INGENIERÍA Y DESARROLLO

Estudio computacional con técnicas heurísticas basadas en recocidos para resolver el problema de secuenciación de tareas

A Computational Study Solving the Flow Shop Problem with Annealing-Based Heuristics

David Álvarez Martínez*,
Eliana Mirledy Toro Ocampo**,
Ramón Alfonso Gallego Rendón***

* Ingeniero en Sistemas y Computación. Docente catedrático, Programa de Ingeniería de Sistemas y Computación, Universidad Tecnológica de Pereira. *akavallo@gmail.com*

** Magíster en Investigación de Operaciones y Estadística, Magíster en Ingeniería Eléctrica. Docente asistente, Facultad de Ingeniería Industrial, Universidad Tecnológica de Pereira. *elianam@utp.edu.co*

*** Doctor en Ingeniería Eléctrica Área de Automática. Docente titular, Programa de Ingeniería Eléctrica, Universidad Tecnológica de Pereira. *ragr@utp.edu.co*
Correspondencia: Vereda La Julita, Pereira, Risaralda (Colombia).

Número 25
Enero-Junio, 2009
ISSN: 0122-3461

Resumen

El secuenciación de tareas es una labor diaria de muchas empresas del sector de productos y servicios donde se busca optimizar algún o algunos de varios objetivos; aquí se propone minimizar el tiempo total de ejecución de todas las tareas. En este documento se presentan los resultados de un estudio computacional extensivo de 11 heurísticas basadas en el recocido: recocido simulado, aceptando el umbral, grabado a grabado y 8 heurísticas basadas en algoritmos demons. Para validar la calidad de las respuestas se seleccionaron 30 problemas de la literatura especializada. Se presentan los resultados obtenidos donde se compara la calidad de la solución con los tiempos de ejecución.
Palabras claves: Algoritmos *demons*, Flow-Shop, heurísticas, recocido simulado, secuenciación de tareas.

Fecha de recepción: 3 de septiembre de 2008
Fecha de aceptación: 24 de marzo de 2009

Abstract

Task sequencing is a daily job of many companies of the products sector and services, where it is sought to optimize one or several objectives. It is proposed here to minimize the total execution time of all tasks. This document presents the results of an extensive computational study of eleven heuristics based on annealing: Simulated Annealing, Threshold Accepting, Record-to-Record and 8 heuristics based on demon algorithms. To validate the quality of the answers, test cases of the specialized literature are used and the results obtained were compared in the quality of the solution and run times.
Key words: Demons algorithms, flow-shop, heuristics, simulated annealing, sequencing of tasks.

1. INTRODUCCIÓN

Muchos trabajos de ingeniería, planificación y manufactura pueden ser modelados como problemas de minimización o maximización de una función de costo sobre un conjunto de variables discretas. Esta clase de eventos son llamados problemas de optimización combinatorial; han recibido una gran atención en el campo de la ciencia durante las últimas tres décadas con el logro de grandes avances. Uno de los avances que se han realizado es la subdivisión de este tipo de problemas en dos categorías: los que se pueden resolver eficientemente o aquellos donde existe un algoritmo que resuelve toda instancia del ejercicio en un tiempo polinomial, como la programación lineal, y problemas para los cuales no se conoce un algoritmo que resuelva toda instancia de la situación propuesta en tiempo polinomial. Estos últimos son referenciados formalmente como

NP; el problema de secuenciación de tareas es considerado de acuerdo a su complejidad matemática como NP- completo.

El problema de secuenciación de tareas en sistemas de producción lineal, *Flow-Shop*, se caracteriza porque todos los trabajos que van a ser programados tienen el mismo flujo de producción, aunque varían los tiempos de ejecución en cada una de las estaciones. Para resolverlo se han propuesto diferentes técnicas y metodologías a fin de encontrar la secuencia óptima de fabricación que cumple con diferentes objetivos, como la inexistencia de tiempos muertos de fabricación, reducción de tiempo de cambio y ajuste de las máquinas, anulación de retrasos, entre otros, pero teniendo en cuenta las restricciones propias de cada situación específica como la velocidad de proceso de las máquinas, capacidad de recursos humanos y materiales, etc.

Las técnicas metaheurísticas han mostrado su gran potencial como herramientas de solución en variados campos de aplicación por su eficiencia en cuanto a tiempos de solución y calidad de las respuestas obtenidas. Aquí se están proponiendo 11 técnicas basadas en algoritmos recocidos para la solución del problema *Flow-Shop* que considera como función objetivo la minimización del tiempo total de ejecución de todas las tareas y se comparan las respuestas obtenidas con las reportadas en la literatura especializada.

Este documento está organizado de la siguiente forma: en la sección 2 se describe el problema del *Flow-Shop* y su modelamiento matemático; en la sección 3 se presentan algunas generalidades de los algoritmos utilizados; en la sección 4 se presenta el procedimiento empleado para calcular la función objetivo, la estructura de vecindad y la adecuación de los algoritmos al problema especifico; en la sección 5 se presentan los casos de prueba y los resultados obtenidos y, finalmente, en la sección 6 se plantean las conclusiones y trabajos futuros con respecto a los problemas de secuenciación.

2. DEFINICIÓN DEL PROBLEMA DEL *FLOW-SHOP*

El problema del *Flow-Shop* permutacional representa un caso particular del problema del *Flow-Shop Scheduling* cuyo principal objetivo es entregar una secuencia óptima para N trabajos en M máquinas.

La solución del problema consiste en secuenciar *n* trabajos (*i=1,..., n*) en *m* máquinas (*j=1,...m*). Un trabajo consiste en *m* operaciones y la *j-ésima* operación de cada trabajo debe ser procesada en la máquina *j*, se debe considerar que:

a) Al iniciar un trabajo en la máquina j este ya ha sido procesado en la máquina j-1 y, adicionalmente, la máquina j no está ejecutando operación alguna. Cada operación tiene un tiempo de procesamiento conocido P_{ij}

b) Si un trabajo está en la i-ésima posición en la máquina 1, entonces ese trabajo estará en la posición i-ésima en todas las máquinas.

c) La secuencia de producción de todos los trabajos es igual.

d) No se consideran tiempos de ajuste de las máquinas entre un trabajo y otro.

e) Cuando se inicia la ejecución de un trabajo en una máquina no puede ser interrumpido.

f) El *makespan* es un parámetro que indica el tiempo total de ejecución de todas las tareas.

Para la solución del problema del *Flow-Shop* permutacional se considera el *makespan* como función objetivo a ser minimizada, resolver el problema significa determinar la permutación que entregue el menor valor de *makespan*. En este contexto se considera el trabajo J_i como un conjunto de operaciones, que pasan por cada máquina una sola vez:

- $J_i\{o_{i1}, o_{i2}, o_{i3}, \ldots, o_{iM}\}$, donde o_{ij} representa la *j-ésima* operación del trabajo J_i;
- o_{ij} operación debe ser procesada en la M_j máquina;
- Por cada operación o_{ij} hay asociado un tiempo de procesamiento p_{ij}.

