UNIVERSIDAD DE SONORA

DIVISIÓN DE CIENCIAS EXACTAS Y NATURALES

DEPARTAMENTO DE MATEMÁTICAS

CIENCIAS DE LA COMPUTACIÓN

Algoritmo Genético para Calendarización en un Caso Especial del Flujo de Tareas con Dos Etapas

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

Presenta:

Yair Castro García

Hermosillo, Sonora. Diciembre del 2007

Universidad de Sonora

Repositorio Institucional UNISON

Excepto si se señala otra cosa, la licencia del ítem se describe como openAccess

UNIVERSIDAD DE SONORA

DIVISIÓN DE CIENCIAS EXACTAS Y NATURALES

DEPARTAMENTO DE MATEMÁTICAS

CIENCIAS DE LA COMPUTACIÓN

Algoritmo Genético para Calendarización en un Caso Especial del Flujo de Tareas con Dos Etapas

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

Presenta:

Yair Castro García

Dr. Carlos Alberto Brizuela Rodríguez — Director de tesis

Dr. Claudio Alfredo López Miranda — Asesor de tesis

Hermosillo, Sonora. Diciembre del 2007

RESUMEN de la tesis de **Yair Castro García**, presentada como requisito parcial para obtener el grado de LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

Algoritmo Genético para Calendarización en un Caso Especial del Flujo de Tareas con Dos Etapas

Resumen aprobado por:

Dr. Carlos Alberto Brizuela Rodríguez
Director de Tesis

Dr. Claudio Alfredo López Miranda
Asesor de Tesis

El problema de calendarizar una secuencia o flujo de tareas es de gran interés en la industria, por ejemplo, en procesos de manufactura, robótica y transferencia de datos en red. Una adecuada programación de tareas permite el buen funcionamiento de cualquier sistema, manteniendo la eficiencia y el control sobre las operaciones. Generalmente los objetivos de optimización de este problema son: minimizar el retraso total, minimizar el tiempo máximo de finalización de las tareas (*Cmax*) y minimizar el número de trabajos retrasados. Un caso típico de aplicación en esta tesis serán las líneas de producción en serie, donde se tiene una máquina en una primera etapa y *m* máquinas idénticas en una segunda etapa, conocido en la literatura como el problema de flujo de tareas de dos etapas con (1 + *m*) máquinas (*Flexible Flow Shop*, FFS).

Cuando el número de máquinas es tres o más, el problema de flujo de tareas es del tipo NP-difícil, para los cuales existen diferentes técnicas de solución como los algoritmos de Ramificación y Acotamiento que tratan problemas de tamaño pequeño (15 trabajos y 5 máquinas en la segunda etapa), así mismo, los algoritmos basados en programación dinámica, o bien heurísticas de tipo voraz. Sin embargo, el costo computacional de dichos algoritmos es excesivo conforme se aumenta el número de tareas o máquinas. Aunque las técnicas exactas garantizan la solución óptima, esta puede no ser alcanzable en la práctica, por esta razón proponemos el uso de los Algoritmos Genéticos (AG), los cuales pueden sacrificar la optimalidad sin perder de vista la calidad en la solución.

Con el objetivo de encontrar la mejor combinación de operadores de cruza y mutación para un AG, que utilice una permutación de enteros para representar a la secuencia de tareas, en este trabajo efectuamos una serie de experimentos combinando cinco operadores de cruzamiento (OBX, PPX, OSX, TwoPoint, SB2OX) y tres operadores de mutación (Insert, Swap y Switch). Se realizaron cinco experimentos de AGs para minimizar el retraso total y dos experimentos para minimizar el *Cmax*, en los cuales variamos el número de tareas desde 5 hasta 200 y un máximo de 20 máquinas, con el fin de estudiar cómo afecta el tamaño del problema a la eficiencia de los operadores genéticos, y por ende en el desempeño del algoritmo. Los resultados muestran que el operador OBX es el operador de de cruzamiento que arroja mayor confianza en las soluciones de flujo de tareas en combinación con los operador de mutación Insert y Swap.

Palabras Clave: Algoritmos Genéticos, Calendarización, Flujo de Tareas, retraso total, *Cmax*.

A DIOS Y A MI FAMILIA

Agradecimientos

Sobre todas las cosas agradezco a Dios, por haberme dado la vida y enseñarme en el transcurso de esta, todo lo que se, por todos los triunfos y caídas, por que él ha sido y seguirá siendo el principal motivo para seguir superándome en todos los aspectos. “Gracias Dios”.

Le doy las gracias a mi mamá por inculcarme siempre el estudio, por sus atinados consejos, por ese amor incondicional que me ha demostrado una y otra vez, por hacerme ver que las cosas se pueden realizar si en verdad lo desea uno, por sus invaluables esfuerzos, ..., por ser mi mamá. Su anhelo siempre ha sido que termine yo una carrera, y por fin, ese día ha llegado, con este trabajo culmina su esfuerzo y es el fruto obtenido después de tantos años. Mamá, la corona y los laureles son para ti.

Reconozco que mi hermana también ha sido un gran motivo para culminar con mis estudios universitarios, porque a pesar de su corta edad es digna de tomarla como ejemplo y admiración, además, por mostrarme su cariño y afecto de hermana.

A la Flaka le estoy agradecido infinitamente por llegar a mí en el momento perfecto, por darme a conocer el mundo de los sentimientos, por brindarme su apoyo, por enseñarme que se puede confiar en las personas a pesar de los altibajos, por brindarme seguridad y por muchas cosas más.

A mis directores de Tesis Dr. Carlos A. Brizuela y Dr. Claudio López Miranda les doy las merecidas gracias por dedicarle su valioso tiempo a mis escritos y a mí, gracias por confiar en un joven que ha sido forjado en esta nación. De la misma manera, les reitero mis profundos agradecimientos a Víctor Yaurima y a su familia por su apoyo, orientación y consejos.

Le estoy agradecido a todos mis profesores por trasmitirme su sabiduría y conocimientos, por ponerme a prueba, reprobarme y por motivarme. Es por ellos que he aprendido gran parte de la vida profesional y real, es por ellos que estoy dispuesto a enfrentarme a la vida y al trabajo, ellos me dieron las armas necesarias para hacer frente a los problemas venideros.

A todos mis compañeros de clase y de desveladas, a amigos, a personas que en el transcurso de mi vida han marcado una huella imborrable en mi ser y en mi memoria, les doy las gracias.

A este mundo por tanta belleza, por ser tan bueno con la humanidad, para él es mi admiración y le doy gracias por tanta satisfacción. “En el transcurso de mi vida espero recompensarte por lo menos una parte de lo que me regalas”.

Hermosillo, Sonora, México — Yair Castro García
Diciembre del 2007

Tabla de Contenido

Lista de Figuras

Figura Página

Lista de Tablas

Tabla Página

Tabla de Símbolos

K	Conjunto
n	Escalar
a	Vector
f(x)	Vector de funciones

Capítulo I

Introducción

I.1 Motivación

En este trabajo aplicamos la computación como una herramienta poderosa para resolver problemas en el área de Investigación de Operaciones (IO). Uno de los problemas de gran interés en la IO es el problema de flujo de tareas, conocido en inglés como *Flexible Flow Shop problem* (FFS), en donde uno de los criterios a optimizar ha sido minimizar los tiempos globales de manufactura de productos, como lo muestra Gupta (1988) en un problema de calendarización con dos etapas. La programación de operaciones y particularmente la optimización del tiempo de secuenciación de las tareas que conllevan los procesos de esas operaciones, es el tema en el que nos enfocamos en este trabajo. Esta no es una tarea fácil si tenemos en cuenta una serie de aspectos como las múltiples soluciones sub-óptimas que puede generar el problema y lo difícil que es llegar a obtener el óptimo global del mismo, como se ve en el trabajo de Sriskandarajah y Sethi (1989), quienes abordan el problema de flujo híbrido de tareas con dos etapas: una máquina en la primera etapa y múltiples máquinas en la segunda etapa, dicho trabajo es importante porque allí se establecen cotas inferiores para diferentes heurísticas.

Una adecuada programación de tareas permite el buen funcionamiento de cualquier sistema, manteniendo la eficiencia y el control sobre las operaciones, por lo que en este

trabajo se plantea como problema de investigación, encontrar una secuencia óptima de n tareas, de tal forma que el retraso total (*T*, *tardiness*) o el tiempo de finalización de todas las tareas (*Cmax, makespan*) sea el mínimo posible, en un ambiente de flujo de tareas flexible (FFS). En nuestro sistema FFS de interés, las máquinas están configuradas de la forma $1+m$, es decir, dos etapas sucesivas de máquinas en donde la primera etapa tiene sólo una máquina y la segunda etapa tiene m máquinas idénticas que trabajan en paralelo. El número de tareas n debe ser mayor al número m de máquinas. El problema planteado se clasifica como NP-difícil (NP-Hard), en base al análisis presentado en (Gupta., 1988). Desde el punto de vista de optimización combinatoria, el problema de flujo de tareas consiste en encontrar una permutación de las n tareas que permita minimizar algún criterio que mida la calidad del calendario. Este es un problema típico de secuenciación de tareas, y para encontrar el calendario óptimo nos apoyamos en los Algoritmos Genéticos (AG), dados a conocer por Holland (1975), padre de dicha heurística.

I.2 Antecedentes

Los problemas de calendarización de flujo de trabajos (tareas), han sido ampliamente estudiados en la literatura, por ejemplo en Dudek *et al.*, (1992). Los investigadores pioneros desde Johnson (1954) han considerado varios criterios a optimizar, tales como el tiempo de finalización total de las tareas (*makespan*), tiempo total del flujo de las tareas (*total flow time*), retraso máximo (*maximum tardiness*), entre otros. En general, no es viable encontrar el calendario óptimo para un número grande de tareas utilizando la fuerza bruta. Es bien sabido que las metaheurísticas de aproximación son alternativas eficientes para esta

clase de problemas (Nawaz *et al*., (1983), Park *et al*., (1984) y Chen *et al*., (1996)), tales como recocido simulado (Osman y Potts (1989) e Ishibuchi *et al*., (1995)), búsqueda tabú (Taillard (1990) y Ben-Daya y Al-Fawzan (1998)) y algoritmos genéticos (AG) (Reeves (1995), Murata *et al*., (1996) y Dimopoulos y Zalzala (2000)). Estas técnicas de aproximación ayudan a encontrar una solución cercana a la solución óptima global, es decir, producen soluciones sub-óptimas.

El problema de flujo de tareas se investiga desde los años 50 con la aportación de Johnson (1954), que proporciona un resultado óptimo para dos máquinas. Sin embargo, cuando el número de máquinas es tres o más, el problema es del tipo NP-difícil como lo muestra Gupta (1988). Una de las primeras publicaciones que trata el flujo de tareas híbrido de dos etapas, con una máquina en la primera etapa y múltiples máquinas en la segunda etapa, se debe a Sriskandarajah y Sethi (1989), que posteriormente en el mismo artículo abordan el problema con el mismo número de máquinas en las dos etapas y utilizan una variante del algoritmo de Johnson aplicada a la descomposición del problema en *M* sub-problemas de taller de flujo simple (*flowshop*). Más adelante Gupta y Tunc (1991) extienden el trabajo de Gupta (1988) para el caso de una máquina en la primera etapa y máquinas múltiples en la segunda etapa, probándose que ofrece resultados óptimos para el caso en el que el número de máquinas en la segunda etapa sea mayor que el de trabajos. Cuando no se cumple la condición anterior, se proponen dos heurísticas y también el uso del algoritmo de Ramificación y Acotamiento (*Branch & Bound*) para abordar problemas de tamaño pequeño. El objetivo es minimizar el tiempo máximo de finalización de los trabajos. Gupta y Tunc (1998) vuelven a atacar el mismo problema pero ahora el objetivo es minimizar el número de trabajos retrasados (*tardy jobs*). En dicho artículo se proponen

seis métodos heurísticos y se hace una extensa comparación entre ellos. Riane *et al*., (2002) tratan el mismo problema, minimizar el *makespan* con una máquina en la primera etapa y dos máquinas diferentes en la segunda. Ellos formulan el problema como uno de programación dinámica, adoptan enfoques de Johnson (1954) e incluyen una heurística del tipo voraz (*greedy*). El objetivo es minimizar el máximo tiempo de finalización de los trabajos (*Cmax*). Lin y Liao (2003) aplican su investigación a una empresa de manufactura de etiquetas, la configuración del taller de flujo es de dos etapas y la característica es el tiempo de preparación dependiente de la secuencia en la primera etapa, múltiples máquinas en la segunda etapa y dos tiempos de finalización (*due dates*). El objetivo es minimizar la máxima tardanza ponderada (*weighted maximal tardiness*). Chang *et al*., (2004), investigan el taller de flujo híbrido de dos etapas; una máquina en la primera etapa y múltiples máquinas en la segunda etapa. El objetivo es minimizar el *makespan*. No se permiten tiempos de espera (*no-wait*). Los tiempos de preparación (*setup time*) y liberación del trabajo (*removal time*) se consideran independientes al tiempo de procesamiento. Se proponen dos heurísticas que ayudan a resolver este problema. Lee y Kim (2004), consideran un flujo de tareas híbrido con dos etapas cuando hay una máquina en la primera etapa y múltiples máquinas idénticas en la segunda etapa. El objetivo es minimizar el retraso total de trabajos (*minimizing total tardiness*). Se determina una cota inferior para crear las fechas de finalización. Se propone un algoritmo basado en ramificación y acotamiento, el cual a través de experimentos computacionales puede encontrar soluciones óptimas a problemas de este tipo hasta con 15 trabajos en un tiempo razonable. Guirchoun *et al*., (2005), investigan en el contexto de un servidor y dos máquinas paralelas. Se supone que el tiempo de procesamiento es igual a una unidad sin tiempos de espera y el objetivo es

minimizar el tiempo total para completar todos los trabajos (*minimize the total completion time, makespan*). Se demuestra que el algoritmo propuesto es óptimo para el caso planteado. Por último, Allaoui y Artiba (2006), tratan el problema de taller de flujo híbrido con dos etapas, cuando hay una máquina en la primera etapa y m máquinas en la segunda. Se supone que cada máquina está sujeta a por lo menos un periodo de no disponibilidad y todos los trabajos no pueden ser reanudados si se interrumpen. Proponen un algoritmo de ramificación y acotamiento para minimizar el *makespan*. Se calcula el peor caso de tres heurísticas: algoritmos LIST, LPT y heurística H, propuesta por Lee y Vairaktarakis (1994).

I.3 Planteamiento del Problema

El problema de flujo de tareas consiste en encontrar un calendario óptimo para una colección de trabajos que deben ser procesados en una serie de etapas en donde por cada etapa existe un conjunto de máquinas. Por calendario (organización de las tareas) se entenderá la manera en que se asignan en el tiempo los recursos (máquinas) a los procesos (tareas).

Existen variaciones del problema de flujo de tareas, cada tipo depende de las características de la línea de producción. Uno de ellos es el problema con almacenamiento nulo, es decir no existe espacio o medio (buffer) para almacenar una cola de procesos entre dos máquinas, lo que ocasiona un gran problema, ya que al terminar la primera máquina debe esperar a que la segunda máquina libere el proceso que está efectuando, para así poder

recibir el que está en espera. Sin embargo, existe el caso contrapuesto que simplifica el problema, donde el almacenamiento entre dos máquinas es infinito y puede ser tan grande como se requiera; así, cuando la primera máquina termine de procesar su tarea, lo deposita en el medio de almacenamiento y continúa con el siguiente proceso, cuando la segunda máquina termine el proceso que se encuentra realizando tomará un nuevo proceso del medio de almacenamiento para llevarlo a cabo.

Otro tipo de problema surge cuando no hay un orden único de trabajos (French, 1982). Es decir, una tarea puede pasar de la máquina 1 a la máquina 3 y de ahí a la 7, así sucesivamente sin seguir un patrón. Otro proceso puede empezar en la máquina 3 y después pasar a la 2, pasando posteriormente a otra. Inclusive hay casos en que no todos los productos utilizan todas las máquinas. También hay un tipo de problema donde las máquinas pueden estar repetidas (Ku *et al*., 1993), cuando existen varias copias de una sola máquina (iguales entre sí), por lo que un proceso puede pasar a la máquina 1(a) o a la máquina 1(b); y ambas máquinas pueden estar ocupadas al mismo tiempo. Esta configuración de máquinas se lleva a cabo para dar movilidad a la línea de trabajo cuando existe un proceso que tiene la particularidad de ser lento.

Otra variación del problema de flujo de tareas se presenta cuando existen máquinas multi-funcionales. Este tipo de máquinas pueden realizar el proceso de un conjunto de máquinas. Es decir, pueden realizar el mismo trabajo que la máquina 1, la máquina 2 o la máquina 6, 7, 8 o cualquier otra. Además, estas máquinas pueden estar repetidas. Este tipo de máquinas ayudan a eliminar los cuellos de botella, pero complican más el problema debido a que también deben calendarizarse.

Las tareas podrían tener ciertas características como: precedencia, interrupción de operaciones, tiempos de preparación entre un tipo de tareas y otro, tiempos de espera nulos entre procesos, por mencionar algunos. Los criterios que más se persiguen para optimizar son: minimización del tiempo de completar todos los trabajos, minimización de costos del tiempo de preparación, minimización del tiempo de flujo, minimización del retraso máximo, minimización del número de trabajos retrasados, entre otros.

En general, en los problemas de producción se intenta minimizar el tiempo máximo de finalización de las tareas, también llamado *Cmax* o *Makespan*, que denota el tiempo más corto en el que todas las tareas son procesadas de principio a fin. Por otro lado, el interés principal de la industria puede ser vender cada producto una vez que este sea finalizado. Entonces el objetivo principal es minimizar el tiempo promedio de flujo: el tiempo que transcurre desde que cada artículo está listo para ser fabricado hasta que es terminado por completo. Otro criterio importante en las líneas de producción es el tiempo promedio de retraso. Este tiempo se presenta cuando cada producto tiene que ser finalizado por algún compromiso en un determinado tiempo el cual se conoce como fecha límite (*due date*). Si el producto se termina antes de la fecha límite no hay penalización, de lo contrario si existe dicha penalización. Por lo tanto para éste caso se requiere que los tiempos de retraso sean mínimos.

En general, la minimización del tiempo de finalización total de las tareas no implica la minimización promedio del flujo o del tiempo promedio de retraso. De hecho cuando se tienen dos o más objetivos como estos, y es importante que todos se optimicen al mismo tiempo, entonces el problema se le conoce como Multi-Objetivo.

