	4,36	4,59	4,48	4,61	3,91	3,97
	Mínimo	1,87	1,87	2,46	2,51	3,66	3,52	3,68	3,91	3,97	4,22	4,05	4,13		
	Máximo	4,10	4,10	3,74	3,74	4,63	4,47	5,06	4,90	4,81	5,20	4,88	5,00		
	% MS	93	100	67	67	33	67	67	33	100	7	100	20		

4,61% ($\alpha = 0,1$) que se presentan para el caso *n/m* = 2/20 y para el caso *n/m* = 12/120, respectivamente. La menor diferencia porcentual promedio (sobre todas las instancias) se produce con $\alpha = 0,0$.

Al comparar los resultados obtenidos con las distintas variantes *GRASP (GRASP-1 (0,1)*, *GRASP-2 (0,1)*, *GRASP-3 (0,1)* y *GRASP-4 (0,0))*, se aprecia que la que obtiene menor diferencia porcentual promedio con respecto de la cota inferior sobre todas las instancias es la heurística *GRASP-4 (0,0)*, seguida de *GRASP-1 (0,1)*. En la Figura 9, se compara la mejor solución encontrada para cada tamaño de problema de las variantes *GRASP* con la heurística *setupECT*.

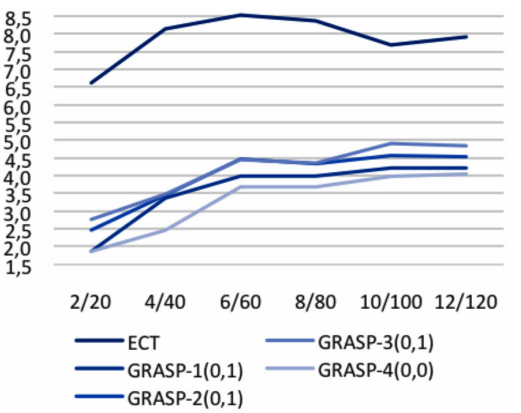

Figura 9. Diferencia porcentual de la mejor solución respecto a la cota inferior.

Se observa que la heurística *setupECT* presenta diferencias porcentuales sobre la cota inferior mayores que todas las variantes *GRASP*, alcanzando valores entre 6,62% y 8,53%, lo que indica, como era esperable, que las cuatro variantes *GRASP* presentadas entregan mejores resultados que una heurística constructiva. Por otro lado, la heurística *setupECT* se evidencia como una heurística constructiva de bajísimo esfuerzo computacional y de aceptable calidad al no distanciarse tan significativamente de métodos metaheurísticos con los que se compara, sobre todo en problemas de mayor tamaño. Se aprecia, además, que de las variantes *GRASP* nuevamente *GRASP-4 (0,0)* y *GRASP-1 (0,1)* son las que presentan diferencias porcentuales menores, las que varían, entre 1,87% y 4,19%, y 1,87% y 4,05%, respectivamente.

Los tiempos CPU(s) de las alternativas *GRASP* se presentan en la Tabla 4, donde se muestra un orden de magnitud para el promedio del tiempo CPU(s) para cada tamaño de problema. Se puede observar, que para la heurística *GRASP-1 (0,1)*, los tiempos computacionales por réplica aumentan considerablemente con el aumento del tamaño

Tabla 4. Tiempo CPU(s) promedio de las heurísticas *GRASP* por réplica.

m/*n*	2/20	4/40	6/60	8/80	10/100	12/120
GRASP-1(0,1)	1,5	12	45	120	300	600
GRASP-2(0,1)	5	6	8	12	15	20
GRASP-3(0,1)	25	25	25	25	25	25
GRASP-4(0,0)	25	25	25	25	25	25

del problema (pasando de 1,5 segundos para los problemas de tamaño *n*/*m* = 2/20 a 10 minutos para los problemas de tamaño *n*/*m* =12/120), mientras que la heurística *GRASP-2 (0,1)* presenta tiempos computaciones promedio por réplica que van desde 5 a 20 segundos, aumentando levemente a medida que aumenta el tamaño de los problemas (es la que presenta menores tiempo de ejecución).

Las alternativas *GRASP-3 (0,1)* y *GRASP-4 (0,0)*, muestran un tiempo computacional promedio por réplica similar del orden de los 25 segundos para todos los tamaños de problemas (notar que estas dos heurísticas en su fase de búsqueda local realizan aproximadamente igual número de operaciones independiente del tamaño del problema). Por otro lado, la heurística *setupECT* presenta tiempos de ejecución menores a 0,01 segundos para las instancias de mayor tamaño. Comparando la Figura 10 con los datos de la Tabla 4 se aprecia la ventaja de la variante *GRASP-4* con respecto a *GRASP-1*, que teniendo rendimientos comparables, el tiempo CPU de la variante *GRASP-4 (0,0)* es significativamente menor que *GRASP-1 (0,1)*.

El realizar la búsqueda en vecindad en forma aleatoria ayuda a mejorar las soluciones reduciendo en forma significativa el tiempo computacional.