Tradicionalmente, la notación del problema es $F \mid permu \mid C_{max}$ considerando como objetivo minimizar todo el tiempo de procesamiento (*makespan)*.

Un ejemplo del problema permutacional del *Flow-Shop* se describe a continuación:

Se definen $\pi_1, \pi_2, \pi_3, \ldots, \pi_N$ como permutaciones. El cálculo del tiempo final $C(\pi_i, j)$ para el *i-ésimo* trabajo dado en la permutación π en la máquina j puede ser calculado de la siguiente forma:

$$C(\pi_1, 1) = p_{\pi 1}, 1$$
$$C(\pi_i, 1) = C(\pi_{i-1}, 1) + p_{\pi i}, 1, \qquad i=2,\ldots, N$$
$$C(\pi_1, 1) = C(\pi_i, j-1) + p_{\pi 1}, j, \qquad j=2,\ldots, M$$
$$C(\pi_i, j) = \max\{C(\pi_{i-1}, 1), C(\pi_1, j-1) + p_{\pi i}, j\} \qquad i=2,\ldots, N; j=2,\ldots, M$$

Bajo estas especificaciones, el valor de la función objetivo, el *makespan* C_{max}, se define como $C(\pi_N, M)$; el tiempo completo de la última operación en la última máquina.

Este problema, como muchos otros en el campo de secuenciación, tiene muchos inconvenientes por resolver ya que técnicamente está clasificado como NP-difícil. Sin entrar en detalles técnicos se dice que un problema es NP-dífícil cuando se demuestra que cualquier algoritmo de solución tiene un tiempo de ejecución que aumenta exponencialmente con el tamaño del problema [1]. El que un problema esté catalogado en esta categoría no significa que no puede resolverse, sino que se deben proponer algoritmos de solución que exploten de forma eficiente su misma estructura matemática para que se encuentren soluciones a la mayoría de las instancias del problema, en tiempos de ejecución relativamente pequeños.

3. HEURÍSTICAS DE BÚSQUEDA LOCAL

Durante los últimos cuarenta años se han desarrollado gran cantidad de heurísticas para la solución del problema de secuenciación de tareas, buscando mejorar el *makespan***,** de forma iterativa, donde cada nueva secuenciación toma un menor tiempo C_{max}. En los años noventa, las investigaciones se centraron en aplicar metaheurísticas de propósito general para el problema del *Flow-Shop***.** Las metaheurísticas seleccionan configuraciones que a veces incrementan el *makespan* y de ese modo obtienen una herramienta para no quedar atrapadas en óptimos locales de mala calidad. En esta sección, se describirán 11 metaheurísticas para el problema del *Flow-Shop***:** recocido

simulado, dos tipos de recocidos determinísticos y ocho variantes de algoritmos *demons*.

3.1. **Recocido simulado**

El algoritmo del recocido simulado (RS) reproduce un conjunto de átomos en equilibrio a una temperatura determinada. El RS empieza con un estado inicial; luego se propone un cambio aleatorio para este estado y se genera un cambio de energía: ΔE es calculado [1] [2].

Si el nuevo estado tiene un nivel más bajo de energía que el estado anterior, $\Delta E \leq 0$, el nuevo estado, es tomado para la siguiente iteración. Sin embargo, si el nuevo estado tiene un nivel de energía más alto que el anterior, se acepta con una probabilidad $P(\Delta E) = \exp(-\Delta E/k_B.\ T)$, donde T es la temperatura y k_B es la constante de Boltzmann. Esta característica hace que el RS sea diferente a los algoritmos de búsqueda local. El algoritmo RS es presentado en la tabla 1.

El programa de enfriamiento en el paso 3.f del RS es crucial para el éxito del algoritmo. El valor de aceptación de un empeoramiento de la energía se mueve inversamente proporcional al ΔE y proporcionalmente a T, que decrece con el tiempo. Si se configura inteligentemente el enfriamiento, el algoritmo puede escapar temprano de óptimos locales y explorar en profundidad otras regiones prometedoras.

3.2. **Recocidos determinísticos**

Se presentan dos versiones determinísticas del recocido simulado que se denominan: aceptando el umbral y grabado a grabado [2] y [3]. Un movimiento de empeoramiento es aceptado, si la desmejora en la función objetivo es menor que un monto determinado.

1) Aceptando el umbral (AU): un umbral U es especificado como se muestra en la tabla 1. Este umbral es la cuota máxima de decremento de la función objetivo aceptada entre una iteración y la siguiente. En el paso 3.c de AU, una nueva secuencia de tareas es aceptada solo si ΔE, el cambio de energía, es menor que U, Así, son aceptadas configuraciones con peor función objetivo. En el paso 3.e de AU, el umbral es enfriado de acuerdo a lo programado.

2) Grabado a grabado (GG): en el algoritmo de grabado a grabado tal y como se muestra en la tabla 1, el *GRABADO* es el mejor *makespan* encontrado hasta el momento. Esta técnica solo puede explorar soluciones con *makespan* menor que la desviación, *D*, más el *GRABADO* (ver paso 3.b la tabla 1). Así, el GG permite aceptar empeoramientos en la función objetivo de una nueva configuración, pero el valor de *GRABADO* + *D* sirve como límite de la desmejora que puede ser aceptada.

3.3. **Algoritmos *demons***

Los algoritmos *demons* son una variante del recocido simulado [5], [6]. El algoritmo *demon* reemplaza la probabilidad de aceptación de una configuración con peor función objetivo por el concepto de crédito llamado *demon*. Cuando el algoritmo acepta una secuencia con función objetivo de peor calidad, el crédito o *demon* decrece con relación al deterioro de la nueva configuración. Similarmente, si la nueva secuencia es de mejor calidad que la anterior, se debita al *demon* la mejora de la función objetivo. Una nueva secuencia de peor calidad solo será aceptada si el demon tiene suficiente crédito para pagar la desmejora. El algoritmo *demon* básico (DB) es dado en la tabla 1.

Wood y Downs [5], [6] modificaron el DB y desarrollaron cuatro algoritmos *demons* estándares para optimización: algoritmo *demon* limitado (DL), algoritmo *demon* aleatorio limitado (DAL), algoritmo *demon* recocido (DR) y algoritmo *demon* aleatorio recocido (DAR). Los algoritmos DL, DAL, DR y DAR son presentados en la tabla 2.

El DB es su forma original no es un algoritmo apropiado para minimización. Wood y Downs [5], [6] propusieron dos modificaciones que remueven gradualmente energía del *demon* para enfrentar la minimización. Una modificación es imponer un límite superior específico como valor al *demon* para que pueda restringir valores de crédito, después de movidas con decrementos de energía. El límite superior previene al *demon* en contra de recibir todo el crédito, de aceptar secuencia de peor calidad. Esta modificación es incorporada en el algoritmo DL (el límite superior es dado por D_0. Ver su implementación en el algoritmo DL en la tabla 2).