En este trabajo se estudia el problema de flujo de tareas con dos etapas 1 + *m* (ver Figura 1), las máquinas son idénticas y se considera almacenamiento infinito entre las dos etapas, la interrupción de un trabajo está permitida sólo en el momento de terminación de la primera etapa. Los criterios que nos interesan optimizar son dos: En primer lugar buscaremos minimizar el retraso total (T) de las tareas (*Total tardiness*); y como un segundo criterio independiente, minimizar el tiempo de finalización total de las tareas (*Cmax*).

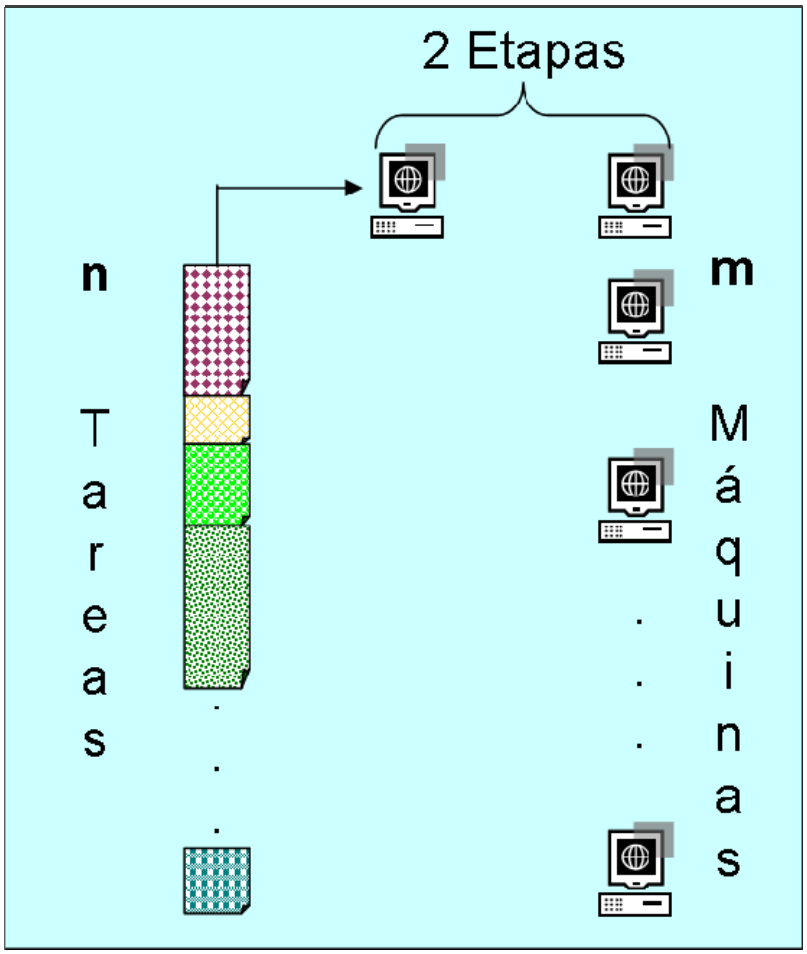


Figura 1: Representación del problema del flujo de tareas con dos etapas de máquinas (1+m).

El problema de flujo de tareas con dos máquinas puede ser resuelto utilizando las reglas de Johnson (Garey y Johnson, 1979), que generan una programación óptima en O($n \log n$) pasos. Desafortunadamente para más de dos máquinas se ha demostrado que el problema se convierte en NP-difícil, para el cual no se conocen (y probablemente no existen) métodos exactos de solución que puedan aplicarse en tiempo polinomial para dimensiones grandes del problema. Es por ello que muchos investigadores enfocan sus trabajos en el desarrollo de nuevos métodos heurísticos, que permitan obtener soluciones de una manera rápida y con calidad, a costa de sacrificar su optimalidad en muchos casos. En el problema que abordamos es difícil obtener soluciones de calidad, por tanto se requieren estrategias de exploración dirigida, debido a que el costo computacional crece exponencialmente al incrementarse el número de tareas y de máquinas en el sistema. Por ejemplo, para procesar 10 piezas (tareas) en cuatro máquinas, tal que en cada pieza se debe realizar una operación por máquina, el número total de combinaciones posibles resulta ser (Companys, 1966):

$$(10!)^4 \approx 1.7 \times 10^{26}$$

Si se considera optimizar solamente el criterio *Cmax*, el problema es NP-difícil (Garey *et al*., 1976). Utilizando una de las computadoras más rápidas, la *Earth Simulator* de NEC en Japón con 5120 procesadores y capaz de alcanzar una velocidad de 35.8 teraflops, equivalentes a 35.8×10^{12} operaciones por segundo, la evaluación de todas las posibles combinaciones se realizaría en 153,590 años. Dicha complejidad nos motivó a usar los AGs como herramienta para la búsqueda de una solución de calidad al problema aquí tratado.

Los AGs, llamados originalmente “planes reproductivos genéticos” fueron desarrollados por John H. Holland (1962), motivado por resolver problemas de aprendizaje de máquina. El algoritmo genético enfatiza la importancia de la cruza sexual (operador principal) sobre la mutación (operador secundario), y utiliza selección probabilística dentro de una población de cromosomas. Sin embargo, los operadores que más le favorecen a cada problema en particular son totalmente desconocidos, así como sus parámetros, los cuales hasta la fecha se encuentran a base de prueba y error o en forma exhaustiva, como lo realiza Ruíz *et al*., (2005). A la cadena binaria (Figura 2) se le llama “cromosoma” y representa una solución potencial del problema. A cada posición de la cadena se la denomina “gene” y al valor dentro de esta posición se le llama “alelo”. Además, un AG puede utilizar representación binaria o decimal para codificar las soluciones a un problema, y se dice que la evolución del algoritmo es a nivel genotipo (representación en el cromosoma). Se ha demostrado en Rudolph (1994) que el AG requiere de elitismo (retener intacto al mejor individuo de cada generación) para garantizar convergencia al óptimo. A diferencia de las técnicas exactas que solo obtienen una solución por corrida, los AGs obtienen varias soluciones por corrida (según el tamaño de población utilizada), además, los AGs han mostrado tener resultados interesantes en trabajos recientes como se puede observar en Ulder *et al*., (1991) y en Syswerda, (1991).

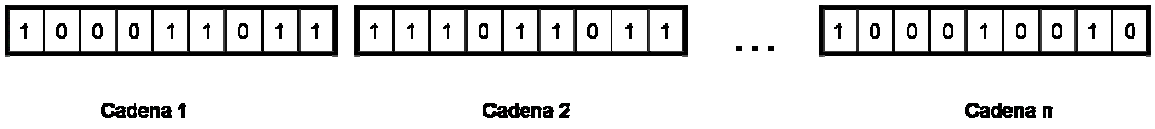


Figura 2: Representación en cadena binaria (cromosoma) utilizada tradicionalmente en los AGs.

I.4 La Metodología

La idea principal de este trabajo es encontrar otra opción viable para resolver el problema de retraso total del flujo de tareas en relación a la presentada por Lee y Kim (2004), quienes resuelven el problema FFS utilizando un algoritmo de ramificación y acotamiento (*branch & bound*), pero limitándose por consumo excesivo de tiempo, a sólo 15 tareas y cinco máquinas en la segunda etapa. Aunque ya existen trabajos donde se comparan diferentes operadores genéticos para el problema de flujo de tareas (Murata *et al*., 1996; Ruíz *et al*., 2005 e Ishibuchi y Shibata., 2004), nosotros pretendemos constatar que los operadores de cruzamiento y mutación, que utilizó Aceves (2003) en su trabajo de tesis "Estudio de Operadores Genéticos para un Problema de calendarización Multi-Objetivo", se comportan de igual manera para este problema de flujo de tareas mono-objetivo y, con base a nuestro estudio, proponer una versión de AG que contemple mayor número de tareas (por ejemplo, 200 tareas y 20 máquinas) que la técnica de ramificación y acotamiento.

Para lograr nuestro objetivo, estudiamos la combinación de cinco operadores de cruzamiento: OBX, PPX, OSX, Two-Point y SB2OX, junto con tres operadores de mutación: insert, swap y switch. Incluimos a OBX e insert ya que en Ishibuchi *et al*. (1995) se concluye que son los operadores que mejor trabajan para permutación de flujo de tareas con representación entera. SB2OX es el operador de cruzamiento que obtiene mejores resultados para el tiempo de finalización de las tareas en el trabajo de Ruiz y Maroto (2006). El resto de los operadores, excepto SB2OX, se tomaron del trabajo de tesis de Aceves (2003) quien realiza una comparación entre todos los operadores mencionados en un problema Multi-Objetivo.

Además, se efectuaron experimentos para el retraso total con casos de distintos tamaños y con todas las posibles combinaciones que surgieron de los operadores de cruzamiento (excepto SB2OX) con los de operadores de mutación, se descartaron los operadores menos eficientes y posteriormente se procedió realizando experimentos con los operadores que mostraron mejores resultados, después, se agrega el operador SB2OX a los experimentos para conocer su desempeño con el retraso. Una vez conocidos los operadores que arrojan mejores resultados, decidimos realizar experimentos con casos en los que únicamente variara el número de máquinas, dejando constantes el número de tareas y tiempos de procesamiento de estas.

También presentamos un estudio para el caso de minimización de *Cmax*. Se efectuaron únicamente dos series de experimentos, la primera serie con los operadores de cruzamiento OBX, SB2OX, operadores de mutación insert, swap y distintos casos. Para la segunda serie se implementaron los mismos operadores, pero los tiempos de procesamiento se dejaron constantes para el mismo número de tareas, sólo se varió el número de máquinas.

I.5 Objetivo Principal

Implementar y comparar diferentes propuestas de operadores de cruzamiento y mutación en un AG utilizado para resolver el problema de calendarización en un ambiente de flujo de tareas flexible de dos etapas con $1 + m$ máquinas.

I.5.1 Objetivos Específicos

1. Investigar el problema de calendarización aplicado al flujo de tareas con 1 + *m* etapas.
2. Comprender la aplicación del cómputo evolutivo para el caso específico de flujo de tareas con dos etapas 1 + *m* máquinas.
3. Proponer e implementar un algoritmo genético para el problema de calendarización para el caso 1 + *m* máquinas.
4. Diseñar casos de prueba para el problema definido de flujo de tareas.
5. Determinar el desempeño de los algoritmos estudiados sobre los casos de prueba generados y compararlos con resultados disponibles en la literatura.

I.6 Organización del trabajo

El presente trabajo de tesis cuenta con cinco capítulos, el primero ya descrito y los cuatro restantes se encuentran organizados de la siguiente manera: En el Capítulo 2 se presenta el problema de flujo de tareas y se dan las definiciones del modelo matemático que contiene las restricciones y sus correspondientes nomenclaturas; además, se definen las funciones a minimizar, se enfatiza la dificultad del problema y se presenta la forma en que trabaja un AG en el sistema 1 + *m*. El Capítulo 3 introduce los algoritmos genéticos estudiados y se muestra la manera en que trabajan. A partir de su discusión, se propone el AG para el problema 1 + *m*, donde se describen los operadores de cruzamiento y mutación a implementar. Adicionalmente se presenta una reseña de los AGs en calendarización. Posteriormente en el Capítulo 4 presentamos el diseño de experimentos y mostramos los

resultados obtenidos. En dicho capítulo probamos los AGs propuestos con diferentes casos tanto para el *Cmax* como para el retraso total (*tardiness*); además, presentamos un análisis de resultados y algunas discusiones breves. El Capítulo 5 presenta las conclusiones a las que se han llegado con este trabajo y las perspectivas de investigación futuras. Finalmente, se presenta la bibliografía y los apéndices.

Capítulo II

Problema de Flujo de Tareas con 1 + *m* Máquinas

En el presente capítulo definimos la nomenclatura del problema de flujo de tareas con (1+*m*) máquinas con sus respectivas restricciones. Además, presentamos las dos funciones objetivo a optimizar con los AGs, estas funciones son el retraso total *T* (*tardiness*) y el tiempo máximo de finalización de todas las tareas *Cmax*.

II.1 Definición del Problema

A continuación definiremos el problema que se abordará en este trabajo de tesis. Un escenario típico para el problema de flujo de tareas son las líneas de producción en serie, donde se tiene una máquina de un tipo en una primera etapa y varias máquinas de un segundo tipo que realizan las mismas funciones en una segunda etapa, es decir, son máquinas idénticas que deben producir varios artículos (trabajos). Cada artículo tiene un tiempo de procesamiento distinto y además necesita pasar por ambas etapas de máquinas para poder ser terminado. El problema es determinar un calendario óptimo que satisfaga todas las condiciones impuestas por las tareas y máquinas en cada una de las etapas, de tal forma que se minimice uno o más de los siguientes criterios: retraso total de las tareas o el tiempo máximo de finalización de dichas tareas. Aunque, en nuestro caso, es necesario recalcar que el problema de flujo de tareas aquí tratado está limitado al caso Mono-Objetivo, es decir, cada objetivo a optimizar constituye un problema independiente.

II.2 Modelo Matemático

Los detalles y restricciones del problema de flujo están dados como sigue:

Sea K_n el conjunto de los n primeros números naturales, i.e. $K_n = \{1,2,...,n\}$. Se investiga un modelo de construcción de un calendario S óptimo con respecto al tiempo para ejecución de un conjunto con n tareas $(n \geq 1)$, que deben ser procesadas en un sistema con dos etapas, configuradas con $1+m$ máquinas $(n > m)$. Tal que, la única máquina de la primera etapa es de un tipo y las *m* máquinas de la segunda etapa, que trabajan en paralelo, son de otro tipo e idénticas entre sí. Cada tarea $j \in K_n$ está compuesta de dos operaciones, O_j^1 y O_j^2. La primera operación de la tarea j (O_j^1) es procesada por la primera máquina en P_j^1 unidades de tiempo, la segunda operación (O_j^2) con duración P_j^2 es ejecutada en una de las m máquinas de la segunda etapa. De esta forma cada trabajo *j* puede quedar definido por los tiempos de procesamiento de cada operación, es decir, $j = (P_j^1, P_j^2)$. La ejecución de un mismo trabajo, en un instante, en más de una máquina está excluida. Cualquier máquina no puede procesar al mismo tiempo más que un trabajo. La interrupción de un trabajo está permitida sólo en el momento de terminación de la primera etapa. El número y la duración de los tiempos de procesamiento de los trabajos son conocidos de antemano. La máquina del primer tipo inicia la ejecución de las operaciones a partir del instante cero. A cada tarea j se le asigna un tiempo de disponibilidad r_j (tiempo a partir del cual puede empezar a ser procesada) y una fecha límite d_j (tiempo para el cual debe ser finalizada). En nuestro caso consideramos que los n trabajos llegan a la entrada del sistema al tiempo cero, i.e. $r_j = 0$. A cada trabajo se le asignan recursos en la segunda etapa de acuerdo a una

estrategia de asignación conocida como *list-scheduling*. Se considera que el espacio de almacenamiento para las tareas es infinito y disponible en cualquier momento.

Bajo la estrategia de asignación *list-scheduling*, el conjunto de los n trabajos, $J = \{j_1, ..., j_n\}$, es dividido en m subconjuntos denotados como J_i, tal que, $J = \bigcup_{i=1}^{m} J_i$, $J_i \neq \varnothing$, $J_\mu \cap J_\nu = \varnothing$, $\mu \neq \nu$. J_i es un subconjunto de trabajos, en donde la segunda operación de cada trabajo es asignada a la máquina i de la segunda etapa, $i \in \{1,2,...,m\}$. Dicha estrategia se muestra en la Figura 3 y se puede ver un ejemplo en la Tabla I.

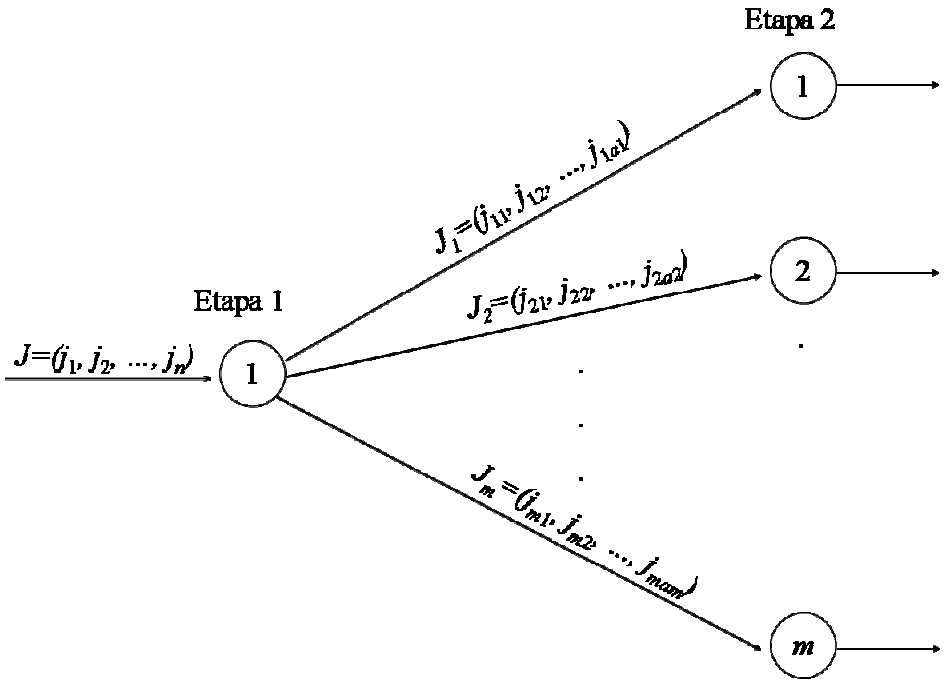


Figura 3: Estrategia de asignación *list-scheduling* para el problema de flujo de tareas con 1 + m máquinas. El total de trabajos es dividido en m subconjuntos tal que $a_1 + a_2 + \cdots + a_m = n$.

Por ejemplo, para el sistema con 1 + 2 máquinas y un conjunto de seis trabajos, definimos $j_1 = (P_{j_1}^1, P_{j_1}^2) = (1,4)$, y análogamente $j_2 = (5,9)$, $j_3 = (3,6)$, $j_4 = (5,5)$,

$j_5 = (5,10)$, $j_6 = (8,7)$ y $J = \bigcup_{i=1}^{2} J_i$. Después de aplicar *list-scheduling* se obtiene la Tabla I con los tiempos de ejecución tal que $J_1 = \{j_1, j_3, j_4, j_6\}$ y $J_2 = \{j_2, j_5\}$.

Tabla I: Tiempo de ejecución de las tareas en las máquinas de cada etapa para una asignacion dada por $J_1 = \{j_1, j_3, j_4, j_6\}$ y $J_2 = \{j_2, j_5\}$.