Las variantes *GRASP-1 (0,1)* y *GRASP-4 (0,0)* se comparan con las metaheurísticas *búsqueda tabú* propuesta por [1], *Meta-Raps* propuesta por [5] y *ACO* propuesta por [10]; los resultados de las metaheurísticas son presentados por [10]. En la Tabla 5 se presenta la diferencia porcentual promedio respecto de la cota inferior considerando la mejor solución obtenida por cada método.

Del análisis que se realiza para cada tamaño de problema, se puede observar claramente que la heurística *GRASP-4 (0,0)* presenta en promedio los mejores resultados en casi todos los tamaños de problema estudiados, siendo superada por *ACO* y *Meta-Raps* para el caso *m*/*n* = 2/20, y levemente por *Meta-Raps* en el caso *m*/*n* = 12/120. En la mayoría de los casos las diferencias porcentuales son inferiores al 5% para cada instancia, por lo que es de interés analizar el porcentaje de veces en que cada heurística obtiene la mejor solución por cada tamaño de problema (ver Figura 10 para esta comparación).

De la Figura 10 se aprecia que para los problemas de tamaño *m*/*n* = 2/20 es la heurística *ACO* la que presenta mejores resultados, obteniendo en el 100% de los casos la mejor solución, mientras que para los problemas con *m* = 4, 6 y 10 máquinas es

Tabla 5. Diferencia porcentual promedio respecto de la cota inferior para cada tamaño de problema.

m/*n*	*Meta-Raps*	*tabu search*	*ACO*	*GRASP-1*	*GRASP-4*
2/20	2,60	3,97	2,52	3,07	3,01
4/40	3,88	5,14	3,44	3,98	3,18
6/60	4,37	6,38	4,58	4,54	4,10
8/80	4,62	6,33	4,44	4,59	4,31
10/100	5,22	6,79	4,91	4,60	4,36
12/120	4,25	7,05	4,64	4,57	4,48
Total	4,16	5,94	4,09	4,23	3,91

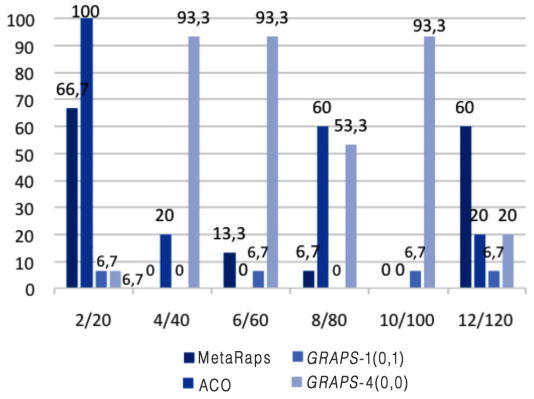

Figura 10. Porcentaje de mejores soluciones por tamaño de problema.

GRASP-4 (0,0) la que obtiene la mejor solución en un mayor número de casos, compartiendo su buen rendimiento en los problemas de tamaño $m/n = 8/80$ con ACO. Para los problemas de $m = 12$ máquinas las mejores soluciones se distribuyen entre diferentes heurísticas, obteniendo en un 60,0% *Meta-Raps*, en un 20,0% *ACO* y *GRASP-4 (0,0)*, y en un 6,7% *GRASP-1 (0,1)*, la mejor solución; llama la atención de que para este tamaño de problema la heurística *GRASP-4 (0,0)* obtiene en un bajo % de las instancias la mejor solución, sin embargo, sus resultados la ubican en forma consistente como la segunda mejor solución en la mayoría de las instancias donde no obtiene la mejor solución, lo que se corrobora con su desviación promedio respecto de la cota inferior para este tamaño de problemas (ver Tabla 5).

Cabe mencionar, que del total de 90 instancias evaluadas, la heurística *GRASP-4 (0,0)* encontró la mejor solución en 54 (60,0%) oportunidades, mientras que *ACO* lo hizo en 30 (33,3%), *Meta-Raps* en 22 (24,4%) y *GRASP-1* en 1 (6,7%) oportunidades. Las heurísticas *GRASP-2* y *GRASP-3 (0,1)*, así como también *tabu search* y la heurística *setupECT*, no encontraron en ninguna oportunidad la mejor solución.

CONCLUSIONES

Se han propuesto cuatro variantes *GRASP* como nuevas alternativas para resolver el problema de máquinas paralelas no relacionadas con tiempos de *setup* dependientes de la secuencia y minimización del *makespan*, las que se diferencian en la estructura de vecindad utilizada en la fase de búsqueda local. Para representar una solución se utiliza una secuencia de trabajos (*GRASP-1* y *GRASP-2*), que al ser asignados a las máquinas mediante un procedimiento de asignación generan el programa de producción, mientras que las heurísticas *GRASP-3* y *GRASP-4* utilizan una matriz de filas de largo variable para representar una solución. En particular, las variantes *GRASP-1* y *GRASP-4* mostraron un mejor rendimiento, y, que al ser comparadas con otros métodos de la literatura, han mejorado la mejor solución conocida en instancias.