La segunda modificación reduce el valor del *demon* acorde a una programación específica de la misma forma que Kirkpatrick y otros [2]

disminuyen la temperatura en RS. Esta modificación es incorporada en el algoritmo *demon* recocido (DR). Con estas dos modificaciones, un algoritmo disminuye el crédito dado al *demon***.** Un algoritmo tendrá menor probabilidad de aceptar secuencias de peor calidad después de aceptar configuraciones con bajos valores de *makespan*, pues este es forzado a buscar secuencias de mejor calidad.

Wood y Downs intentaron incorporar en los algoritmos DL y DR componentes aleatorios en los cuales el valor del *demon* fuera reemplazado por el valor medio del *demon* y para que este, a su vez, hiciera el papel de acreedor. El valor del *demon* es una variable aleatoria *gaussiana* (normal) centrada alrededor del valor medio del *demon* (D_M) con una desviación estándar específica (D_{Sd}).

Las dos versiones aleatorias (DAL y DAR) de los algoritmos limitados y recocidos se obtienen reemplazando D con D_M (ver la tabla 2) y agregando el paso (3.d en la tabla 2) para generar el valor del *demon* de una distribución con media D_M **+** un valor de *ruido gaussiano*. Por el uso de componentes aleatorios, Wood y Downs alcanzan largos incrementos en energía que no son permitidos por algoritmos determinísticos.

Se señala que Wood y Downs probaron los cuatro algoritmos (DL, DAL, DR y DAR) para instancias del problema del cartero viajante.

3.4. **Variantes del algoritmo demon**

Estas versiones fueron construidas con base en cuatro algoritmos demons estándares presentados por Pepper, Golden y Wasil [7]: algoritmo *demon* recocido limitado (DRL), algoritmo *demon* aleatorio limitado recocido (DARL), algoritmo *demon* recocido híbrido (DRH) y algoritmo *demon* recocido limitado híbrido (DRLH). Los algoritmos DRL, DARL, DRH y DRLH son presentados en la tabla 3.

En DRL y DARL se aplicaron límites y enfriamiento al valor del *demon* esperando que este forzara al valor del *demon* a decrecer lentamente durante el proceso. Esto puede llevar a secuencias de alta calidad.

En el DARL se enfría la desviación estándar del valor del *demon* además de enfriarse el valor medio del *demon*. En el DRLH, se enfría la desviación

estándar del valor del *demon* y se enfría el valor límite del *demon*. Se tienen así diferentes enfriamientos programados para la desviación estándar, el valor medio del *demon* y el valor límite del *demon*. Al mismo tiempo, los componentes aleatorios son reducidos por lo que las versiones híbridas tienden hacia su contraparte determinística.

Tabla 1
Recocido simulado, aceptando el umbral, grabado a grabado y *demon* básico

RS
1. Escoja una secuencia inicial *S*.
2. Halle la temperatura inicial $T = T_0 > 0$.
3. Repita:
   a. Escoja una nueva secuencia S′.
   b. Sea $\Delta E = E(S)' - E(S)$, donde *E(S)* es la energía (*makespan*) de la secuencia *S*.
   c. Si $\Delta E \leq 0$, acepte la nueva secuencia, haga S=S′.
   d. De lo contrario, si $exp(-\Delta E/T) \geq rand(0,1)$, acepte la nueva secuencia.
   e. De lo contrario, rechace la secuencia.
   f. Reduzca la temperatura de acuerdo al enfriamiento programado.
4. Hágalo hasta la condición de parada.

AU
1. Escoja una secuencia inicial *S*.
2. Escoja un umbral $U = U_0 > 0$.
3. Repita:
   a. Escoja una nueva secuencia *S′*.
   b. Sea $\Delta E = E(S') - E(S)$, donde *E(S)* es la energía (*makespan*) de la secuencia *S*.
   c. Si $\Delta E \leq U$, acepte la nueva secuencia, haga $S = S'$.
   d. De lo contrario, rechace la secuencia.
   e. Reduzca el umbral de acuerdo al enfriamiento programado.
4. Hágalo hasta la condición de parada.

GG
1. Escoja una secuencia inicial *S* y sea $GRABADO = E(S)$.
2. Escoja una desviación $D > 0$.
3. Repita:
   a. Escoja una nueva secuencia *S′*.
   b. Si $E(S') < GRABADO + D$.
      i. Haga $S = S'$.
      ii. Si $E(S') < GRABADO$, asigne $GRABADO = E(S')$.
4. Hágalo hasta la condición de parada.

DB
1. Escoja una secuencia inicial *S*.
2. Escoja valor de *demon* $D > 0$.
3. Repita:
   a. Escoja una nueva secuencia S′.
   b. Sea $\Delta E = E(S') - E(S)$, donde *E(S)* es la energía (*makespan*) de la secuencia S.
   c. Si $\Delta E < D$, acepte la nueva secuencia y haga $D = D - \Delta E$.
4. Hágalo hasta la condición de parada.

Tabla 2
Demon limitado, *demon* limitado aleatorio, *demon* recocido
y *demon* recocido aleatorio

DL
1. Escoja una secuencia inicial *S*.
2. Escoja valor de *demon* $D = D_0 > 0$.
3. Repita:
   a. Escoja una nueva secuencia S′.
   b. Sea $\Delta E = E(S') - E(S)$, donde $E(S)$ es la energía (*makespan*) de la secuencia S.
   c. Si $\Delta E < D$, acepte la nueva secuencia y haga $D = D - \Delta E$.
   d. Si $D > D_0$, haga $D = D_0$.
4. Hágalo hasta la condición de parada.

DAL
1. Escoja una secuencia inicial *S*.
2. Escoja valor de *demon* $D_M = D_{M0} > 0$.
3. Repita:
   a. Escoja una nueva secuencia S′.
   b. Sea $\Delta E = E(S') - E(S)$, donde $E(S)$ es la energía (*makespan*) de la secuencia S.
   c. Si $\Delta E < 0$, acepte la nueva secuencia y haga $D_M = D_M - \Delta E$.
   d. De lo contrario.
      i. $D = D_M$ + *ruido gaussiano*
      ii.Si $\Delta E < D$, acepte la nueva secuencia $S = S'$ **y** $D_M = D_M - \Delta E$.
   e. Si $D_M > D_{M0}$, haga $D_M = D_{M0}$.
4. Hágalo hasta la condición de parada.

DR
1. Escoja una secuencia inicial *S*.
2. Escoja valor de *demon* $D > 0$.
3. Repita:
   a. Escoja una nueva secuencia S′.
   b. Sea $\Delta E = E(S') - E(S)$, donde $E(S)$ es la energía (*makespan*) de la secuencia S.
   c. Si $\Delta E < D$, acepte la nueva secuencia y haga $D = D - \Delta E$.
   d. Reduzca *D* de acuerdo al enfriamiento programado.
4. Hágalo hasta la condición de parada.

DAR
1. Escoja una secuencia inicial *S*.
2. Escoja valor de *demon* $D_M = D_{M0} > 0$.
3. Repita:
   a. Escoja una nueva secuencia S′.
   b. Sea $\Delta E = E(S') - E(S)$, donde $E(S)$ es la energía (*makespan*) de la secuencia S.
   c. Si $\Delta E < 0$, acepte la nueva secuencia y haga $D_M = D_M - \Delta E$.
   d. De lo contrario,
      i. $D = D_M$ + *ruido gaussiano*.
      ii. Si $\Delta E < D$, acepte la nueva secuencia $S = S'$ **y** $D_M = D_M - \Delta E$.
   e. Reduzca D_M de acuerdo al enfriamiento programado.
4. Hágalo hasta la condición de parada.