Etapa	**Máquina**	**Subconjunto**	**Tareas**					
			j_1	j_2	j_3	j_4	j_5	j_6
1	1		1	5	3	5	5	8
2	1	J_1	4	0	6	5	0	7
	2	J_2	0	9	0	0	10	0

Una solución al problema se define como la permutación $\pi = \{\pi(1), \pi(2), ..., \pi(n)\}$, con $\pi(j) \in J$. El objetivo es construir la permutación óptima π^* del conjunto de tareas $\{1,2,...,n\}$ que minimice la función objetivo utilizando *list-scheduling* para la segunda etapa.

Ahora se procede a establecer la notación para las restricciones y cada una de las funciones objetivo a minimizar. El tiempo de inicio de cada operación se denota como s_{kj}, donde $k = \{1,2\}$ representa la etapa y $j = 1,2,\ldots,a_i$ denota una tarea específica en J_i. Cualquier solución factible debe cumplir:

$$s_{kj} \geq r_j \ \forall \ j \in J_i, \ i = 1,...,m$$

En particular, para k=1 (1)

$$s_{1\pi(j)} + P^1_{\pi(j)} \leq s_{2\pi(j)}, \ j \in J_i. \quad (2)$$

No es necesario ilustrar la primera restricción ya que todos los tiempos r_j son igual a cero para toda $j \in J_i$, i.e., todas las tareas están listas para ser procesadas a partir de $t = 0$.

La desigualdad (ec. 2) nos dice que el tiempo de inicio, $s_{1\pi(j)}$, de la operación uno $O^1_{\pi(j)}$, de la tarea $\pi(j)$, más su tiempo de procesamiento $P^1_{\pi(j)}$, debe ser menor o igual que el tiempo de inicio $s_{2\pi(j)}$ de la misma tarea $\pi(j)$ en su operación dos $O^2_{\pi(j)}$.

Todo par de operaciones que pertenecen a tareas consecutivas y procesadas en la misma máquina para la primera etapa deben satisfacer la siguiente restricción:

$$s_{1\pi(j)} + P^1_{\pi(j)} \leq s_{1\pi(j+1)}, \tag{3}$$

La desigualdad (ec. 3) significa que el tiempo de inicio $s_{1\pi(j)}$ de la operación que pertenece a la tarea $\pi(j)$ más su tiempo de procesamiento $P^1_{\pi(j)}$ debe ser menor o igual que el tiempo de inicio $s_{1\pi(j+1)}$ de la siguiente tarea $\pi(j+1)$ en su operación uno $O^1_{\pi(j+1)}$.

La segunda restricción se muestra en la Figura 4, donde se observa que cada tarea debe terminar la operación en la máquina actual antes de pasar a la siguiente máquina (siguiente operación). La tercera restricción se muestra en la Figura 5, donde se muestra que cada máquina debe terminar la operación de la tarea actual antes de continuar con la operación de la siguiente tarea.

Nótese en la Figura 7 que de acuerdo a la regla de despacho *list-scheduling*, la tarea por realizar en la segunda etapa (tarea 2) se asignó aleatoriamente a una de las máquinas desocupadas (máquina 2), pero la tarea 3 se asigna a la máquina con menos trabajo por realizar, en este caso la máquina 1.

Para efectos computacionales, el número total de soluciones o permutaciones para el problema aquí estudiado es (Lee y Kim, 2004):

$$\binom{n-1}{m-1}\frac{(n!)^2}{2}, \tag{4}$$

Con ésta expresión se pudieran mostrar todas las soluciones para 5 tareas y 2 máquinas. A pesar de tener éstas pequeñas cantidades, el número de soluciones posibles es de 28,800. Este número aunque es grande, es alcanzable computacionalmente, sin embargo, en el presente trabajo se estudiaran casos con $n = 200$ y $m = 20$.

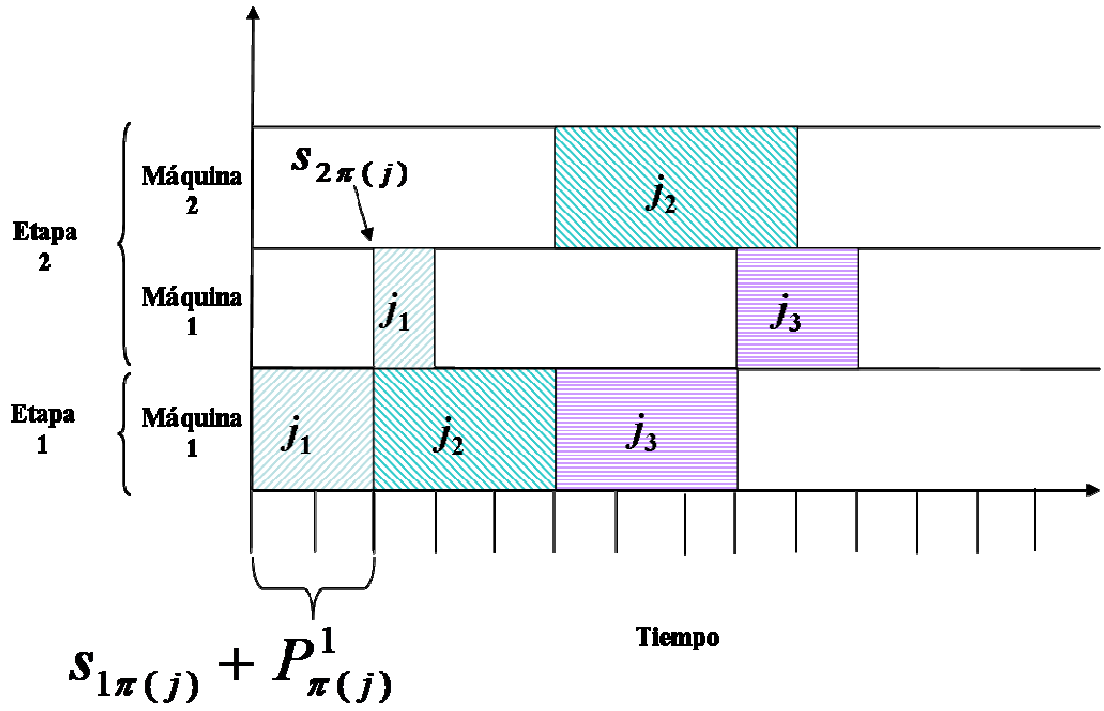


Figura 4: Diagrama de Gantt que ilustra la restricción (2) con tres tareas y dos máquinas en la segunda etapa.

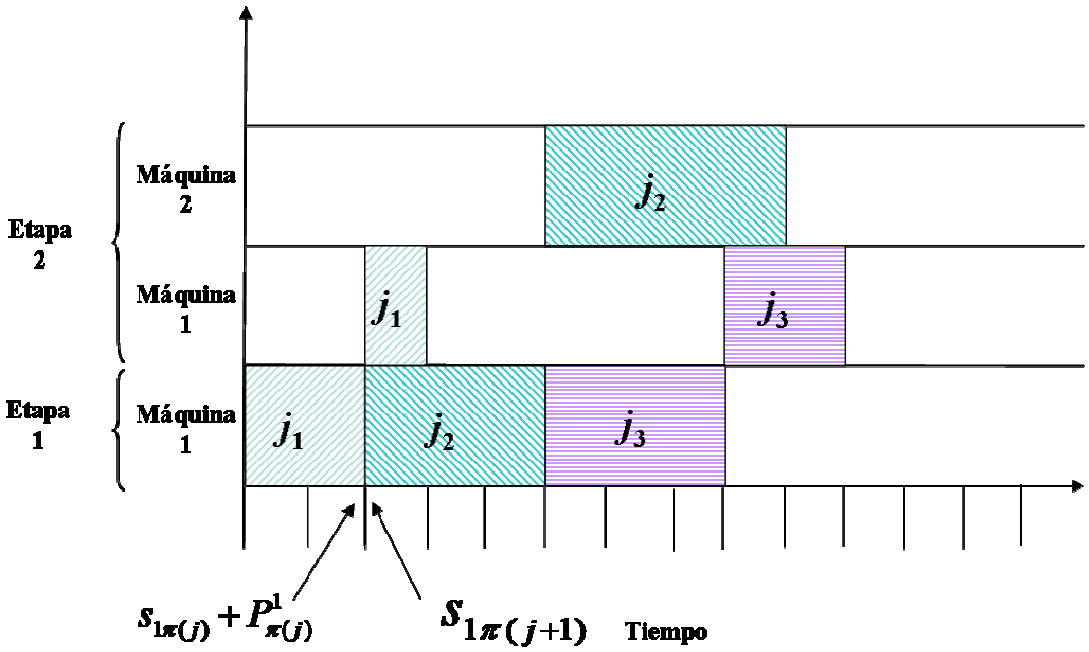


Figura 5: Ilustración de la restricción (3) con dos máquinas en la segunda etapa y tres tareas en donde $k = 1$.

II.2.1 Retraso Total (*Tardiness*)

El retraso para cada trabajo se define como el $\max\{0, C_i - d_i\}$, donde C_i y d_i son el tiempo de finalización y fecha límite de cada trabajo i, respectivamente. En la Figura 6 se ilustra la primera función objetivo (ec. 5) a minimizar, siendo ésta, el retraso total T (*minimizing total tardiness*).

$$Min\ T = \sum_{i=1}^{n} \max(0, C_i - d_i)\,. \tag{5}$$

Al evaluar T con el ejemplo de la Figura 6 se obtiene:

$$T = \max(0, C_1 - d_1) + \max(0, C_2 - d_2) + \max(0, C_3 - d_3).$$

Sustituyendo los valores:

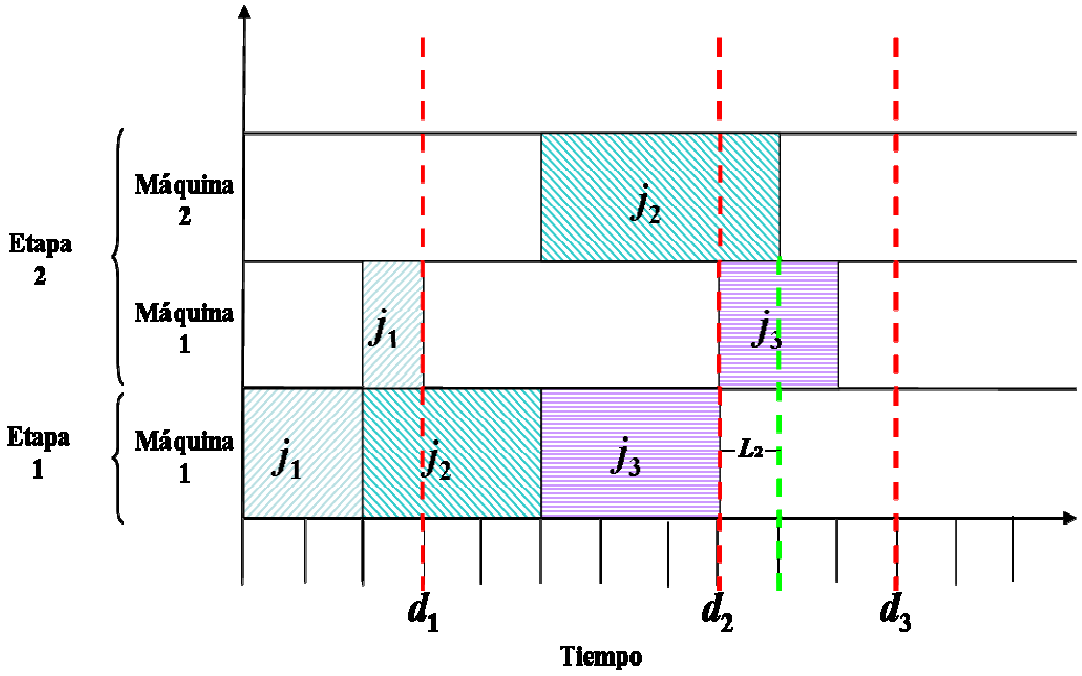


Figura 6: Calendario y fechas límite para cada tarea (d_1, d_2 y d_3) y el retraso único (L_2) relativo a d_2 de la tarea j_2.

$$T = \max(0, 3 - 3) + \max(0, 9 - 8) + \max(0, 10 - 11)$$

$$T = 0 + 1 + 0$$

$$T = 1.$$

El resultado para T es congruente con lo mostrado en la Figura 6 ya que j_2 es el único trabajo que rebasa el tiempo de procesamiento permitido por su fecha límite (d_2). Obsérvese también que j_1 termina sus dos operaciones justo a tiempo de lo permitido por d_1, y j_3 termina una unidad antes de lo indicado por su fecha límite (d_3).

II.2.2 Tiempo Total de Finalización (*Cmax*)

La segunda función a minimizar de manera independiente se ilustra en la Figura 7 y es el tiempo de finalización de la última tarea (ec. 6). El objetivo es encontrar la permutación óptima π^* que minimice *Cmax*.

$$Min\ \ Cmax = \max_i \{C_i\} \tag{6}$$

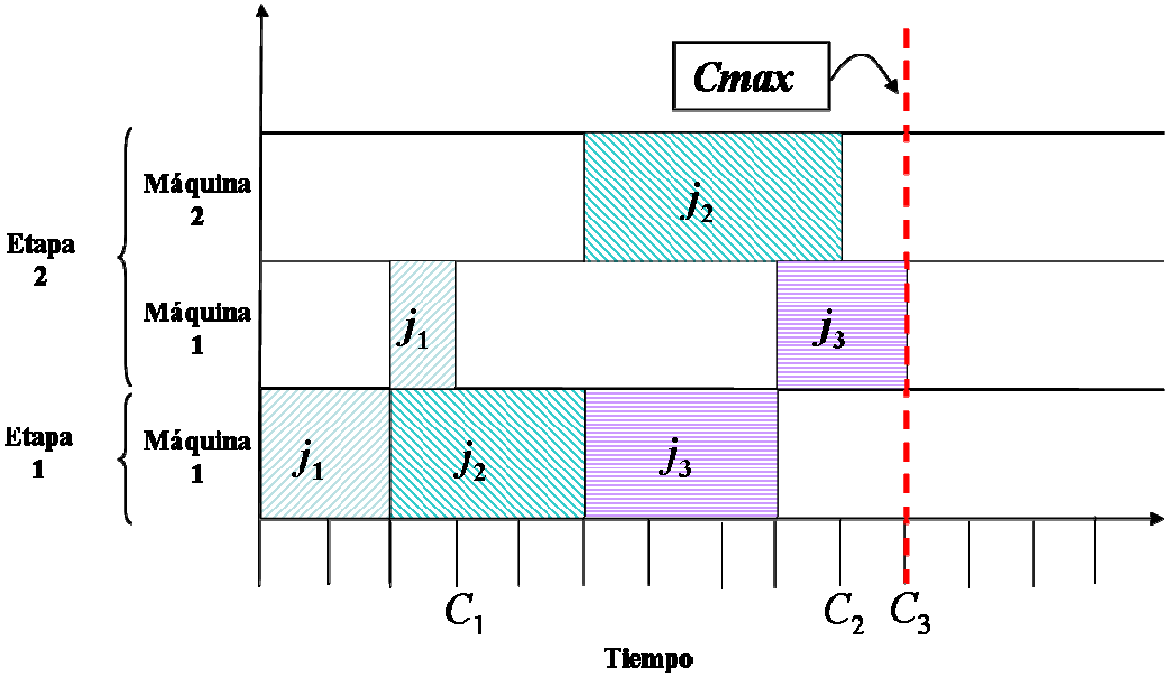


Figura 7: Calendario y tiempos de finalización (C_1, C_2 y C_3) de cada tarea y el tiempo total *Cmax* para una permutación particular, con la configuración de 1 + 2 máquinas y tres tareas.

En resumen, los objetivos a minimizar por separado del problema de flujo de tareas son:

Minimizar T o Minimizar *Cmax*

Sujetos a las restricciones (1) - (3).

II.3 Descripción del Sistema 1 + *m*

A continuación se realiza una descripción del sistema conformado por 1 + *m* máquinas. El flujo de las tareas de este sistema puede ser optimizado por cualquier AG y cualquier estrategia de asignación ya que no existe dependencia entre ambos, es más, se pueden efectuar distintas combinaciones de AGs y estrategias para encontrar la que brinde mejores resultados. El sistema consta de varias partes y para su mejor comprensión se ha dividido en tres fases, en cada una de esta fases se llevan a cabo diferentes procesos. A continuación damos a conocer dichas fases y posteriormente se describen:

- **Fase 1**
    1) Representación de los trabajos.
    2) Ejecución del AG.
    3) Procesamiento de la primera parte del trabajo.
- **Fase 2**
    1) Asignación de recursos a la tarea por medio de alguna estrategia.
    2) Almacenamiento temporal del trabajo.
- **Fase 3**
    1) Procesamiento de la segunda parte del trabajo.
    2) Evaluación.

En la Fase 1 se representan los trabajos: Inicialmente los trabajos se encuentran listos para ser procesados y tienen que ser representados en la notación correspondiente para ser

manipulados con el AG; en el paso 2) se ejecuta el AG para encontrar el calendario que optimiza la función objetivo. Posteriormente en el paso 3), se procesa cada uno de los trabajos en la primera etapa que consiste de una sola máquina, este procesamiento se lleva a cabo atendiendo la permutación (individuo) de las tareas arrojada por el AG en cada una de sus iteraciones. La asignación de recursos para los trabajos se lleva a cabo en el paso 1) de la Fase 2, lo cual significa que por medio de alguna estrategia de asignación se calendariza el trabajo para que sea procesado en la segunda etapa de máquinas. Si el recurso asignado (máquina) se encuentra desocupado, entonces el trabajo pasa directamente a la segunda etapa a procesarse sin mayor retraso, de lo contrario, el trabajo es almacenado temporalmente en un espacio que se considera lo suficientemente grande y disponible en cualquier momento, que viene siendo el paso 2). En la Fase 3 se procesan los trabajos en la segunda etapa de máquinas, los trabajos son tomados del medio de almacenamiento. Después, se continúa con el paso 2): El valor de aptitud de cada uno de los individuos se obtiene al evaluarlos en la función objetivo, esto se lleva a cabo cuando se procesan todas las tareas en el sistema por completo y se obtiene el calendario final. Las secuencias de fases descritas anteriormente se muestran en la Figura 8.

El sistema de flujo de tareas aquí planteado en conjunción con la estrategia de asignación *list-scheduling*, se utilizarán para realizar los experimentos en el Capítulo IV usando el AG presentado en el siguiente capítulo.