Las variantes *GRASP* fueron comparadas con una heurística constructiva *setupECT* propuesta en este trabajo, que, aunque de buen desempeño fue superada por todas las variantes *GRASP*. Esto es algo que se esperaba, dado el carácter de heurística constructiva de la heurística *setupECT* propuesta, pero es destacable la calidad de la solución que obtiene con un bajo tiempo computacional, sobre todo en problemas de mayor tamaño.

Las variantes *GRASP-1* y *GRASP-4* mostraron los mejores resultados, por lo que se seleccionaron para comparar su desempeño con otros algoritmos de la literatura. Ambas heurísticas se muestran competitivas, destacando la heurística *GRASP-4* que supera a las otras heurísticas en un gran número de casos, y presentando bajos tiempos computacionales comparados con los de la variante *GRASP-1*.

REFERENCIAS

[1] M. Helal, G. Rabadi and A. Al-Salem. "A Tabu Search Algorithm to Minimize the Makespan for the Unrelated Parallel Machines Scheduling Problem with Setup Times". International Journal of Operations Research. Vol. 3 Nº 3, pp. 182-192. 2006.

[2] X. Zhu and W.E. Wilhelm. "Scheduling and Lot Sizing with Sequence-dependent Setup: A Literature Review". IIE Transactions. Vol. 38 Nº 11, pp. 987-1007. 2006.

[3] A. Allahverdi, C.T. Ng, T.C.E. Cheng and M.Y. Kovalyov. "A Survey of Scheduling Problems with Setup Times or Costs". European Journal of Operational Research. Vol. 187 Nº 3, pp. 985-1032. 2008.

[4] A. Al-Salem. "Scheduling to Minimize Makespan on Unrelated Parallel Machines with Sequence Dependent Setup Times". Engineering Journal of the University of Qatar. Vol. 17, pp. 177-187. 2004.

[5] G. Rabadi., R.J. Moraga and A. Al-Salem. "Heuristics for the Unrelated Parallel Machine Scheduling Problem with Setup Times". Journal of Intelligent Manufacturing. Vol. 17 Nº 1, pp. 85-97. 2006.

[6] E. Vallada and R. Ruiz. "A Genetic Algorithm for the Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times. European Journal of Operational Research. Vol. 211 Nº 3, pp. 612-622. 2011.

[7] P-Ch. Chang and S-H. Chen "Integrating Dominance Properties with Genetic Algorithms for Parallel Machine Scheduling Problems with Setup Times". Applied Soft Computing. Vol. 11 Nº 1, pp. 1263-1274. 2011.

[8] K-Ch. Ying, Z-J. Lee and Sh-W. Lin. "Makespan minimization for scheduling unrelated parallel machines with setup times". Journal of Intelligent Manufacturing. Vol. 23 Nº 5, pp. 1795-1803. 2012.

[9] K. Fleszar, Ch. Karalambous and K-S. Hindi. "A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times". Journal of Intelligent Manufacturing. Vol. 23 Nº 5, pp. 1949-1958. 2012.

[10] J.P. Arnaout, R. Musa and G. Rabadi. "A Two-stage Ant Colony Optimization Algorithm to Minimize the Makespan on Unrelated Parallel Machines-part II: Enhancements and Experimentations". Journal of Intelligent Manufacturing. Vol. 25 Nº 1, pp. 43-53. 2014.

[11] J.P. Arnaout, G. Rabadi and R. Musa. "A Two-stage Ant Colony Optimization Algorithm to Minimize the Makespan on Unrelated Parallel Machines with Sequence-dependent Setup Times". Journal of Intelligent Manufacturing. Vol. 21 Nº 6, pp. 693-701. 2009.

[12] R.L. Graham, E.L. Lawlwr, J.K. Lenstra and A.H.G. Rinnooy-Kan. "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey". Annals of Discrete Mathematics. Vol. 5 Nº 2, pp. 287-326. 1979.

[13] T.A. Feo and M. Resende. "A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem". Operations Research Letters. Vol. 8 Nº 2, pp. 67-71. 1989.

[14] M. Resende and C. Ribeiro. "Greedy Randomized Adaptive Search Procedures". In Handbook of Metaheuristics, pp. 219-249. 2003.

[15] E. Salazar y J.C. Medina. "Minimización del makespan en máquinas paralelas idénticas con tiempos de preparación dependientes de la secuencia utilizando un algoritmo genético". Ingeniería Investigación y Tecnología. Vol. XIV Nº 1 pp. 43-51. 2013.

[16] O.H. Ibarra and C.E. Kim. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. Journal of the ACM (JACM). Vol. 24 Nº 2, pp. 280-289. 1977.

[17] Salazar, E. Programación de Sistemas de Producción con SPS_Optimizer. Revista ICHIO. Vol. 1 Nº 2, pp. 33-46. 2010.

ANEXO

```
procedure MV()
  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
end_procedure
```

Figura A.1. Pseudocódigo heurística del mejor vecino *MV* (ver [15]).