Tabla 3

Demon recocido limitado, *demon* aleatorio recocido limitado, *demon* recocido híbrido y *demon* recocido limitado hibrido

DRL

1. Escoja una secuencia inicial *S*.
2. Escoja valor de *demon* $D = D_0 > 0$.
3. Repita:
   a. Escoja una nueva secuencia S′.
   b. Sea $\Delta E = E(S') - E(S)$, donde $E(S)$ es la energía (*makespan*) de la secuencia S.
   c. Si $\Delta E < D$, acepte la nueva secuencia y haga $D = D - \Delta E$.
   d. Reduzca D_0 de acuerdo al enfriamiento programado.
   e. Si $D > D_0$, hasta $D = D_0$.
4. Hágalo hasta la condición de parada.

DARL

1. Escoja una secuencia inicial *S*.
2. Escoja valor de *demon* $D_M = D_{M0} > 0$.
3. Repita:
   a. Escoja una nueva secuencia S′.
   b. Sea $\Delta E = E(S') - E(S)$, donde $E(S)$ es la energía (*makespan*) de la secuencia S.
   c. Si $\Delta E < 0$, acepte la nueva secuencia y haga $D_M = D_M - \Delta E$.
   d. De lo contrario,
      i. $D = D_M$ + *ruido gaussiano*.
      ii. Si $\Delta E < D$, acepte la nueva secuencia $S = S'$ **y** $D_M = D_M - \Delta E$.
   e. Reduzca D_{M0} de acuerdo al enfriamiento programado.
   f. Si $D_M > D_{M0}$, haga $D_M = D_{M0}$.
4. Hágalo hasta la condición de parada.

DRH

1. Escoja una secuencia inicial *S*.
2. Escoja valor de *demon* $D_M = D_{M0} > 0$.
3. Repita:
   a. Escoja una nueva secuencia S′.
   b. Sea $\Delta E = E(S') - E(S)$, donde $E(S)$ es la energía (*makespan*) de la secuencia S.
   c. Si $\Delta E < 0$, acepte la nueva secuencia y haga $D_M = D_M - \Delta E$.
   d. De lo contrario,
      i. $D = D_M$ + *ruido gaussiano*.
      ii. Si $\Delta E < D$, acepte la nueva secuencia $S = S'$ **y** $D_M = D_M - \Delta E$.
   e. Reduzca D_{M0} de acuerdo al enfriamiento programado.
   f. Reduzca D_{Si} de acuerdo al enfriamiento programado.
4. Hágalo hasta la condición de parada

DRLH

1. Escoja una secuencia inicial *S*.
2. Escoja valor de *demon* $D_M = D_{M0} > 0$.
3. Repita:
   a. Escoja una nueva secuencia S´.
   b. Sea $\Delta E = E(S') - E(S)$, donde $E(S)$ es la energía (*makespan*) de la secuencia S.
   c. Si $\Delta E < 0$, acepte la nueva secuencia y haga $D_M = D_M - \Delta E$.
   d. De lo contrario,
      i. $D = D_M$ + *ruido gaussiano*
      ii. Si $\Delta E < D$, acepte la nueva secuencia $S = S'$ **y** $D_M = D_M - \Delta E$.
   e. Reduzca D_{M0} **d**e acuerdo al enfriamiento programado.
   f. Reduzca D_{Sd} de acuerdo al enfriamiento programado.
   g. Si $D_M > D_{M0}$, haga $D_M = D_{M0}$.
4. Hágalo hasta la condición de parada.

4. PROCEDIMIENTOS

4.1. **Codificación del problema**

4.1.1. ***Matriz de tiempos***

Los tiempos de duración de cada operación para un trabajo determinado en una máquina se representan mediante una matriz, que se denomina matriz de tiempos o matriz de duración con la arquitectura mostrada en la figura 1.

La columna j de la matriz de tiempos representa el tiempo que tarda en finalizar cada operación en la máquina i. La operación del trabajo 1 en la máquina 1 toma 5 unidades de tiempo, $D_{1,1} = 5$

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

Figura 1. Matriz de tiempos

4.1.2. ***Secuencia de trabajo***

El orden de entrada de los trabajos a las máquinas se representa mediante un vector, donde su propia longitud indica el número de trabajos a procesar;

cada elemento dentro del vector representa la tarea que se ejecutará en cada posición, como se observa en la figura 2. Si se lee de izquierda a derecha, el trabajo 2 es el primero que se ejecutará y el trabajo 4 está en la segunda posición del vector; esto representa que el trabajo 4 entrará en segunda instancia.

Trabajo 2	Trabajo 4	Trabajo 1	Trabajo 3

Figura 2. Vector de ordenamiento

4.1.3. *Cálculo del* **makespan**

El cálculo del *makespan* guarda relación con la matriz de tiempos y la secuencia de trabajo, el *Flow-Shop* presenta las siguientes particularidades: el primer trabajo programado solo debe respetar la secuencia tecnológica y la primera máquina nunca tiene tiempo ocioso entre tareas; estas dos propiedades permiten llenar la primera columna y la fila de la matriz de inicio de la siguiente forma:

- El tiempo de inicio de la operación del primer trabajo en la máquina *i* será el tiempo de inicio de la operación del primer trabajo en la máquina *i-1* más el tiempo de duración de la operación del primer trabajo en la máquina *i-1*.
  $T_{i,1} = T_{i-1,1} + D_{i-1,1}$, donde $1 < i < m$.

- El tiempo de inicio de la primera operación del trabajo *j* en la primera máquina, donde $1 < j < n$, será el tiempo de inicio de la primera operación del trabajo *j-1* en la primera máquina más el tiempo de duración de la primera operación del trabajo *j-1* en la primera máquina.
  $T_{1,j} = T_{1,j-1} + D_{1,j-1}$, donde $1 < j < n$.

Para terminar la construcción de la matriz de inicio se recorre por columnas o filas y el tiempo de inicio $T_{i,j}$ será el máximo entre:

- El tiempo de inicio $T_{i,j-1}$ más el tiempo de duración $D_{i,j-1}$
- El tiempo de inicio $T_{i-1,j}$ más el tiempo de duración $D_{i-1,j}$

Luego de terminada la matriz de inicio, el valor del *makespan* es la suma de las dos esquinas inferiores localizadas a la derecha tanto de la matriz de tiempos ordenada como de la matriz de inicio, $D_{m,n} + T_{m,n} = makespan$

Ejemplo de aplicación del algoritmo para calcular el makespan

Paso 1. Reordenar la matriz de tiempos según la secuencia dada tal como se muestra en la figura 3.

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

Figura 3. Matriz reordenada con base en la secuencia propuesta

Paso 2. Crear la matriz de tiempos tal y como se muestra en la figura 4.