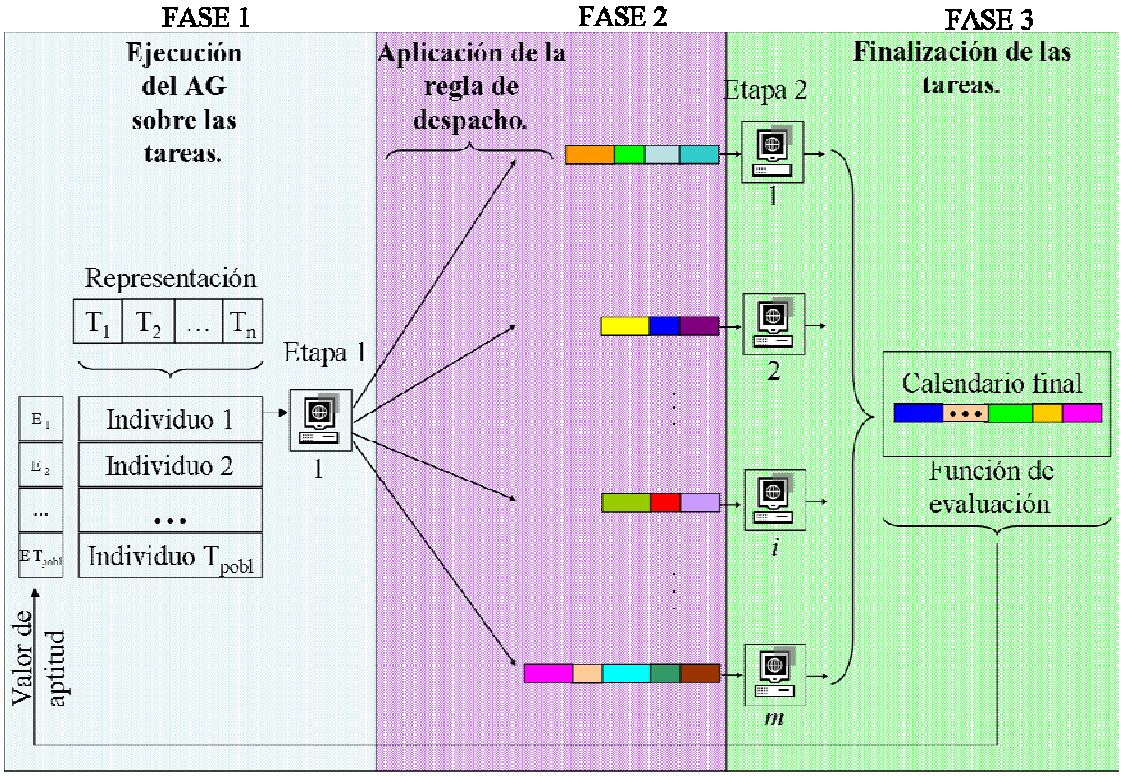


Figura 8: Descripción del funcionamiento del sistema virtual del flujo de tareas para aplicar el AG.

Capítulo III

Algoritmos Estudiados

El presente capítulo se enfoca a describir los algoritmos genéticos (AGs) que aplicaremos en los problemas de flujo de tareas. Se presentan los antecedentes de la aplicación de AGs en calendarización de procesos. En particular, se describe el AG propuesto para el problema 1 + *m* junto con la representación utilizada y los operadores de cruzamiento y mutación.

III. 1 Algoritmos Genéticos

Los AG están compuestos por un multi-conjunto de soluciones que forman la población, y dichas soluciones están codificadas en cromosomas (individuos). Por medio de la función de aptitud se evalúa a cada individuo de la población y se le asigna un valor de aptitud. El individuo con mejor aptitud representa a la mejor solución. La población de tamaño T_{pobl} va evolucionando al aplicarle iterativamente una serie de operadores hasta que se cumpla algún criterio de paro (C_{paro}). Una iteración completa del AG es llamada generación y se puede describir como sigue: Un mecanismo de selección toma algunos individuos (los más aptos) de la población de acuerdo a su valor de aptitud, cada individuo tiene una probabilidad de ser seleccionado. Los individuos seleccionados (padres) se aparean por medio de un proceso llamado cruzamiento y generan un nuevo individuo, al que se le denomina descendiente (hijo), después de la cruza, algunos descendientes pueden

experimentar una mutación. Posteriormente la nueva población es evaluada y todos los procesos son repetidos hasta cumplirse el criterio de paro. En la Figura 9 se observa un bosquejo general de los procedimientos y secuencia del AG estándar.

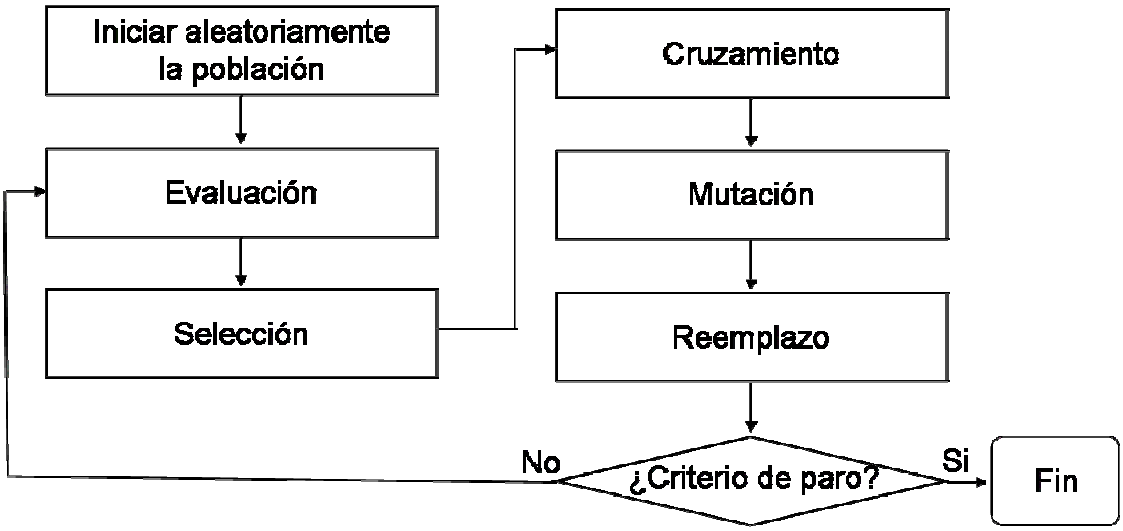


Figura 9: Diagrama de flujo para el AG estándar.

Los AG trabajan con una representación de la solución y no con la solución misma. Se le llama genotipo a la codificación (binaria, entera o decimal) de los parámetros que representan una solución del problema a resolverse, y se le denomina fenotipo a la decodificación del cromosoma, es decir, a los valores obtenidos al pasar de la representación (binaria o entera) a la utilizada por la función objetivo, para un ejemplo véase la Figura 10.

Figura 10: Representación de una cadena binaria utilizada en el AG estándar.

Para una descripción detallada a cerca del funcionamiento de los AGs se puede recurrir a Holland (1975), Goldberg (1989), Gen y Cheng (1997), Mitchell (1996), por citar algunos. Debido a que los AGs son de naturaleza estocástica, se genera un resultado distinto en cada ocasión que se ejecutan, aunque se mantengan las mismas condiciones, dichos resultados se denotan por el vector $x = [x_1, x_2, ..., x_{N_{\exp}}]$, es decir, en N_{exp} corridas se pueden obtener N_{exp} resultados diferentes. Por esta razón el algoritmo es ejecutado un número suficientemente grande de veces ($N_{\exp}$) y posteriormente se calcula el promedio de los resultados ($\bar{x} = \frac{x_1 + x_2 +, ..., + x_{N_{\exp}}}{N_{\exp}}$) con su respectiva desviación estándar (SD_x). De acuerdo a Rudolph (1994), entre mayor sea el número de ejecuciones N_{exp} del AG, mayor es la probabilidad *p* de acercarse al óptimo.

III. 2 Algoritmos Genéticos en Calendarización

Se han hecho numerosos intentos para solucionar el problema de calendarización mono-objetivo por medio de los AGs. Jain y Bagachi (2000); Cartwright y Tuson (1994), Murata *et al.*, (1996). Jin *et al.*, (2002), tratan el problema de tres etapas con máquinas paralelas

idénticas aplicado a la industria de tarjetas de circuito impreso, con el objetivo de minimizar el *makespan*. Proponen un procedimiento global basado en AGs. Bertel y Billaut (2004) investigan un taller de flujo híbrido de tres etapas con recirculación, en el que pretenden minimizar el retraso ponderado de trabajos mediante una formulación de programación lineal, al mismo tiempo, describen un algoritmo voraz (*greedy*) basado en reglas de asignación (*dispatching rules*), así como un AG. Se comparan estos dos algoritmos a través de un experimento computacional. Se concluye que el AG propuesto resulta tener mayor eficiencia. Portmann *et al*. (1998) proponen un algoritmo de Ramificación y Acotamiento. Utilizan un algoritmo genético hibridizado en la fase de ramificación del algoritmo para mejorar las estimaciones de las cotas inferiores y superiores, de esta manera se consigue acotar más el árbol de búsqueda. Wang y Li (2002), consideran el problema de taller de flujo con múltiples etapas, máquinas paralelas idénticas, el objetivo es minimizar el *Cmax*. Usan el LPT (*longest processing time*) para la asignación de los trabajos y proponen un algoritmo genético para la secuenciación. Morita y Shio (2005), consideran el taller de flujo con múltiples etapas para minimizar el tiempo de completar los trabajos. Se propone un método híbrido de ramificación y acotamiento con AG, el cual mejora los métodos basados únicamente en ramificación y acotamiento reduciendo el número de nodos a buscar. Vazquez y Salhi (2005) reportan el desempeño de técnicas para la solución de un taller de flujo flexible usando cuatro variantes de AGs y reglas de despacho (*dispatching rules*) como FIFO (*First In, First Out*), SPT (*Shortest Processing Time*), LPT y SDD (*Shortest Due Dates*), así como el SBP (*Shifting Bottleneck Procedure*). Se prueban con cuatro criterios de optimización: minimización del tiempo máximo de finalización (*Cmax*), minimización del máximo retraso (*maximum tardiness*),

minimización del tiempo de finalización ponderado de todos los trabajos (*total weighted completion times*) y la minimización del retraso máximo ponderado (*total weighted tardiness*). Se concluye resaltando el notable desempeño de la inclusión de los AGs. Ruiz y Maroto (2006), proponen AGs para un taller de flujo híbrido donde las máquinas son no relacionadas, restricciones de elegibilidad de máquinas en cada etapa y se consideran tiempos de preparación (*setup time*) dependientes de la secuencia. Finalmente, en Ishibuchi y Shibata (2004) se examinan siete operadores de cruzamiento: tres versiones de cruzamiento de un punto, tres tipos de cruzamiento de dos puntos y una versión del cruzamiento uniforme. Al mismo tiempo evalúan tres operadores de mutación: switch, swap, shift. Los criterios que minimizan son dos: minimización del máximo retraso y minimización del tiempo de finalización de la última tarea, los mismos operadores y criterios son utilizados para el problema en versión Multi-Objetivo. Con base a los trabajos previos mencionados, nosotros proponemos comparar diferentes operadores de cruzamiento y mutación en el problema de calendarización, en particular, para un ambiente de flujo de tareas flexible de dos etapas con $1 + m$ máquinas.

III. 3 Algoritmo Genético Propuesto

A continuación se describe el algoritmo genético propuesto para el presente trabajo (Figura 11) y posteriormente los operadores de cruzamiento y mutación.

1) Se crea una población aleatoria inicial de T_{pobl} individuos.

2) Se evalúa la aptitud de cada individuo.
3) Se aplica selección por torneo binario.
4) Se toma al individuo con mayor aptitud (súper-individuo) de la población de padres y se guarda (proceso denominado elitismo).
5) De los individuos seleccionados (padres) se toman dos contiguos y se les aplica el operador de cruzamiento con probabilidad Pc, generando así la población (T_{pobl}) de los nuevos hijos.
6) A éstos hijos se les aplica un operador de mutación con probabilidad Pm.
7) Finalmente se reemplaza la población original con la de hijos y el proceso se repite desde el paso 2) hasta satisfacer el criterio de paro.

En el paso 1) se debe crear la población inicial con T_{pobl} individuos y suficiente diversidad para que el AG tenga mayor espacio de búsqueda, al mismo tiempo se trata de evitar la convergencia prematura, es decir, reducir la probabilidad de que el AG se estanque rápidamente en un óptimo local. Cada individuo consta de n elementos, los cuales no se deben repetir, de lo contrario se estaría calendarizando el mismo trabajo en dos ocasiones, esto es diferente a tener dos tareas con los mismos tiempos a procesar, lo cual si puede suceder.

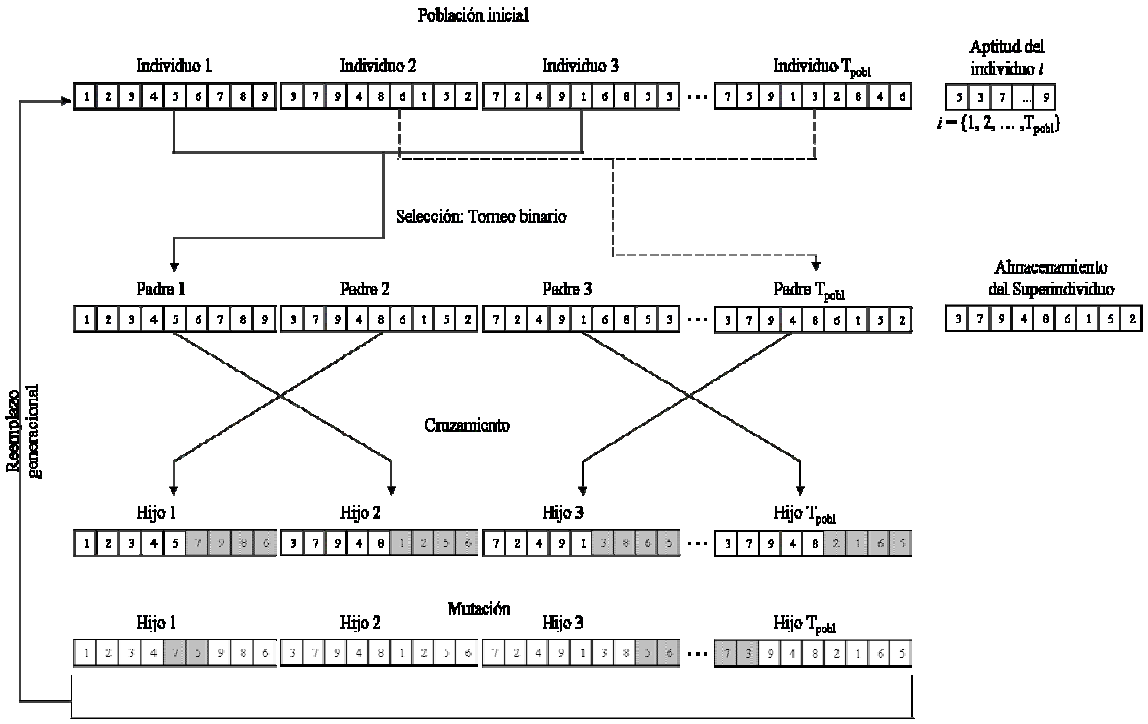


Figura 11: Descripción del funcionamiento del AG para optimizar T o *Cmax*.

Posteriormente, en el paso 2) se calcula la aptitud de cada individuo, dicho valor se obtiene evaluando al individuo en la función objetivo del retraso total *T* (ec. 5), o bien de acuerdo a la función *Cmax* (ec. 6). Para la selección, se toman dos individuos al azar, se confronta su aptitud y el ganador pasa a la siguiente generación, este proceso se repite hasta completar la generación de tamaño T_{pobl}. Este tipo de torneo binario permite que el peor individuo pueda ser seleccionado y trasmitido a la siguiente generación, así como puede suceder lo contrario con el mejor individuo o de igual manera, ambos pueden ser copiados. Para más detalles del torneo binario consultar Coello *et al*., 2003 que realiza una introducción a la computación evolutiva y Brindle (1981), quien efectúa un estudio extenso de esta técnica de selección. El paso 3) de nuestra implementación se lleva a cabo de la

siguiente forma: se toma el valor de aptitud de dos individuos seleccionados al azar, y de estos dos, se copia directamente al individuo con mayor aptitud a la población de padres, esto se efectúa hasta completar la población de padres en su totalidad. En 4), de la población de padres se toma al individuo con mayor aptitud, es decir, al individuo que minimiza el criterio con mayor efectividad y se conserva hasta ser sustituido por un individuo que tenga mejor aptitud en las siguientes generaciones. Cuando se llega al paso 5), se aplica bajo cierta probabilidad (Pc) un operador de cruzamiento a la población de padres (ver Figura 13, Figura 14, Figura 15 y Figura 16). Dicho operador toma a dos individuos contiguos de la población de padres y genera otros dos descendientes que son guardados consecutivamente en la población llamada hijos. Bajo éste criterio el proceso se repetirá $T_{pobl}/2$ veces para completar la población de los padres. Si no se lleva a cabo el cruzamiento entre los individuos, entonces ambos individuos son copiados directamente. Para cuando se llega al paso 6) se aplica un operador de mutación (ver Figura 17) a cada uno de los individuos de la población de los hijos, éste operador se aplica con probabilidad Pm, si no ocurre el evento mutación (la probabilidad generalmente es baja), entonces queda intacto ese individuo en particular. Finalmente, en la generación actual, paso 7), se reemplaza a la población inicial con la población de hijos para poder seguir iterando hasta satisfacer el criterio de paro (C_{paro}). El criterio de paro utilizado para este AG se cumple cuando el súper-individuo no se reemplaza durante un cierto número de iteraciones, es decir, un número de iteraciones sin cambios en su aptitud.

III.3.1 Representación

Los algoritmos genéticos utilizan de manera habitual la representación binaria como lo muestra el ejemplo en la Figura 2. Sin embargo, en el problema de flujo de tareas esta representación no es la adecuada debido a que la solución del problema es una permutación de números enteros. Asignar un mapeo entre una cadena binaria y una permutación es más complicado que trabajar con la permutación misma. Por lo tanto, en el algoritmo nuestra representación es entera, como lo muestra el ejemplo de la Figura 12 con los trabajos $j_1 = (1,4)$, $j_2 = (5,9)$, $j_3 = (3,6)$, $j_4 = (5,5)$, $j_5(5,10)$, $j_6 = (8,7)$, es decir, cada individuo es una permutación de los primeros n números naturales; donde n es el número de tareas. Nótese en la Figura 12 que los tiempos de procesamiento no intervienen en la representación usada para el genotipo. De acuerdo a esta representación tenemos distintos tipos de cruzamiento y mutación.

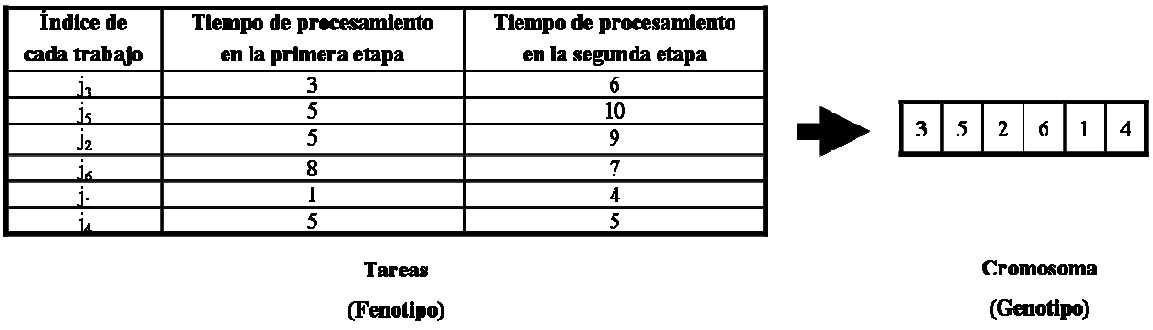


Figura 12: Representación entera (genotipo) de las tareas en el cromosoma.

III.3.2 Operadores de cruzamiento

- **OBX** (Cruzamiento basado en el orden). (Gen y Cheng., 2000). Se genera una máscara aleatoria de unos y ceros, los genes del padre 1 que corresponden con unos de la máscara se copian al hijo en el mismo *locus* (posición), posteriormente los genes faltantes se toman del padre 2 manteniendo el orden en que estos aparecen. La Figura 13 (izquierda) muestra cómo trabaja éste operador.
- **PPX** (Cruzamiento con relación de precedencia). (Bierwirth *et al*., 1996). La idea para este operador es que un subconjunto de relaciones de precedencia de genes de los padres sean preservados en el hijo. En la Figura 13 (derecha) se muestra que los 1's en la máscara indican que los genes del padre 1 serán copiados al hijo y los 0's indican que los genes serán copiados del padre 2.
- **OSX** (Cruzamiento de un segmento). (Gen y Chen, 2000). Se toman dos puntos al azar $s1$ y $s2$ (válidos en el rango del número de tareas), posteriormente, desde el gen en la primera posición hasta $s1$ del padre 1 se copian al hijo en los mismos *loci*. Después, el hijo se sigue llenando desde el *locus* $s1$ hasta el *locus* $s2$ con elementos del padre 2 comenzando por la posición 1 de este último y copiando aquellos genes que no han sido copiados del padre 1. Los genes desde la posición $s2+1$ en el hijo hasta la posición n se copian del padre 1 considerando únicamente los que aún no han sido copiados. Este operador se puede ver en la Figura 14 (izquierda).
- **Two point** (Cruzamiento de dos puntos). (Ishibuchi y Shibata (2004); Coello *et al*., 2003). Se seleccionan dos puntos al azar en ambos padres como se muestra en la Figura 14

(derecha), se copia del padre 1 desde la posición 1 hasta $s1$ y del $s2$ a la posición n en el hijo en las posiciones correspondientes, los genes faltantes se toman del padre 2 considerando los que no han sido copiados.

- **SB2OX** (Cruzamiento de dos puntos que conserva bloques similares en ambos padres). Ruíz *et al*., (2005) especifican que este operador identifica y mantiene bloques. Primero, los bloques comunes de trabajos en ambos padres son copiados a los hijos correspondientes en el mismo orden, como se muestra en la Figura 15 (izquierda); después, se generan dos puntos de corte al azar ($s1$ y $s2$), los genes entre estos *loci* se heredan directamente a los hijos, Figura 15 (centro), finalmente, los elementos faltantes son copiados del padre al descendiente contrario conservando el orden y cuidando que no se repitan dichos elementos, Figura 15 (derecha).
- **OPX** (Cruzamiento con un punto de corte). Se genera un punto de corte al azar $s1$ para ambos padres, a partir de este punto de corte hasta n se copian los genes a los hijos correspondientes como se muestra en la Figura 16 (izquierda), posteriormente, los elementos faltantes en los hijos son tomados del padre contrario en orden de izquierda a derecha y cuidando que no se repitan dichos elementos, como se ilustra en la Figura 16 (derecha). Este operador se describe en Ishibuchi y Shibata (2004) y Coello *et al*., (2003).

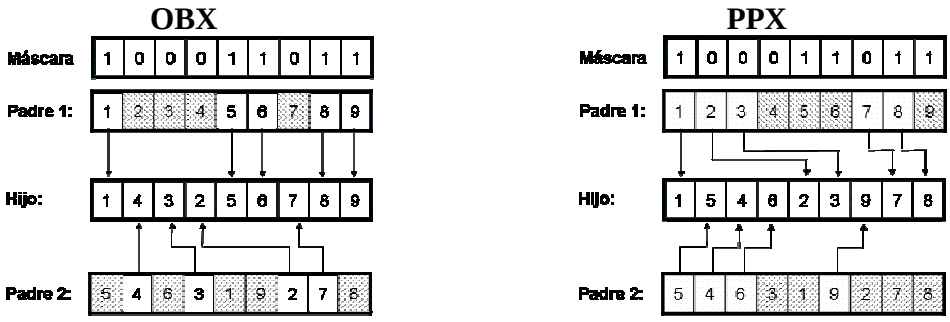


Figura 13: Ejemplos de los operadores OBX y PPX para n = 9.

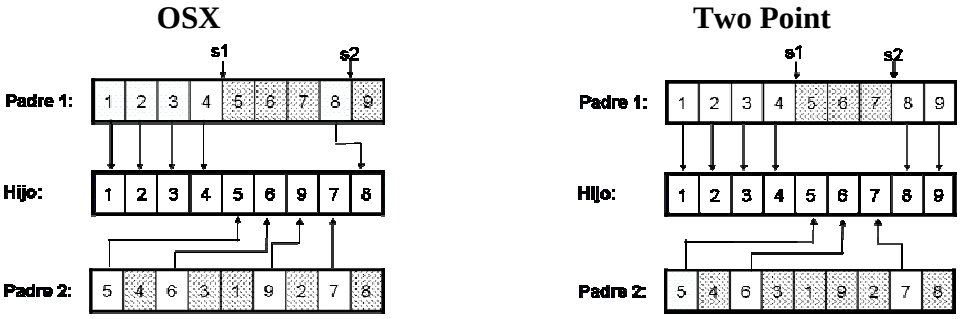


Figura 14: Ejemplos de los operadores OSX y Two point para n = 9.

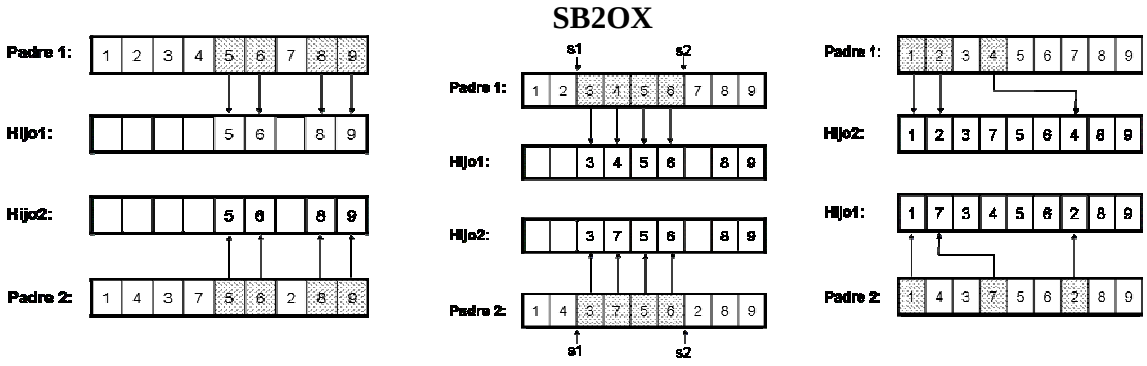


Figura 15 : Cruzamiento de dos puntos que conserva bloques similares en ambos padres (SB2OX).

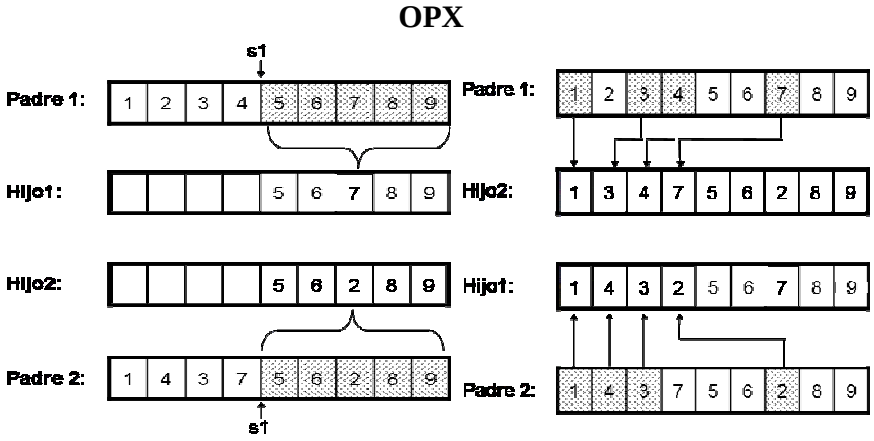


Figura 16: Cruzamiento de un punto (OPX).

III.3.3 Operadores de Mutación

Los operadores de mutación que se utilizan para los experimentos son estándar y son mencionados en Gen y Cheng (1997).

- **Insert** (Shift). Se seleccionan dos *loci* al azar $s1$ y $s2$, si $s1 < s2$ entonces el gen correspondiente a $s1$ se coloca en la posición $s2$ y todos los genes desde $s1$ +1 hasta $s2$ se recorren una posición hacia s1, como se muestra en la Figura 17 (izquierda). Pero si $s1 > s2$ entonces la operación de corrimiento se realiza hacia $s2$. Este operador de mutación también fue utilizado por otros autores en sus correspondientes algoritmos genéticos para el flujo de trabajos (Reeves, 1995 y Murata *et al*., 1996).
- **Swap** (arbitrary two-job change). Se seleccionan dos *loci* al azar y sus genes se intercambian como se muestra en la Figura 17 (centro) (Ishibuchi y Shibata, 2004).
- **Switch** (Adjacent two-job change). Se selecciona el *locus* $s1$ al azar para intercambiarlo con su sucesor inmediato a la derecha, si se selecciona el *locus* n,

entonces se intercambia éste con el *locus* 1. Se muestra un ejemplo de este operador en la Figura 17 (derecha) (Ishibuchi y Shibata, 2004).

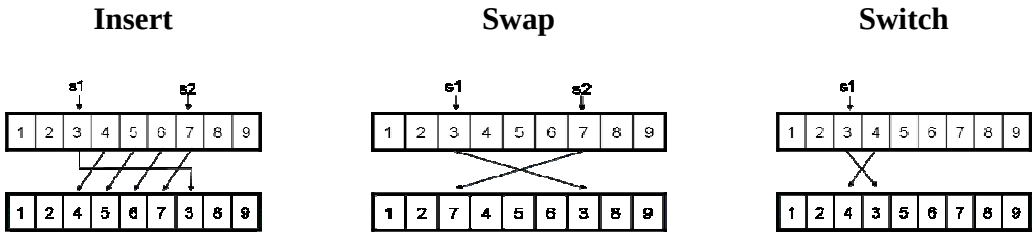


Figura 17: Operadores de mutación insert, swap y switch para $n = 9$.

Una vez conocido el AG propuesto, la representación de las tareas y los operadores genéticos, procedemos a efectuar los experimentos en el siguiente capítulo.

Capítulo IV

Diseño de Experimentos y Resultados

Con el fin de estudiar el desempeño de los algoritmos genéticos desarrollados para el problema de flujo de tareas, en particular el desempeño de los operadores de cruzamiento y mutación, realizamos una serie de experimentos con distinto número de tareas y de máquinas, para analizar tanto el retraso total (*T*) como el tiempo de finalización total de las tareas (*Cmax*). Se busca con esto, analizar la robustez relativa de los operadores cuando se varían los dos parámetros principales (número de tareas y de máquinas).

IV.1 Generación de Casos

Para generar los tiempos de procesamiento de cada tarea requeridos por nuestros experimentos, usaremos el programa de Taillard[1]. Sin embargo, dicho programa no genera fechas límite (*due dates*) para las correspondientes tareas, por lo que optamos generarlas siguiendo el método de Gupta y Tunc (1998), quienes resuelven un problema de flujo de tareas de dos etapas con el objetivo de minimizar el número de trabajos con retraso. El mismo método de obtención de fechas límite es retomado por Lee y Kim (2004) para estudiar un algoritmo de ramificación y acotamiento. El objetivo fue minimizar el retraso total en un problema de flujo de tareas con dos etapas, en donde existe una máquina en la primer etapa y un número de máquinas idénticas en la segunda etapa.

[1] Sitio personal del Profesor Eric Taillard: http://mistic.heig-vd.ch/taillard/

A continuación se presenta la nomenclatura utilizada para obtener las fechas límite.

a_i: Tiempo de procesamiento de la primera operación del trabajo i ($i = 1,2,...,n$), en la primera etapa.

b_i: Tiempo de procesamiento de la segunda operación del trabajo i en la segunda etapa.

m: Número de máquinas en la segunda etapa.

n: Número total de trabajos a ser procesados.

Cada fecha límite se generó (ver ejemplo en la Tabla II) siguiendo la distribución uniforme discreta DU(P × L, P × U), tal que

$$P = \left(\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i / m + (n-1) \cdot \max\left\{ \sum_{i=1}^{n} a_i, \sum_{i=1}^{n} b_i / m \right\} \right) / n \qquad (7)$$

La ecuación (7) fue tomada de Lee y Kim (2004), donde P es una cota inferior para el tiempo de finalización de todos los trabajos. L y U son parámetros utilizados para controlar el ajuste de cada fecha límite en el problema. El manipular la creación de las fechas límite con parámetros en el rango [0, 1], nos permite crear fechas límite con mayor ajuste a los tiempos de finalización de cada tarea o con mayor holgura. En el trabajo de Lee y Kim (2004) se mencionan diez parámetros de ajuste (L,U): {(0.2, 0.4), (0.2, 0.6), (0.2, 0.8), (0.2, 1.0), (0.4, 0.6), (0.4, 0.8), (0.4, 1.0), (0.6, 0.8), (0.6, 1.0), (0.8,1.0)}, y para efectos del presente trabajo, únicamente se generaron fechas límite con (0.2, 0.4) porque es uno de los parámetros que las genera con mayor ajuste (P × 0.2, P × 0.4), además, resultados de dichos autores, indican que el número de problemas resueltos es independiente a los parámetros usados (Tabla III).

Tabla II: Siete fechas límite (FL) en unidades de tiempo para un conjunto de 10 tareas. Las primeras columnas contienen las fechas límite con mayor ajuste, las últimas columnas muestran lo contrario para la misma tarea.

	FL 1	**FL 2**	**FL 3**	**FL 4**	**FL 5**	**FL 6**	**FL 7**
Tarea 1	153	121	238	295	298	374	334
Tarea 2	227	342	153	300	257	233	532
Tarea 3	142	275	229	536	303	354	375
Tarea 4	182	329	353	498	318	267	466
Tarea 5	211	191	256	430	288	372	268
Tarea 6	130	130	329	416	336	432	345
Tarea 7	159	190	188	499	303	454	523
Tarea 8	156	202	200	468	313	356	470
Tarea 9	151	134	233	504	328	230	497
Tarea 10	173	165	410	459	319	353	263

Tabla III: Número de problemas resueltos en Lee y Kim (2004) con su Algoritmo BB4 de ramificación y acotamiento probado con diferentes niveles de exigencia de las fechas límite.

Parámetros (L, U) para generar fechas límite con diferente ajuste	***Número de problemas probados***	***Número de problemas resueltos***	***Promedio del límite inferior del tiempo de CPU (s)***
(0.2, 0.4)	28	24	23.4
(0.2, 0.6)	28	26	19.6
(0.2, 0.8)	28	25	23.1
(0.2, 1.0)	28	27	4.0
(0.4, 0.6)	28	24	59.3
(0.4, 0.8)	28	21	79.5
(0.4, 1.0)	28	25	8.1
(0.6, 0.8)	28	26	9.9
(0.6, 1.0)	28	24	35.7
(0.8, 1.0)	28	24	56.5

En la Tabla III se puede apreciar la columna (*L*, *U*) que contiene diferentes parámetros para generar las fechas límite. Por ejemplo, (0.2, 0.4) al multiplicarlo por *P* genera, con alta probabilidad fechas límite ajustadas, tal que no se puede terminar de procesar la tarea en su totalidad antes de lo indicado por dicha fecha o tan cerca de ésta que

justo a tiempo se pueda terminar de procesar por completo la tarea, la pareja (0.8, 1.0) tiene el efecto contrario. También se observa en la tabla que el ajuste de estas fechas generadas a partir de los datos proporcionados en la columna (*L*, *U*), no afecta seriamente al desempeño del Algoritmo 4 de ramificación y acotamiento (denotado como Algoritmo BB4 en Lee y Kim, 2004) en relación al número de problemas. Esto se debe a que para cada pareja (*L*, *U*) se probaron 28 casos y se resolvieron entre 21 y 26 de ellos sin mostrar algún patrón o dependencia de las fechas límite, lo mismo se observa para los límites inferiores de tiempo de CPU.

Los experimentos realizados se llevaron a cabo con distintos casos (ver Apéndice A), estos fueron generados a partir del programa de Taillard. Cada caso se generó con cierto número de trabajos como datos de entrada y cierto número de máquinas para la segunda etapa. Los tiempos de ejecución de cada tarea proporcionados por el programa de Taillard son generados al azar en base a una semilla y bajo la distribución uniforme discreta $DU(1, 100)$. Para efectos de ilustrar los archivos de salida con resultados arrojados por el sistema véase el Apéndice B.

IV.2 Experimentos

Los experimentos se ejecutaron en una PC de escritorio con sistema operativo Windows XP, procesador AMD Athlon XP 2500+ de 1.84 Ghz y 512 Mb en memoria principal. El AG y el programa de Taillard se implementaron en el lenguaje ANSI C.

La Tabla IV contiene los nombres de los algoritmos utilizados para los experimentos. Estos algoritmos combinan los distintos operadores de cruzamiento y mutación a excepción

del operador de cruzamientos SB2OX con el operador de mutación switch debido a que Ishibuchi y Shibata (2004) y Aceves (2003) coinciden al concluir que este último operador de mutación no tiene un buen desempeño como insert y swap, además, el operador SB2OX se implementó tiempo posterior a los otros operadores de cruzamiento.