Paso 2.1 Ubicar un cero en la esquina superior izquierda.

Paso 2.2 $T_{i,1} = T_{i-1,1} + D_{i-1,1.}$ Figura 4.

Paso 2.3 $T_{1,j} = T_{1,j-1} + D_{1,j-1.}$ Figura 5.

Paso 2.4 Máximo entre el tiempo de inicio $T_{i,j-1}$ más el tiempo de duración $D_{i,j-1}$ y el tiempo de inicio $T_{i-1,j}$ más el tiempo de duración $D_{i-1,j}$, tal como se muestra en la figura 6

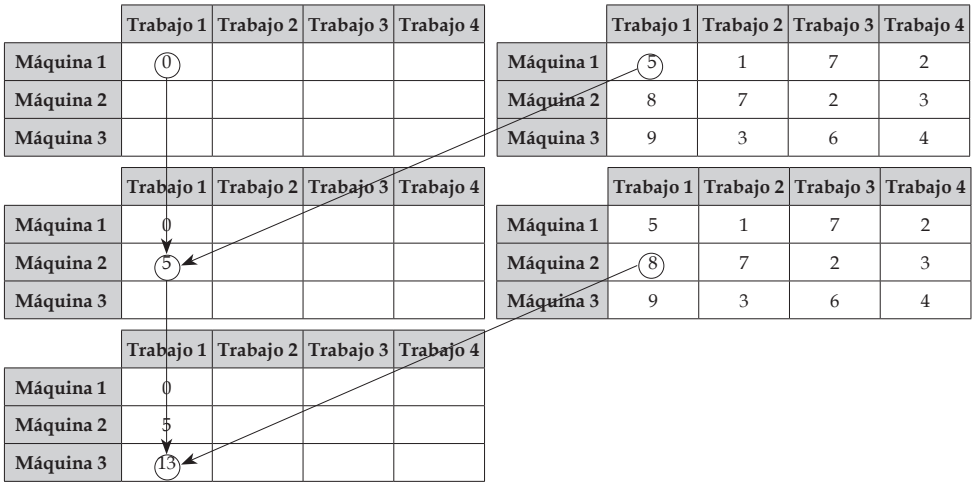

Figura 4. Matriz de tiempos después de aplicar el paso 2.2

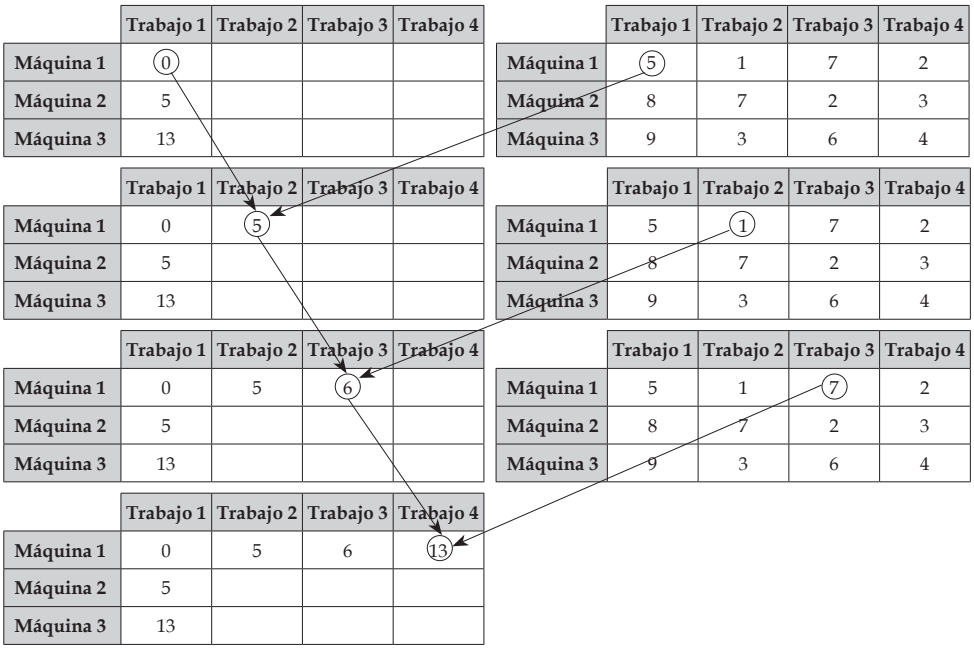

Figura 5. Matriz de tiempos después de aplicar el paso 2.3

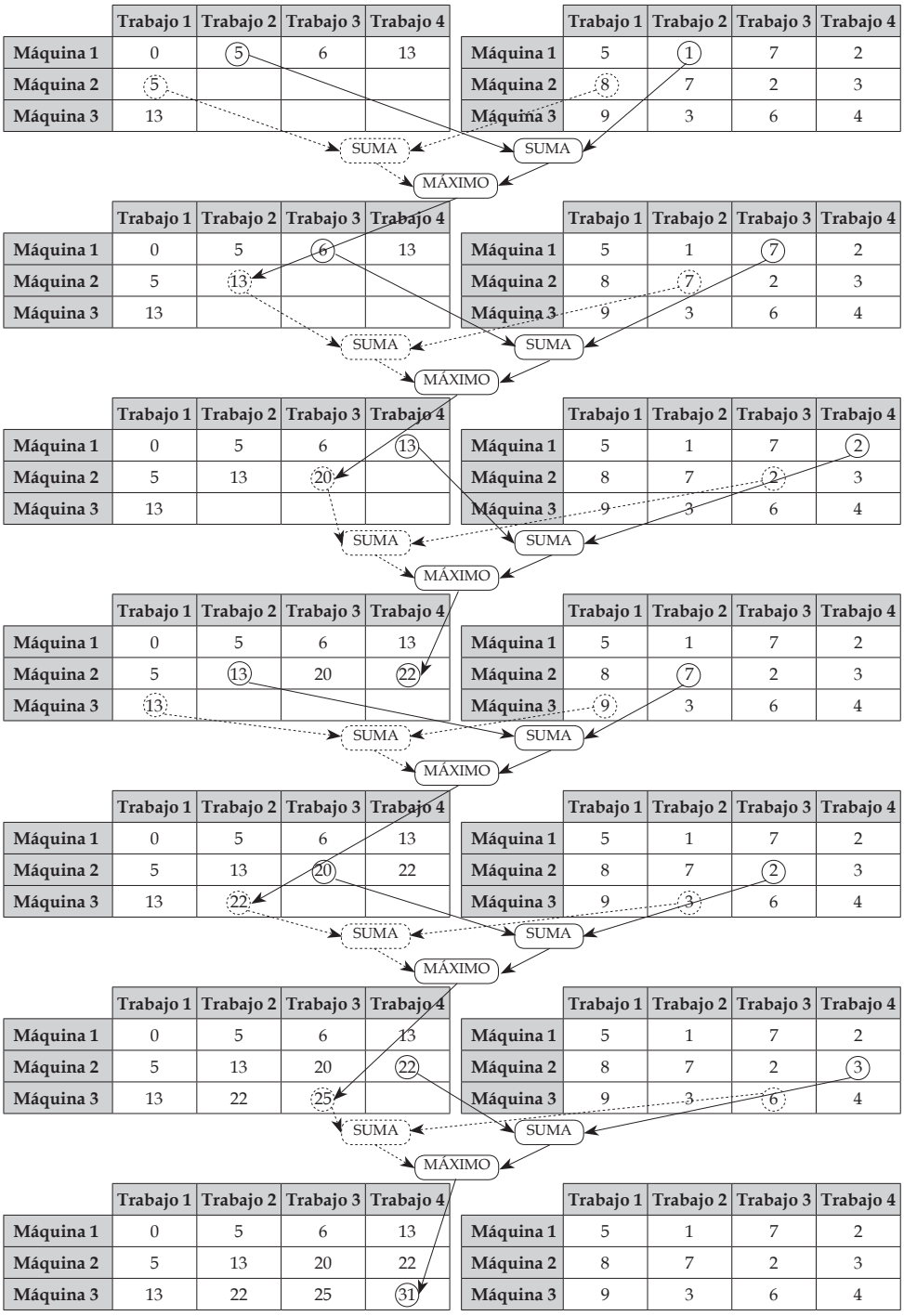

Figura 6. Matriz de tiempos después de aplicar el paso 2.4

Paso 3. Sumar las esquinas inferiores situadas hacia la derecha de la matriz de inicio y de tiempos. De acuerdo como se muestra en la figura 7.

	Trabajo 1	Trabajo 2	Trabajo 3	Trabajo 4
Máquina 1	0	5	6	13
Máquina 2	5	13	20	22
Máquina 3	13	22	25	31

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

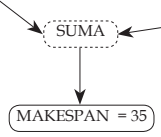

Figura 7. Cálculo final del *makespan*

El *makespan* del ejercicio para la matriz y secuencia dada es: *makespan = 31 + 4 = 35*.

4.2. **Estructura de vecindad**

La estructura de vecindad para cualquier algoritmo es muy importante; incluso, se puede asegurar que es la clave de un buen desarrollo. Aquí la estructura de vecindad es la misma empleada para los 11 métodos estudiados. Luego de analizar el problema de secuenciación de tareas y las posibles estructuras de vecindad estudiadas, se logra identificar que los mejores resultados alcanzados se encuentran utilizando troca (s*wap*) y corrimiento de tareas.

La troca o *swap* consiste en seleccionar dos tareas cualesquiera de la secuencia figura 8; e intercambiarlas entre sí, ver figura 9 para obtener un nueva secuencia ver figura 10.

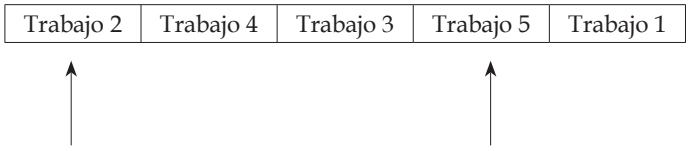

Figura 8. Elección de tareas que se truecan

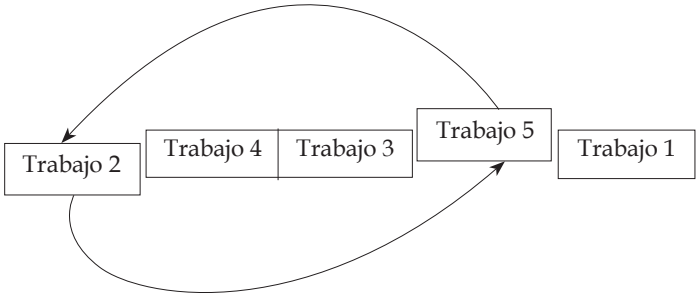

Figura 9.Trueco de tareas