Tabla IV: Combinación entre operadores de cruzamiento y mutación utilizados en los AGs.

	Operadores de cruzamiento					**Operadores de mutación**		
	OBX	PPX	OSX	Two_Point	SB2OX	insert	swap	switch
Algoritmo 1	X					X		
Algoritmo 2	X						X	
Algoritmo 3	X							X
Algoritmo 4		X				X		
Algoritmo 5		X					X	
Algoritmo 6		X						X
Algoritmo 7			X			X		
Algoritmo 8			X				X	
Algoritmo 9			X					X
Algoritmo 10				X		X		
Algoritmo 11				X			X	
Algoritmo 12				X				X
Algoritmo 13					X	X		
Algoritmo 14					X		X	

A continuación se presentan los resultados que surgen a partir de los experimentos realizados con los algoritmos. Primero se abordan los experimentos concernientes al retraso total (T) y posteriormente los experimentos que optimizan el tiempo máximo de finalización de todas las tareas (*Cmax*).

IV.2.1 Resultados del Retraso Total

Todos los AGs se configuraron con los parámetros que se presentan en la Tabla V, con excepción del tamaño de población (T_{pobl}) que se modifica de acuerdo a los resultados que se observaban en experimentos anteriores al que se encuentra en estudio.

En las series de experimentos para T y para *Cmax* que se presentan a continuación, los tiempos de procesamiento de cada tarea y las fechas de finalización de cada una de éstas no son las mismas entre los experimentos realizados con un algoritmo y otro, aunque ambos casos presenten el mismo número de tareas o sean casos consecutivos, a menos de que se especifique lo contrario para alguna serie. Esto significa que entre el caso 25 × 5 (25 tareas y 5 máquinas) y 25 × 10 (25 tareas y 10 máquinas) los tiempos de procesamiento y las fechas límite son diferentes. Lo anterior se debe a que en cada ocasión que se genera un caso, se generan distintos tiempos de procesamiento para las tareas.

Tabla V: Parámetros utilizados por los AGs en los experimentos.

Parámetro	Descripción	Valor
Pm	Probabilidad de mutación	0.01
Pc	Probabilidad de cruzamiento	0.9
C_{paro}	Repeticiones del mejor individuo	100
$N_{\exp}$	Número de ejecuciones de cada algoritmo	30

IV.2.1.1 Retraso Total: Experimento 1

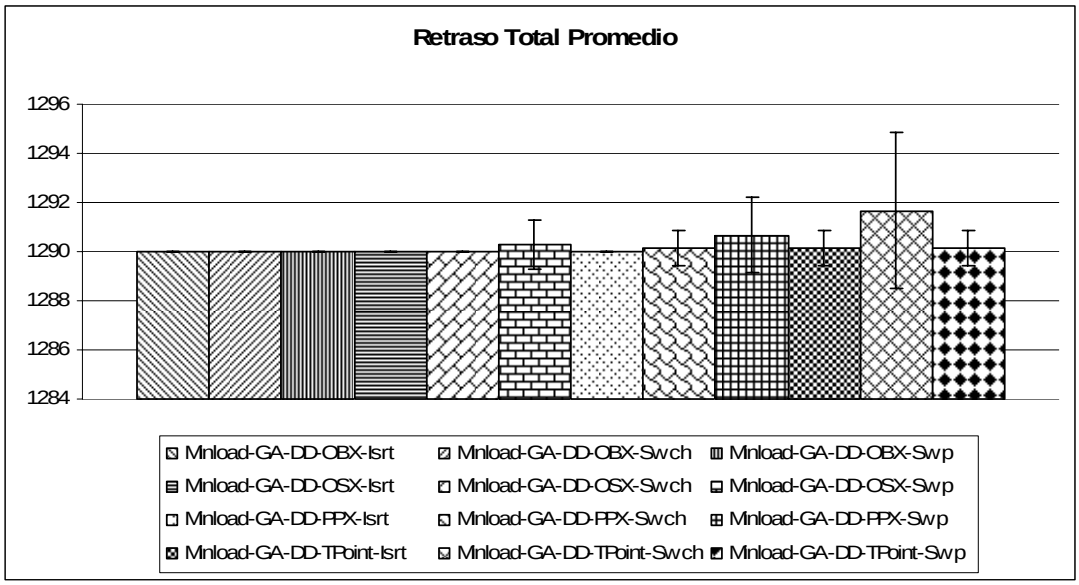
En esta primera serie de experimentos dedicada al retraso total, se ha utilizado diversidad en los tamaños de población y en los tamaños de los casos (Tabla VI). Dicha diversidad se introdujo con el propósito de conocer la calidad de los resultados y la robustez de cada

algoritmo. La intención es observar y descartar a los algoritmos que presentan un bajo desempeño para optimizar T después de $N_{\exp}$ (Figura 18)**,** es decir, $\overline{T}$ (promedio del retraso total). Un esquema mejor para esta fase de experimento puede ser fijar el caso y analizar los tamaños de población o dejar fijo el tamaño de población y variar los casos del problema.

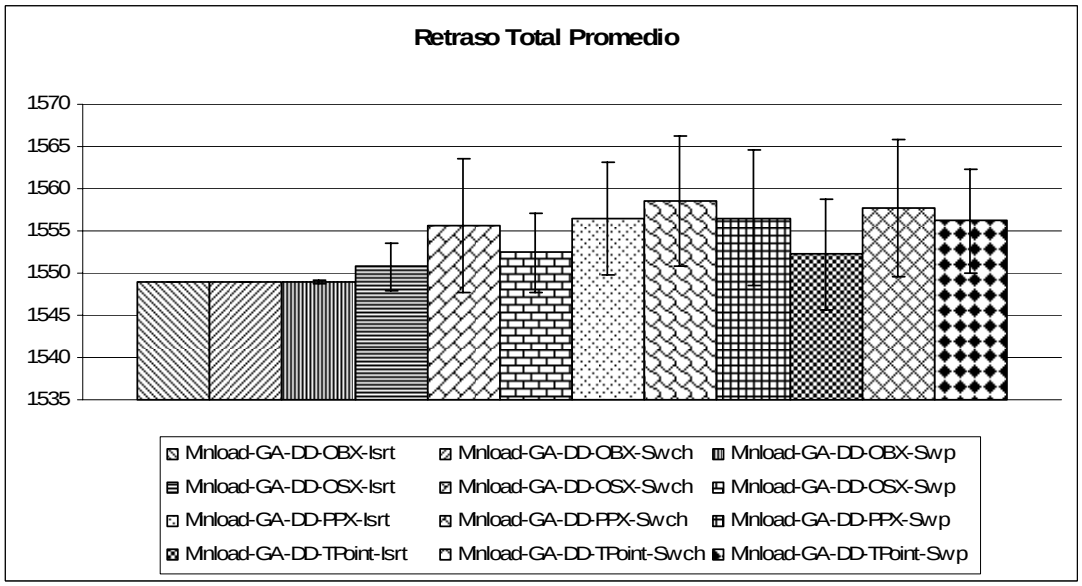
Tabla VI: Tamaños de población y casos utilizados con los AGs.

Figura	**Algoritmos ejecutados**	**Tamaño de la población** T_{pobl}	**Tamaño del caso (*n* × *m*)**
18-(a)	{1,...,12}	200	10 × 2
18-(b)	{1,...,12}	250	13 × 4
18-(c)	{1,...,12}	300	16 × 5
18-(d)	{1,...,12}	300	17 × 6
18-(e)	{1,...,12}	350	20 × 8
18-(f)	{1,...,12}	400	23 × 9

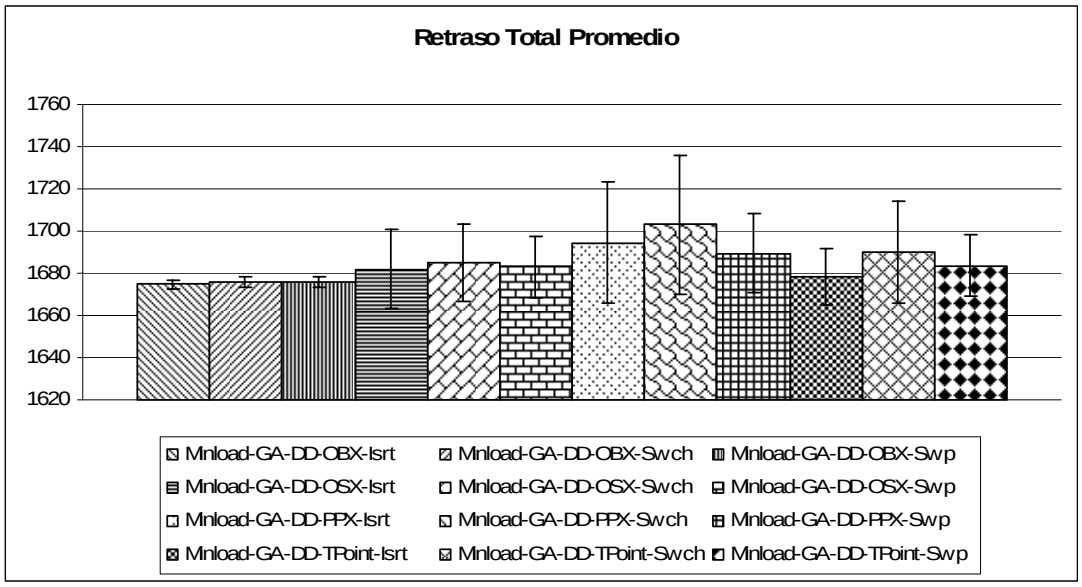
Los nombres de los algoritmos que aparecen al pie de las Figura 18-(a) hasta la Figura 18-(f) están compuestos por cinco partes: La primera parte corresponde a la forma en que se asignan recursos a los procesos en la segunda etapa de máquinas de acuerdo a la estrategia *list-scheduling* (en las figuras se indica como MinLoad). Posteriormente se hace alusión a la heurística utilizada para resolver el problema (AG). Después se indica el criterio a optimizar que en nuestro caso es el retraso total (DD) o el tiempo de finalización de todas las tareas (*Cmax*). En seguida se nombra el operador de cruzamiento utilizado. Finalmente, se indica el operador de mutación. Por ejemplo: MinLoad-GA-DD-OBX-insert indica que se utilizó la regla de despacho *list-scheduling* en un AG, aplicado a la optimización de las fechas límite DD, usando el operador de cruzamiento OBX y el operador de mutación insert.


Retraso Total Promedio
1296
1294
1292
1290
1288
1286
1284
Mnload-GA-DD-OBX-Isrt
Mnload-GA-DD-OBX-Swch
Mnload-GA-DD-OBX-Swp
Mnload-GA-DD-OSX-Isrt
Mnload-GA-DD-OSX-Swch
Mnload-GA-DD-OSX-Swp
Mnload-GA-DD-PPX-Isrt
Mnload-GA-DD-PPX-Swch
Mnload-GA-DD-PPX-Swp
Mnload-GA-DD-TPoint-Isrt
Mnload-GA-DD-TPoint-Swch
Mnload-GA-DD-TPoint-Swp

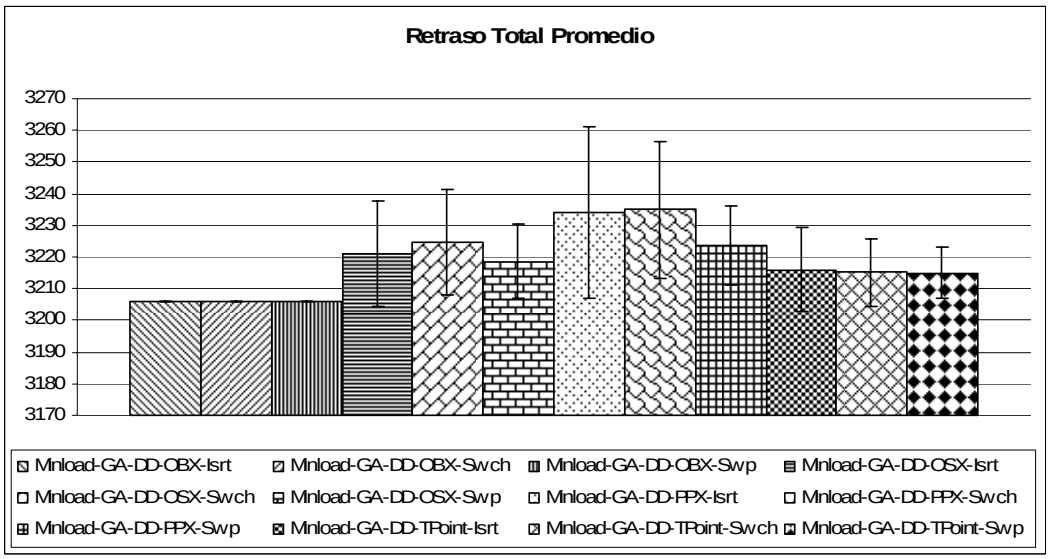
(a)


Retraso Total Promedio
1570
1565
1560
1555
1550
1545
1540
1535
Mnload-GA-DD-OBX-Isrt
Mnload-GA-DD-OBX-Swch
Mnload-GA-DD-OBX-Swp
Mnload-GA-DD-OSX-Isrt
Mnload-GA-DD-OSX-Swch
Mnload-GA-DD-OSX-Swp
Mnload-GA-DD-PPX-Isrt
Mnload-GA-DD-PPX-Swch
Mnload-GA-DD-PPX-Swp
Mnload-GA-DD-TPoint-Isrt
Mnload-GA-DD-TPoint-Swch
Mnload-GA-DD-TPoint-Swp

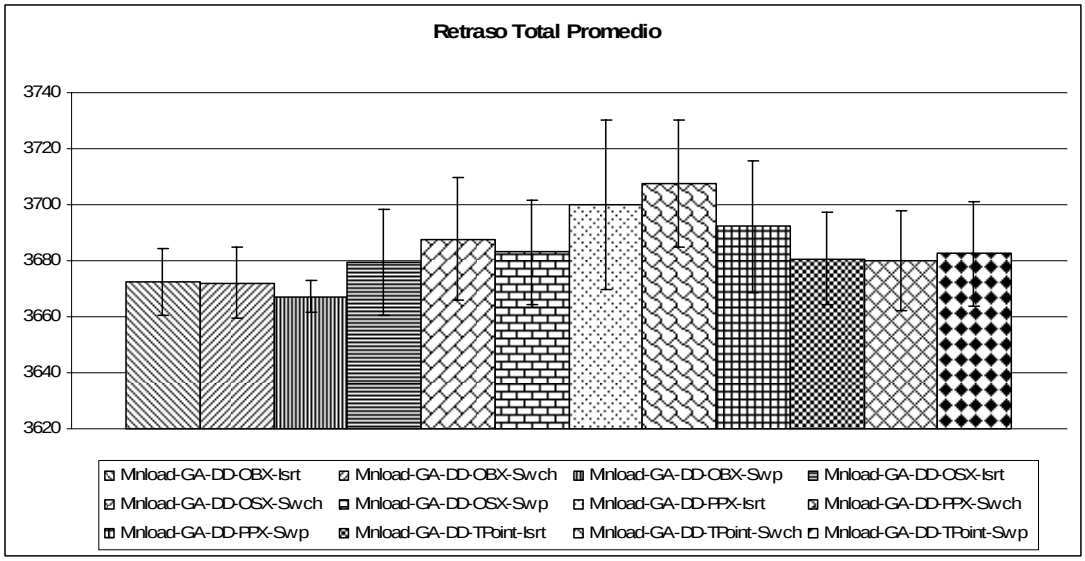
(b)


Retraso Total Promedio
1760
1740
1720
1700
1680
1660
1640
1620
Mnload-GA-DD-OBX-Isrt
Mnload-GA-DD-OBX-Swch
Mnload-GA-DD-OBX-Swp
Mnload-GA-DD-OSX-Isrt
Mnload-GA-DD-OSX-Swch
Mnload-GA-DD-OSX-Swp
Mnload-GA-DD-PPX-Isrt
Mnload-GA-DD-PPX-Swch
Mnload-GA-DD-PPX-Swp
Mnload-GA-DD-TPoint-Isrt
Mnload-GA-DD-TPoint-Swch
Mnload-GA-DD-TPoint-Swp

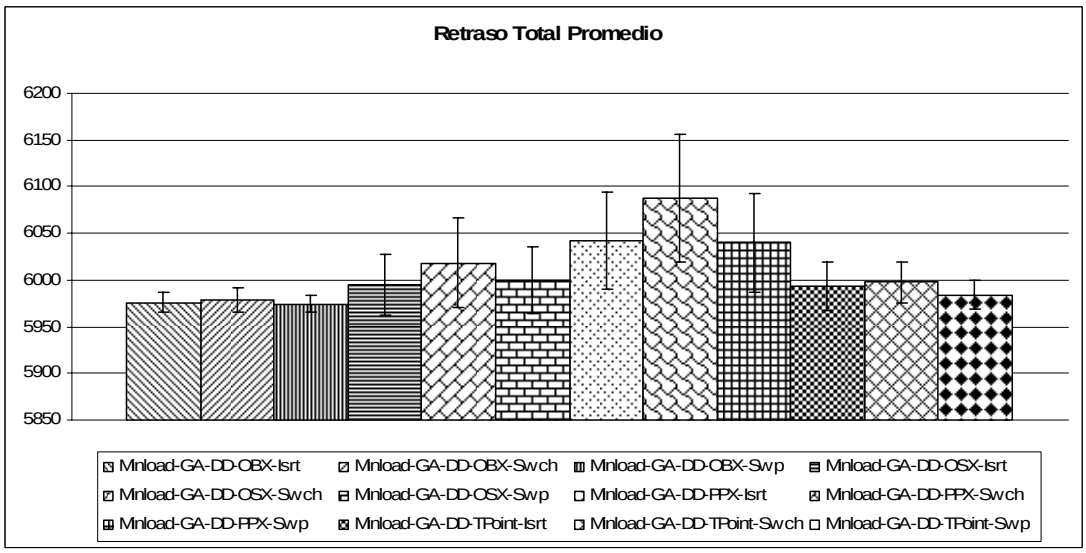
(c)


Retraso Total Promedio
3270
3260
3250
3240
3230
3220
3210
3200
3190
3180
3170
Mnload-GA-DD-OBX-Isrt
Mnload-GA-DD-OBX-Swch
Mnload-GA-DD-OBX-Swp
Mnload-GA-DD-OSX-Isrt
Mnload-GA-DD-OSX-Swch
Mnload-GA-DD-OSX-Swp
Mnload-GA-DD-PPX-Isrt
Mnload-GA-DD-PPX-Swch
Mnload-GA-DD-PPX-Swp
Mnload-GA-DD-TPoint-Isrt
Mnload-GA-DD-TPoint-Swch
Mnload-GA-DD-TPoint-Swp

(d)



(e)



(f)

Figura 18: Retraso total promedio de cada AG ({1,...,12}) con N_{exp} = 30. El segmento de línea en cada barra muestra su correspondiente desviación estándar. Entre más pequeños sean el promedio y su desviación estándar el AG correspondiente es más robusto. En a) la mayor desviación estándar la obtuvo TPoint-Switch y OBX obtuvo mayor estabilidad en el promedio de *T*. Nuevamente OBX vuelve a presentar mayor robustez en b). Cuando se realiza el experimento c), OBX permanece por debajo del promedio del resto de los algoritmos. El experimento d) muestra que OBX logra mejor desempeño que el resto de los algoritmos, PPX muestra lo contrario. En e) y f) OBX y PPX se comportan de manera similar a d).

En la Figura 18 se observa que los algoritmos basados en combinaciones del operador OBX con cualquiera de los tres operadores de mutación insert, swap y switch (primeras tres barras) muestran un desempeño notable para $\overline{T}$ en comparación con el resto de los algoritmos ya que muestran estabilidad al minimizar $\overline{T}$ y menor SD_T. Los algoritmos que mantienen un desempeño intermedio son los que contienen a los operadores de cruzamiento OSX y Two-Point con los tres operadores de mutación, pero en mayor frecuencia con el operador switch. Finalmente, el algoritmo que presenta peor desempeño para minimizar $\overline{T}$**,** es el que incluye a PPX con los tres operadores de mutación. Lo mismo se observa para las desviaciones estándar (SD_T , *Standard Deviation*), correspondientemente.

A través de los experimentos anteriores se ha examinado el desempeño de los algoritmos que incluyen a cuatro de los cinco operadores de cruzamiento en combinación con cada uno de los tres operadores de mutación. Se puede concluir que los algoritmos que combinan al operador OBX con insert y swap muestran resultados con mayor calidad para $\overline{T}$ y menor SD_T. A su vez coincide con la descripción de Ishibuchi y Shibata (2004). Este mejor desempeño se logra con cualquier T_{pobl} y con cualquier caso.

IV.2.1.2 Retraso Total: Experimento 2

En base a los resultados del Experimento 1, optamos por realizar un segundo experimento para $\overline{T}$ únicamente con los Algoritmos 1 y 2, con el fin de saber cuál operador de mutación (insert o swap) obtiene ventaja sobre el otro bajo las mismas condiciones y mismo operador de cruzamiento (i. e, OBX-insert y OBX-swap).

Para esta serie de experimentos utilizamos únicamente 14 máquinas en la segunda etapa y un T_{pobl} de 200 individuos. Con estos parámetros constantes se desea observar el efecto en $\overline{T}$ provocado únicamente por el incremento de trabajos. También deseamos ver si se muestran cambios en el tiempo de resolución del problema. Los resultados obtenidos se muestran en la Tabla VII.

Tabla VII: Resultados de los experimentos realizados con los algoritmos 1 y 2 para $\overline{T}$.

Caso *(n × m)*	***Algoritmo***	$\overline{T}$	***Mejor resultado de cada AG.***	SD_T	$\bar{t}(seg)$	$SD_t(seg)$
24 × 14	1	5,233	5,192	42	3.32	0.24
	2	5,235	5,192	43	3.28	0.25
25 × 14	1	6,261	6,257	17	3.62	0.12
	2	6,258	6,257	2	3.69	0.51
30 × 14	1	6,344	6,308	39	5.17	0.37
	2	6,335	6,308	28	5.42	0.92
40 × 14	1	13,156	13,137	16	17	4
	2	13,163	13,141	12	12	2
50 × 14	1	15,110	15,094	29	45	10
	2	15,106	15,097	9	38	5
80 × 14	1	50,320	50,069	119	98	20
	2	50,297	50,078	144	91	12
100 × 14	1	72,409	72,312	118	434	106
	2	72,438	72,316	119	340	19
120 × 14	1	86,860	86,632	163	380	106
	2	86,911	86,548	216	308	25
140 × 14	1	156,358	155,772	384	557	64
	2	156,414	155,746	475	545	48
160 × 14	1	195,903	195,151	386	1034	172
	2	195,805	194,931	336	922	103
180 × 14	1	223,693	222,873	451	1629	323
	2	223,843	223,140	416	1482	176
200 × 14	1	283,148	282,162	676	1971	265
	2	282,872	282,027	449	1858	94

Al observar la Tabla VII se concluye que no existe un patrón claro para definir el mejor algoritmo. Los resultados muestran variación tanto para $\overline{T}$ como para SD_T. En ocasiones el Algoritmo 2 que combina al operador de cruzamiento OBX con el operador de mutación swap muestra resultados con mayor calidad para $\overline{T}$ como en el caso 25 × 14, pero no siempre es así (caso 24 × 14). El mismo comportamiento se observa para la SD_T y la obtención del mejor resultado logrado por cada algoritmo, por ejemplo, el Algoritmo 1 obtiene mejor resultado con el caso 40 × 14, sin embargo, el Algoritmo 2 obtiene mejor resultado en el caso 140 × 4. El tiempo se incrementa conforme se aumenta el número de trabajos y se dispara en el caso 100 × 14 por encima de los 100 segundos. El Algoritmo 2 que usa el operador swap resuelve el problema en menos tiempo que el Algoritmo 1. Lo mismo se cumple para la desviación estándar.

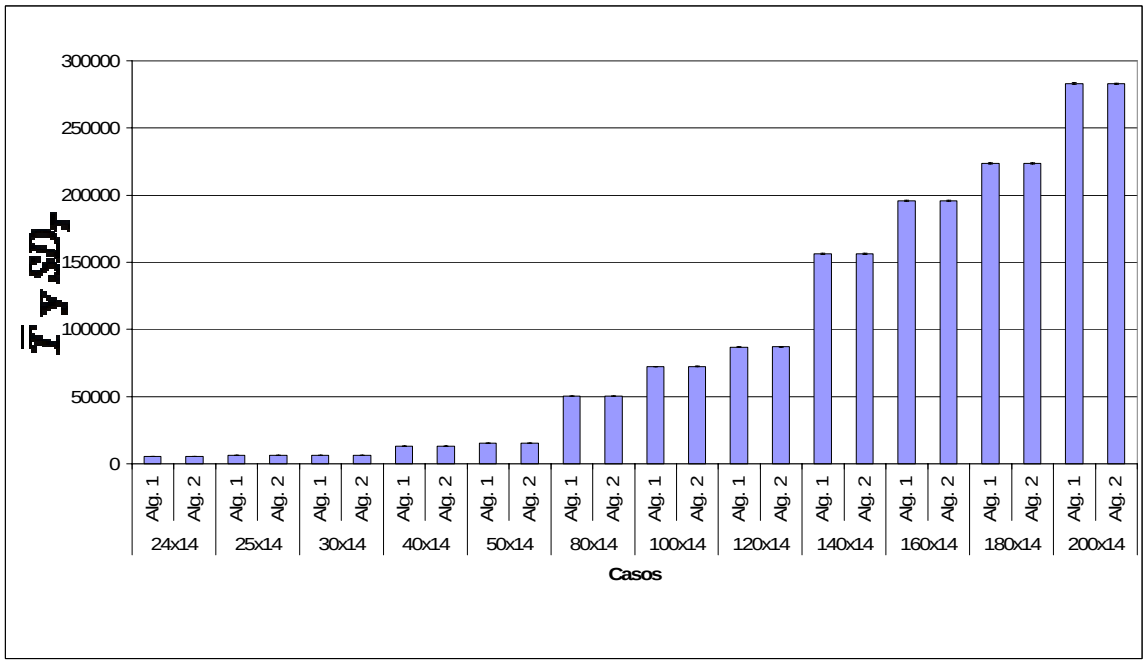


Figura 19: Comportamiento de $\overline{T}$ **y** SD_T con N_{exp} = 30.

Se observa en la Figura 19 que la SD_T de cada experimento es despreciable con relación al valor logrado por los AGs para $\overline{T}$ en cada caso.

Se puede concluir que no existe una clara diferencia en el desempeño de los Algoritmos 1 y 2 para optimizar T. Debido al valor tan pequeño de SD_T en el intervalo de (0, 700), en comparación con la magnitud de $\overline{T}$ entre (0, 300 000) (ver Tabla VII), en la Figura 19 sólo se aprecia la barra que representa a $\overline{T}$ y a SD_T como un punto en la parte superior de las barras para algunos casos. Un análisis estadístico de estos resultados nos dirían, muy probablemente, que las diferencias observadas entre ambos algoritmos no son estadísticamente significativas.

La Tabla VIII presenta los incrementos del retraso total en promedio y los tiempos de ejecución en promedio que existen entre un caso que tienen un determinado número de tareas y otro conteniendo dos veces este número. La intención es observar el efecto que surge al variar el número de tareas únicamente.

Se observa que al duplicar el número de tareas, el promedio del retraso se duplica, quintuplica y cuadruplica (últimos dos casos), correspondientemente. Estos resultados nos indican que no hay una clara relación en el incremento entre el número de tareas y $\overline{T}$. Lo mismo se aplica para $\bar{t}(seg)$.

Tabla VIII: Porcentaje del incremento (drástico) para $\overline{T}$ y para $\bar{t}(seg)$ al duplicar el número de tareas.

Casos comparados ($n \times m$) con ($n \times m$)	**Algoritmo**	**(%) Incremento del retraso total**	**(%) Incremento del tiempo** $\bar{t}(seg)$
25 × 14 con 50 × 14	1	241	1233
	2	241	1054
50 × 14 con 100 × 14	1	479	972
	2	480	892
80 × 14 con 160 × 14	1	389	1060
	2	389	1018
100 × 14 con 200 × 14	1	391	454
	2	391	546

IV.2.1.3 Retraso Total: Experimento 3

A continuación se muestran los resultados de $\overline{T}$ usando los algoritmos 1 y 2 (Tabla IX). La T_{pobl} se mantiene en 200 individuos. Los casos se generaron con cuatro conjuntos de tareas (25, 50, 100 y 200) y cada caso se probó con tres conjuntos de máquinas (5, 10 y 20) en la segunda etapa del sistema. En total son 24 experimentos. Los tamaños de los casos están basados en el trabajo que presentan Ruíz *et al*., (2005).

Los resultados en la Tabla IX no muestran una secuencia o patrón, es decir, hay casos en que el Algoritmo 1 muestra mejor SD_T (caso 25 × 5) y hay casos que lo hace el Algoritmo 2 (caso 200 × 5). En base a esto, sigue sin definirse un algoritmo con ventaja sobre el otro. Se observa lo mismo en la columna del mejor resultado (casos 25 × 20 y 50 × 20) y en los resultados de la SD_T.

Antes de realizar los experimentos se consideraba que conforme se duplicara el número de máquinas en los casos, $\overline{T}$ disminuiría considerablemente, sin embargo, no existe una clara proporcionalidad entre el número de máquinas y el resultado que se obtiene

para $\overline{T}$. No hay que descartar que sí disminuye $\overline{T}$ (excepto con los casos 100 × 10 y 200 × 20). Conforme se incrementa el número de máquinas, los algoritmos consumen más tiempo para resolver el problema. Se observa que $\overline{T}_{seg}$ se mantiene semejante entre los casos n × (5 y 10) y que con los casos n × 20 tiende a incrementarse. Conforme se duplica n y m, la $SD_t(seg)$ tiende a crecer y a tener el mismo patrón que $\bar{t}(seg)$.

Tabla IX: Resultados de los experimentos realizados para los algoritmos 1 y 2 con 12 casos.

Caso *(n × m)*		***Algoritmo***	$\overline{T}$	***Mejor resultado de cada AG.***	SD_T	$\bar{t}(seg)$	$SD_t(seg)$
25 ×	5	1	5,339	5,324	14.30	1.51	0.19
	5	2	5,346	5,324	14.29	1.46	0.10
	10	1	2,453	2,447	4.95	1.21	0.17
	10	2	2,454	2,447	5.40	1.22	0.15
	20	1	2,274	2,250	10.70	2.21	0.29
	20	2	2,277	2,271	6.03	2.15	0.21
50 ×	5	1	17,816	17,728	104.85	10.22	1.04
	5	2	17,784	17,728	87.85	10.26	0.85
	10	1	17,035	16,957	54.48	10.31	0.85
	10	2	17,043	16,957	53.60	10.44	0.77
	20	1	15,083	15,061	43.53	12.69	1.55
	20	2	15,096	15,059	50.93	11.84	1.12
100 ×	5	1	82,653	82,387	175	126	8
	5	2	82,656	82,420	143	124	8
	10	1	66,462	66,328	138	108	10
	10	2	66,483	66,335	110	104	7
	20	1	66,131	65,713	260	126	10
	20	2	66,134	65,640	259	130	11
200 ×	5	1	288,132	286,528	860	1,515	73
	5	2	287,932	286,812	733	1,603	137
	10	1	272,931	272,036	520	1,495	89
	10	2	272,955	271,421	678	1,521	144
	20	1	280,833	279,703	767	1,586	127
	20	2	280,569	279,282	793	1,571	123

Algunas razones por las que puede no existir una relación de proporcionalidad entre $\overline{T}$ y el incremento del número de máquinas son las siguientes:

a) Se puede deber a que tanto los tiempos de procesamiento y las fechas de finalización para cada tarea no son las mismas entre un caso y otro. Esto quiere decir que entre el caso 25 × 5 y 25 × 10 no tienen las mismas fechas límite ni los mismos tiempos de procesamiento, por lo tanto, si no hay relación en estos datos, tampoco existirá dicha relación en los resultados de *T*.

b) Otra posibilidad es que para algunos casos (por ejemplo 200 × 5 y 200 × 10) las fechas límite presenten gran diferencia entre sí, siendo menor dicha diferencia entre el primer caso (e.g. 200 × 5) con un tercero (e.g. 200 × 20) y es por eso que se hace notorio el desempeño para $\overline{T}$ con el último caso.

c) La otra posibilidad es que aunque se incremente el número de máquinas no se podrá disminuir más el retraso debido a que las fechas de finalización se encuentran tan ajustadas que las tareas no pueden ser terminadas en su totalidad antes de que las fechas límite expiren. Un ejemplo se presenta en la Figura 20 (izquierda) en donde se muestra el caso de la tarea (j_2) con las fecha de finalización antes de poder procesarla por completo, aunque se incrementa el número de máquinas y/o se realice otra combinación de tareas (Figura 20 derecha) se sigue presentando retraso para la tarea (j_2) y (j_1).

En la Sección IV.2.1.5 del presente capítulo se realizarán experimentos en donde los tiempos de procesamiento y las fechas límite se mantienen iguales para los casos que tienen el mismo número de tareas. Solamente se duplicará el número de máquinas. Lo anterior es para confirmar alguna(s) de las hipótesis descritas anteriormente.

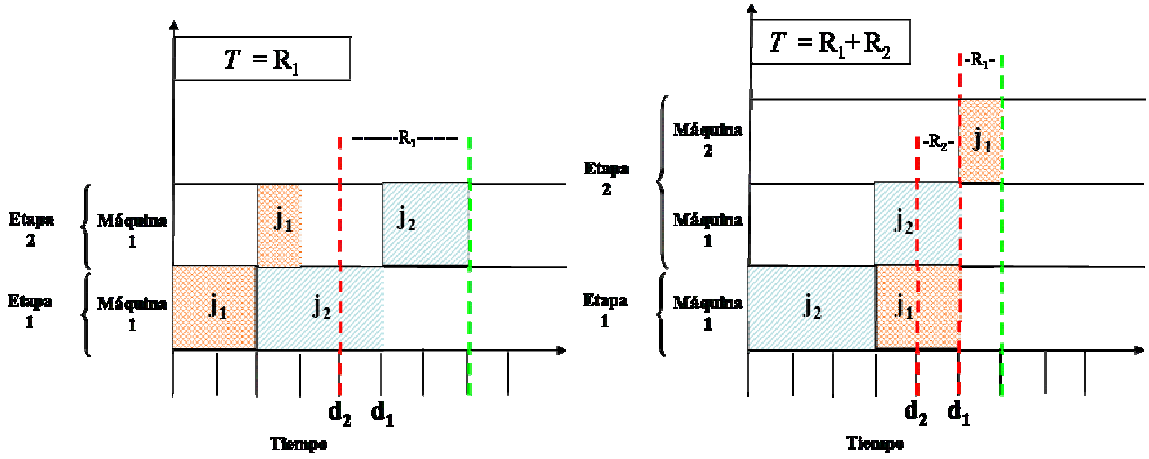


Figura 20: Ejemplo de retraso total con dos tareas. A la izquierda se presenta un retraso de 3 unidades (R_1), al incrementar el número de máquinas en la segunda etapa se logra disminuir 2 unidades el retraso ($R_1 + R_2$) (derecha). Es la consecuencia de tener fechas ajustadas para las tareas.

La Figura 21 muestra las desviaciones estándar para cada caso. Se observa que se obtuvieron desviaciones estándar pequeñas (despreciables considerando el rango de valores que se maneja para $\overline{T}$) (ver Tabla IX) con relación al retraso promedio e incrementan su tamaño conforme se duplica n y m, la idea de ilustrar $\overline{T}$ y SD_T, es precisamente, hacer notar que la SD_T es prácticamente nula, lo cual indica que los AGs muestran buen desempeño para optimizar *T*.

Después de observar los resultados de los experimentos se puede decir que conforme se va incrementando el número de tareas y máquinas, le cuesta más trabajo al algoritmo disminuir el retraso total. Esto se aprecia en la Figura 22 en donde se graficó el número de tareas contra el promedio del retraso total por tarea (T / n).