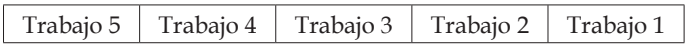

Figura 10. Secuencia después del trueco o trueque

El corrimiento consiste en seleccionar una tarea al azar de la secuencia, ver figura 11; escoger una nueva posición aleatoria para la tarea asignada, ver figura 12; y llevar la tarea escogida a la nueva posición dada, ver figura 13, obteniendo una nueva secuencia vecina a la anterior, ver figura 14.

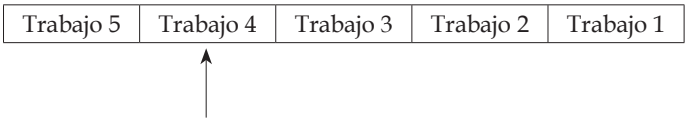

Figura 11.Elección del elemento que se intercambia

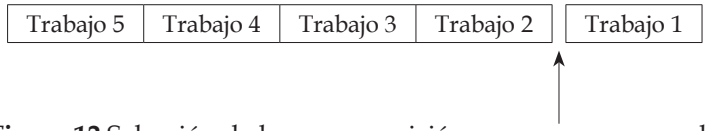

Figura 12.Selección de la nueva posición que va a ser ocupada

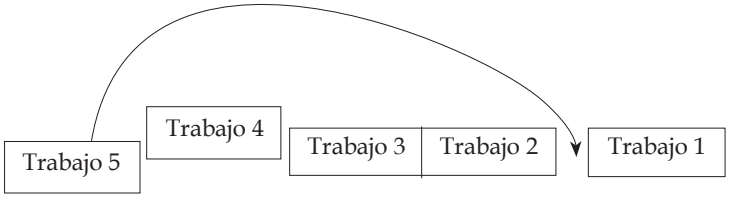

Figura 13.Corrimiento del elemento a la nueva posición

Figura 14.Generación de la configuración vecina

En este trabajo se definió una estructura de vecindad basada en los procesos de intensificación y diversificación de la búsqueda local, creando un método que combina las técnicas anteriormente nombradas sin dar prioridad a alguno de los dos procesos. Se integra una variable aleatoria con distribución normal centrada en 0.5, *next,* Si *next* es mayor que 0.5, la elección será la troca, de lo contrario el corrimiento. El algoritmo utilizado se describe en la tabla 4.

Tabla 4
Algoritmo elección de la técnica para la nueva secuencia

1. Sea *next = rand(0,1)*
2. Si *next >0.5*, Aplique *TROCA*
3. De lo contrario, utilice *CORRIMIENTO*

5. ESTUDIO COMPUTACIONAL

En esta sección se describen los resultados del análisis computacional, se compara la complejidad de los algoritmos y se detalla la implementación.

5.1. **Selección de la muestra**

Se seleccionaron 30 problemas del *Flow-Shop* de la librería de Éric Taillard del sitio web [8]. El tamaño de los problemas tiene un rango que va desde 20 trabajos y 5 máquinas hasta 100 trabajos y 20 máquinas, que ha sido usado en estudios computacionales anteriores. Los problemas elegidos aleatoriamente se presentan en la tabla 5 y fueron clasificados de acuerdo al tamaño; esta clasificación se presenta en la tabla 6.

Tabla 5
Problemas seleccionados aleatoriamente

Número de trabajos	Número de máquinas	Problemas
20	5	ta007, ta008, ta004
20	10	ta012, ta017, ta014, ta020
20	20	ta026, ta022, ta030
50	5	ta032, ta038, ta040
50	10	ta043, ta048, ta049, ta050
50	20	ta052, ta054, ta051
100	5	ta066, ta067, ta070
100	10	ta078, ta077, ta076, ta075
100	20	ta088, ta082, ta087

Tabla 6
Clasificación de los problemas

Problemas pequeños (P)	Problemas medianos (M)	Problemas grandes (G)
ta007	ta032	ta066
ta008	ta038	ta067
ta004	ta040	ta070
ta012	ta043	ta078
ta017	ta048	ta077
ta014	ta049	ta076
ta020	ta050	ta075
ta026	ta052	ta088
ta022	ta054	ta082
ta030	ta051	ta087

5.2. **Comparación de complejidad**

Cuando el $\Delta E < 0$, el esfuerzo computacional requerido para todos los métodos es semejante, al contrario de $\Delta E > 0$. Se puede notar que existen diferencias entre los esfuerzos realizados por cada técnica, esto teniendo en cuenta que las asignaciones similares para todas las técnicas no son tomadas para el estudio. Las comparaciones se muestran en la tabla 7 y las operaciones son clasificadas así *a* (suma y resta), *c* (comparación y asignación), *m* (multiplicación y división), *e* (exponenciación) y *r* (generar un número aleatorio).

La operación de recocido requiere de una multiplicación para planes exponenciales negativos o una resta para planes lineales, mientras que la operación de limitante sólo requiere de una comparación y una asignación.

Tabla 7
Complejidad de los algoritmos aceptación de una solución de peor calidad

Algoritmo	$\Delta E \leq 0$	$\Delta E > 0$ *rechazado*	$\Delta E > 0$ *aceptado*
RS	-	*c, e, m, r*	*c, e, m, r*
AU	-	C	*c*
GG	-	C	*c*
DB	*a*	C	*a, c*
DL	*a*	C	*a, c*
DAL	*a*	*4a, c, 2m, r*	*5a, c, 2m, r*

Continúa...

DR	-	*C*	*a, c*
DAR	*a*	*4a, c, 2m, r*	*5a, c, 2m, r*
DRL	-	*C*	*a, c*
DARL	a	4a, c, 2m, r	5a, c, 2m, r
DRH	a	4a, c, 2m, r	5a, c, 2m, r
DRLH	a	4a, c, 2m, r	5a, c, 2m, r

5.3. **Detalles de la implementación**

Cada algoritmo fue escrito en MATLAB ® 7.0 y todo el trabajo computacional se realizó en una máquina con unas especificaciones cercanas a una CPU Pentium(R) 3,0 GHz y 504 MB de RAM.

Como se recomienda en la literatura, se determinó una única variable de parámetro a los once algoritmos, que fuera capaz de arrojar resultados satisfactorios para todos los problemas de estudio. La idea central era encontrar un parámetro que fuera lo suficiente fuerte para resolver cada instancia del problema y así a la vez probar la robustez del método. A fin de obtener unos valores eficientes que sirvieran en este estudio se recaracterizaron los parámetros; tanto los originales como los recaracterizados son dados en la tabla 8. Estas caracterizaciones son un intento de estandarizar los parámetros de los 11 algoritmos en el problema de secuenciacióndе tareas.

Tabla 8
Parámetros originales y parámetros recaracterizados

Algoritmo	**Parámetros originales**	**Parámetros recaracterizados por**
RS	L_0, α, β	U, D, D_0, $D_{m0} = pm \times E_{inicial}$
AU	U	$D_{Sd} = pd \times E_{inicial}$
GG	D	α α
DL	D_0	β β
DAL	D_{M0}, D_{Sd}	$L_0 = pm \times N_{trabajos}$
DR	D_0, α	
DAR	D_0, α	$E_{inicial}$ *es el makespan de la secuencia inicial*
DRL	D_{M0}, D_{Sd}, α	α, β *valores para el enfriamiento programado*
DARL	D_{M0}, D_{Sd}, α	$N_{trabajos}$ *número de trabajos del problema*
DRH	D_0, D_{Sd}, α, β	
DRLH	D_{M0}, D_{Sd}, α, β	

Para encontrar los valores de los parámetros, se escogieron 3 problemas de diferente tamaño; se crearon parámetros para cada uno; se recombinaron, y se seleccionó uno de ellos a fin de obtener un parámetro que entregara resultados de buena calidad. Los valores de los parámetros resultantes de los experimentos computacionales de la recombinación son presentados en la tabla 9.

Tabla 9
Valores finales de parámetros

Algoritmo	**pm**	**pd**	**α**	**β**
RS	2		0.97	1.06
AU	.2		.9999	
GG	0.001			
DL	0.001		0.97	
DAL	0.001	0.002		
DR	0.001		0.97	
DAR	0.001	0.002	0.97	
DRL	0.001		0.97	
DARL	0.001	0.002	0.97	
DRH	0.001	0.002	0.5	0.8
DRLH	0.001	0.002	0.8	0.8

Cada algoritmo fue ejecutado 20 veces para todos los 30 problemas, usándose los parámetros encontrados con la recombinación realizada experimentalmente. Como la secuencia inicial fue dada mediante una función aleatoria, todas las corridas de cada algoritmo tienen una semilla diferente. La solución entregada por cada ejecución fue medida con relación a la solución óptima y se compararon los resultados entre los 11 algoritmos.

4.4. **Resultados computacionales**

En esta sección se presenta la discusión de los resultados obtenidos al ejecutar los 11 algoritmos implementados para los problemas de Eric Taillard tomados de la referencia [8] de la cual se tomaron 30 problemas en forma aleatoria, que fueron resueltos 20 veces para cada uno de los algoritmos propuestos. Los parámetros usados son presentados en la tabla 9. Estos son el resultado de un parámetro único y que será usado en todos los problemas

y con todos los algoritmos. Para su cálculo fue necesario llevar a cabo un estudio inicial que involucrara todas las técnicas y los problemas.

Los resultados obtenidos se presentan en la tabla 10. El análisis se realiza por grupos teniendo en cuenta: media, desviación estándar e intervalo de confianza. Adicionalmente se calcula la media total a fin de observar el desempeño global del algoritmo, mediante la expresión:

$$Error\ \ medio\ \ total\ =\ \sum_{i=1}^{10\,problemas}\ \sum_{j=1}^{20\,ejecuciones}\ \frac{Solicitud\ \ obtenida_{i,j} - Soluci\text{—}n\text{—}ptima_i}{Soluci\text{—}n\text{—}ptima_i} * 100\%$$

En la tabla 10 se presentan los errores muestrales de las instancias para cada algoritmo, la desviación estándar y el intervalo de confianza a un 95%, además del tiempo total de ejecución.

El error muestral es la desviación de la respuesta obtenida con respecto al óptimo. Los mejores resultados observados son los siguientes:

P el algoritmo con mayor precisión se observa en el RS.
M en esta categoría se presenta un empate entre los algoritmos DRL y DARL.
G el algoritmo con mayor precisión se observa en el AU.

El algoritmo con mejor comportamiento en el contexto global (T) es el RS porque presenta el menor error y la menor desviación estándar.

Tabla 10
Estadísticos descriptivos del estudio computacional

	Descriptores estadísticos	**RS**	**AU**	**GG**	**DL**	**DAL**	**DR**	**DAR**	**DRL**	**DARL**	**DRH**	**DRLH**
P	Media	0,5753	0,7568	1,0594	1,3553	0,8177	1,2559	0,8633	1,1880	1,1688	1,2608	1,3181
	Desviación estándar	0,3844	0,4135	0,5825	0,7285	0,4740	0,6765	0,4589	0,6843	0,7101	0,8316	0,7604
	Intervalo de confianza	0,0076	0,0082	0,0115	0,0144	0,0093	0,0134	0,0091	0,0151	0,0157	0,0164	0,0150
M	Media	1,3732	1,4903	1,4225	1,6351	1,3848	1,5185	1,2169	1,2063	1,2063	1,7552	1,8877
	Desviación estándar	1,5080	1,7946	1,6057	2,0475	1,8126	1,7931	1,4858	1,4947	1,4947	2,1038	2,1115
	Intervalo de confianza	0,0299	0,0356	0,0318	0,0406	0,03594	0,0355	0,0294	0,0296	0,0296	0,0417	0,0418

Continúa...

G	Media	1,1179	1,1087	1,1938	1,3722	1,5772	1,2344	1,5927	1,1980	1,5796	1,6020	1,6174
	Desviación estándar	1,2245	1,3147	1,2732	1,5538	1,7298	1,3989	1,9150	1,4611	1,9406	1,9237	1,9135
	Intervalo de confianza	0,0242	0,0261	0,0266	0,0324	0,03615	0,0292	0,0400	0,0289	0,0384	0,0381	0,0379
T	Media	1,0222	1,1186	1,2252	1,4542	1,2599	1,3363	1,2243	1,1974	1,3182	1,5394	1,6077
	Desviación estándar	1,0390	1,1743	1,1538	1,4433	1,3388	1,2895	1,2866	1,2134	1,3818	1,6197	1,5951
	Intervalo de confianza	0,0206	0,0233	0,0233	0,0291	0,02716	0,0260	0,0261	0,0245	0,0279	0,0321	0,0316
Tiempo ejecución (horas)		52,9336	57,2537	49,9019	52,9534	60,5980	54,1319	56,5643	72,9177	53,7458	43,3492	46,7797

En la figura 15 se presentan los resultados de los algoritmos con mejor calidad de respuesta y mejores tiempos de ejecución.

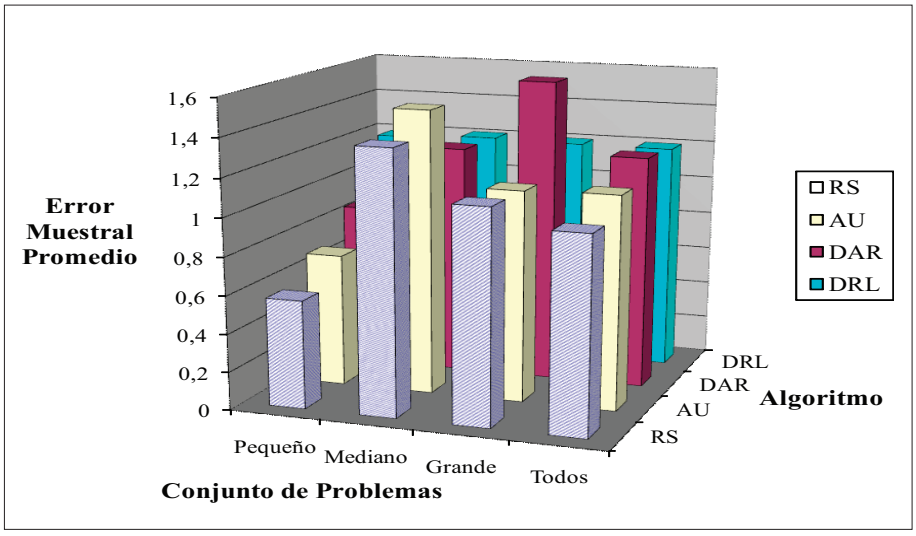

Figura 15. Gráfica de los algoritmos con mejor comportamiento

6. CONCLUSIONES Y RECOMENDACIONES

Se ha resuelto el problema de secuenciación de tareas (*Flow-Shop*) usando 11 versiones basadas en recocidos y clasificadas en recocido simulado, recocido determínistico y algoritmo *demons*, con los que se obtienen resultados de gran interés académico.

En este estudio se obtuvieron en algunos de los casos de prueba respuestas de mejor calidad a las reportadas en la referencia [8].

En la literatura especializada solo se encuentran implementaciones al *Flow-Shop* usando la técnica del recocido simulado; por lo tanto, la inclusión de las otras 10 técnicas es un aporte para la solución de este problema.

Fueron implementadas, además del *Simulated Annealing*, 10 versiones de algoritmos basadas en *annealing*, en las cuales se busca determinar el número de parámetros que mejor comportamiento presenten en los algoritmos. Así, por ejemplo, con el algoritmo AU se requiere de un solo parámetro de calibración y se obtienen resultados similares al algoritmo del RS. En este sentido el AU podría facilitarles a las personas con poca experiencia la solución de este problema de forma adecuada.

En trabajos futuros se podría intentar aplicar la metodología propuesta al problema de *Job-Shop* donde el secuenciamiento de tareas es diferente para cada producto.

Se propone, además, llevar a cabo la implementación del problema usando la optimización multiobjetivo, que presenta entre los objetivos sugeridos: tiempo de flujo, tardanza media, *makespan*.

BIBLIOGRAFÍA

[1] R. Gallego, A. Escobar and E. Toro, *Técnicas metaheurísticas de optimización*, 2nd Ed., Pereira: Universidad Tecnológica de Pereira, 2008.

[2] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing", *Science*, vol. 220, pp. 671-680, 1983.

[3] G. Dueck, "New optimization heuristics: The great deluge algorithm and the record-to-record travel," *J. Computat. Phys.*, vol. 104, pp. 86-92, 1993.

[4] G. Dueck and T. Scheuer, "Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing," *J. Computat. Phys.*, vol. 90, pp. 161-175, 1990.

[5] I. Wood and T. Downs, "Fast optimization by demon algorithms," in *ACNN í98; 9th Australian Conf. Neural Networks*, 1998.

[6] ––– "Demon algorithms and their application to optimization problems," in *IEEE World Congr. Computational Intelligence*, pp. 1661-1666, 1998.

[7] J. Pepper, B. Golden and E. Wasil, "Solving the Traveling Salesman Problem with Annealing-Based Heuristics: A Computational Study," *IEEE Trans. Syst., Man, Cybern. A,* vol. 32, pp. 72-77, 2002.

[8] É. Taillard, "Sheduling instances", *mistic.heig-vd.ch.* [Online]. Available: http://ina2.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html. [Accessed: Feb. 26, 2009].