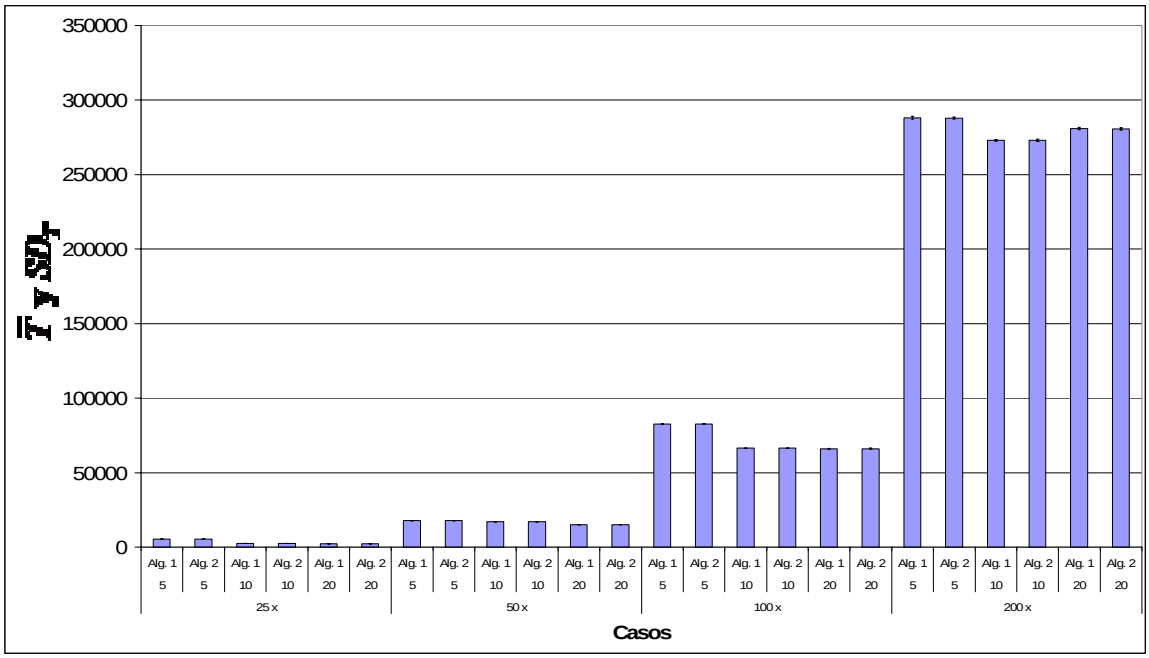


Figura 21: $\overline{T}$ y $SD_{\overline{T}}$ obtenidos con cada algoritmo después de ejecutarlos 30 veces.

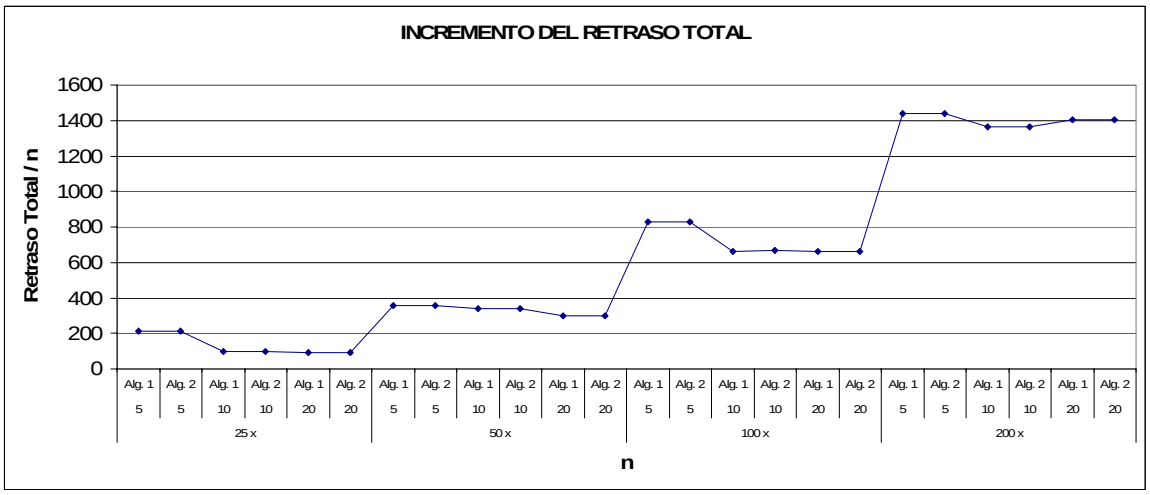


Figura 22: Retraso total por tarea (T / n) en función del número de tareas para cada algoritmo.

El incremento del retraso se hace más notorio en los casos (25, 50, 100 y 200) × 5, es decir, el algoritmo no logra disminuir el retraso con la misma calidad teniendo 5 que teniendo 10 ó 20 máquinas en la segunda etapa.

IV.2.1.4 Retraso Total: Experimento 4

En esta sección se realiza un estudio del retraso total usando los Algoritmos 13 y 14, además de los Algoritmos 1 y 2. En la Tabla X se presentan los resultados de los 24 experimentos. Se han incluido los Algoritmos 13 y 14 que utilizan al operador de cruzamiento SB2OX y que a su vez este utiliza el operador de cruzamiento de un punto (OPX), en caso de no encontrar segmentos de cadenas iguales en los cromosomas a cruzar. Dicho operador (SB2OX) resulta ser el que presenta mejores resultados de todos los probados en el trabajo de Ruíz *et al*., (2005). En dicho trabajo se minimiza el *Cmax*. Se consideran los tiempos de preparación (*setup time*), máquinas diferentes y en cada etapa distinto número de ellas. Se experimenta con $T_{pobl} = 50$, parámetro basado también en el trabajo de dichos autores. El tamaño de este parámetro se ha incluido con la intención de conocer la calidad de los resultados en comparación con $T_{pobl} = 200$ en los algoritmos 1 y 2. Es importante hacer notar que los casos (tiempos de procesamiento) y las fechas límite (due date) son exactamente las mismos que se utilizaron en los experimentos de la sección anterior.

Tabla X: Resultados de experimentos realizados con los algoritmos 1, 2, 13 y 14.

Caso *(n × m)*		***Algoritmo***	$\overline{T}$	***Mejor resultado de cada AG.***	SD_T	$\bar{t}(seg)$	$SD_t(seg)$
25 ×	5	1	5,375	5,335	24.25	0.58	0.17
	5	2	5,369	5,338	23.61	0.59	0.17
	5	13	5,480	5,351	90.78	0.69	0.35
	5	14	5,464	5,339	82.08	0.75	0.35
	10	1	2,459	2,447	8.92	0.60	0.18
	10	2	2,464	2,450	11.72	0.51	0.06
	10	13	2,580	2,456	98.51	0.57	0.19
	10	14	2,541	2,461	65.01	0.58	0.21
	20	1	2,293	2,250	32.03	0.75	0.21
	20	2	2,291	2,254	26.33	0.74	0.22
	20	13	2,366	2,279	74.81	0.85	0.28
	20	14	2,363	2,279	51.49	0.85	0.33
50 ×	5	1	18,087	17,750	220.86	3.59	1.58
	5	2	18,104	17,795	173.26	3.50	1.23
	5	13	18,830	18,428	347.49	3.07	1.08
	5	14	18,835	18,226	464.76	2.56	1.06
	10	1	17,185	17,026	102.45	3.41	0.89
	10	2	17,200	17,034	122.63	3.32	1.06
	10	13	18,000	17,355	449.38	3.41	1.64
	10	14	17,856	17,212	351.19	3.09	1.54
	20	1	15,356	15,230	104.67	4.15	1.47
	20	2	15,330	15,141	98.13	3.40	0.72
	20	13	16,301	15,520	465.00	3.63	1.65
	20	14	16,151	15,510	565.37	3.56	1.58
100 ×	5	1	84,277	83,383	441	33	8
	5	2	84,369	83,288	596	32	9
	5	13	89,856	85,768	3,030	16	6
	5	14	88,064	84,246	2,173	14	5
	10	1	67,721	67,168	463	35	11
	10	2	67,751	66,615	519	28	6
	10	13	72,752	69,339	3,260	18	8
	10	14	71,596	68,726	2,456	15	4
	20	1	67,914	66,750	708	35	13
	20	2	67,632	66,570	649	33	8
	20	13	73,000	69,653	2,282	20	7
	20	14	73,302	68,482	2,947	14	6

Caso *(n × m)*		***Algoritmo***	$\overline{T}$	***Mejor resultado de cada AG.***	SD_T	$\bar{t}(seg)$	$SD_t(seg)$
200 ×	5	1	301,963	296,579	3,105	394	130
	5	2	298,939	291,958	2,853	466	137
	5	13	327,008	306,692	11,474	143	36
	5	14	329,049	306,671	19,303	70	14
	10	1	284,217	278,778	2,920	453	177
	10	2	284,757	278,081	3,678	428	152
	10	13	311,281	294,147	13,792	145	34
	10	14	307,591	283,439	15,505	81	21
	20	1	293,167	288,218	2,732	525	182
	20	2	291,672	284,955	3,431	497	208
	20	13	319,648	302,535	9,745	138	32
	20	14	317,372	301,359	15,053	79	18

De la Tabla X se concluye que los Algoritmos 13 y 14 tienen menor desempeño que los Algoritmos 1 y 2 para minimizar $\overline{T}$. Además, conforme se duplica el número de tareas y máquinas la diferencia se acentúa considerablemente con respecto a $\overline{T}$. Lo mismo se observa en la columna del mejor resultado para SD_T. Aunque los Algoritmos 13 y 14 no logran obtener resultados con la misma calidad que los Algoritmos 1 y 2 para $\overline{T}$, si consumen menos tiempo en la resolución del problema y la correspondiente $SD_t(seg)$ es mas pequeña, este hecho se acentúa conforme se duplica n y m.

A pesar de que los Algoritmos 1 y 2 obtienen los mejores resultados con respecto al retraso, se sigue manteniendo la irregularidad para definir a un claro ganador entre estos dos algoritmos. A partir del caso 50 × 20 hasta el caso 200 × 20 el Algoritmo 2 presenta el mejor resultado. Esto indica que el operador de mutación swap tiende a obtener mejores resultados conforme se incrementan n y m, sin embargo, su SD_T muestra ser mayor que la del Algoritmo 1 en algunos casos entre 50 × 20 y 200 × 20. Por estas razones no se puede

asegurar que un algoritmo presente mayor eficiencia con respecto al otro para optimizar $\overline{T}$. Para $\bar{t}(seg)$ se muestra irregularidad ya que en algunos experimentos el Algoritmo 1 consume menos tiempo (50 × 10) que el Algoritmo 2 y viceversa (50 × 20). Lo mismo es válido se aplica para $SD_t(seg)$.

La diferencia de la desviación estándar del retraso entre los Algoritmos 1 y 2 con respecto a los Algoritmos 13 y 14 se aprecia mejor en los casos con 200 tareas (Figura 23). Primeramente, se observa que la SD_T para los Algoritmos 13 y 14 se va incrementando conforme se incrementa el número de tareas en los casos, además, se incrementa en forma considerable en comparación a la SD_T obtenida con los Algoritmos 1 y 2.

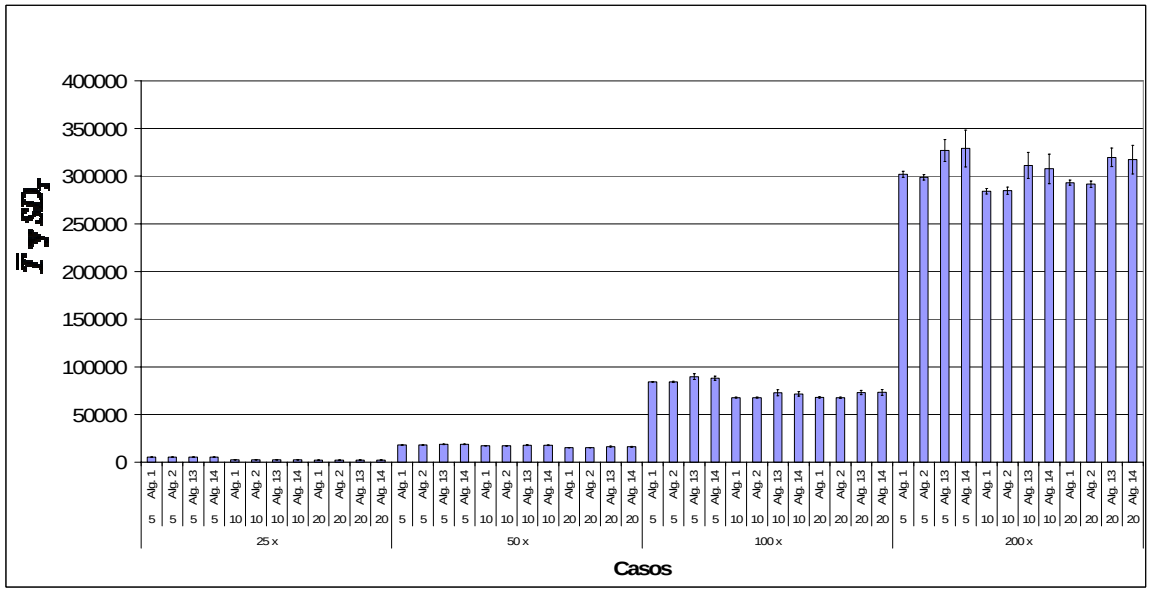


Figura 23: $\overline{T}$ y $SD_{\overline{T}}$ de los algoritmos 1, 2, 13 y 14 con 30 corridas y diferentes casos.

Es importante hacer notar que los experimentos con los Algoritmos 1 y 2 y una población de tamaño 50 han reducido la calidad de los resultados comparándolos con los experimentos realizados con una población de tamaño 200 de la sección IV.2.1.3.

IV.2.1.5 Retraso Total: Experimento 5

Para la presente serie de experimentos los tiempos de procesamiento entre casos que contienen el mismo número de tareas no varían, solamente se incrementa el número de máquinas en la segunda etapa, es decir, el caso 25 × 5 contiene los mismos tiempos de procesamiento que el caso 25 × 10. Los experimentos se realizaron con los Algoritmos 1, 2 13 y 14. Estos experimentos se llevaron a cabo con la intención de corroborar las probables razones o hipótesis expuestas en la sección IV.2.1.3 dedicada al retraso. T_{plobl} se ha establecido en 100 basándose en Ishibuchi y Shibata (2004) y Aceves (2003).

Tabla XI: Resultados del experimento 5 en donde los casos cuentan con el mismo número de tareas, los mismos tiempos de procesamiento y las mismas fechas de finalización (due dates).

Caso *(n × m)*		***Algoritmo***	$\overline{T}$	***Mejor resultado de cada AG.***	SD_T	$\bar{t}(seg)$	$SD_t(seg)$
25 ×	5	1	5,350	5,324	19.06	1.02	0.12
	5	2	5,349	5,324	13.42	1.01	0.13
	5	13	5,383	5,324	37.29	1.18	0.44
	5	14	5,383	5,337	26.49	1.20	0.43
	10	1	5,348	5,324	15.17	1.11	0.14
	10	2	5,347	5,324	20.33	1.09	0.11
	10	13	5,402	5,329	53.48	1.20	0.39
	10	14	5,373	5,335	29.96	1.38	0.40
	20	1	5,348	5,324	18.21	1.25	0.11
	20	2	5,353	5,324	16.71	1.30	0.25

Caso *(n × m)*		***Algoritmo***	$\overline{T}$	***Mejor resultado de cada AG.***	SD_T	$\bar{t}(seg)$	$SD_t(seg)$
	20	13	5,401	5,324	53.85	1.52	0.55
	20	14	5,385	5,326	39.61	1.78	0.63
50 ×	5	1	17,869	17,728	101.00	5.99	1.30
	5	2	17,895	17,728	100.65	6.00	1.35
	5	13	18,372	17,854	373.95	5.80	1.42
	5	14	18,260	17,815	239.55	5.48	1.89
	10	1	17,885	17,728	92.14	6.09	1.24
	10	2	17,891	17,728	140.14	5.99	0.90
	10	13	18,370	17,890	274.46	6.66	2.20
	10	14	18,222	17,810	284.87	6.43	2.51
	20	1	17,928	17,728	141.38	6.32	1.48
	20	2	17,898	17,737	99.51	6.89	1.86
	20	13	18,300	17,972	211.29	7.35	1.98
	20	14	18,237	17,835	275.44	7.61	2.83
100 ×	5	1	82,990	82,616	210	61	11
	5	2	83,000	82,661	205	53	8
	5	13	85,705	83,792	1,172	43	10
	5	14	85,178	83,375	1,544	33	9
	10	1	83,001	82,516	226	62	15
	10	2	83,003	82,577	273	52	7
	10	13	85,832	83,179	1,481	46	18
	10	14	85,052	83,492	1,281	34	8
	20	1	82,997	82,487	271	64	14
	20	2	83,101	82,659	249	55	9
	20	13	86,008	84,010	1,394	44	13
	20	14	84,622	83,054	1,179	45	13
200 ×	5	1	291,423	289,232	1,644	585	77
	5	2	290,740	288,355	1,239	617	107
	5	13	309,338	298,623	7,669	426	116
	5	14	301,681	294,789	5,674	283	49
	10	1	291,151	289,207	1,343	991	145
	10	2	290,594	287,946	1,122	968	156
	10	13	307,919	300,577	5,365	665	154
	10	14	303,705	293,718	8,602	405	60
	20	1	291,068	289,412	1,387	989	132
	20	2	290,668	288,271	1,439	1,02	228
	20	13	307,333	296,569	6,347	738	178
	20	14	305,050	293,331	8,754	417	62

Aunque se incremente únicamente el número de máquinas en los casos con que se realizaron los experimentos, nos damos cuenta que los Algoritmos 13 y 14 (Tabla XI) vuelven a mostrar menor desempeño que el obtenido con los Algoritmos 1 y 2 para minimizar $\overline{T}$. Lo mismo se observa en la columna del mejor resultado, como ejemplo, tenemos el caso 25 × 10, en donde dichos algoritmos no logran obtener resultados con la misma calidad que los Algoritmos 1 y 2. Conforme se incrementa el número de tareas y máquinas se acentúa más la diferencia. Esto se confirma con la SD_T que guarda el mismo comportamiento. Sin embargo, se observa que para los algoritmos 13 y 14 el $\bar{t}(seg)$ disminuye considerablemente conforme se incrementa el número de tareas y máquinas en comparación con los Algoritmos 1 y 2. SD_T muestra irregularidad ya que para algunos casos con mayor número de máquinas es menor y viceversa sin conservar algún patrón.

De nueva cuenta, los Algoritmos 1 y 2 muestran una secuencia o patrón irregular, es decir, no se define claramente un ganador, en lo que a $\overline{T}$, SD_T y a la obtención del mejor resultado se refiere. Lo mismo se nota para sus correspondientes $\bar{t}(seg)$ y $SD_t(seg)$.

Con respecto a la observación planteada en la Sección IV.2.1.3 en donde se consideraba previamente a los experimentos (IV.2.1.3 y IV.2.1.5) que conforme se duplicara el número de máquinas, $\overline{T}$ disminuiría considerablemente, podemos confirmar que se cumple la hipótesis c) planteada en la Sección IV.2.1.3. Aunque se incremente un número grande de máquinas en la segunda etapa, no se podrá disminuir el retraso debido a que en la primera etapa se procesa la primera parte de todas las tareas provocando que las fechas de finalización de otras tareas que se van a procesar posteriormente, expiren, es

decir, el retraso no podrá ser menor al que se obtiene en la primera etapa, al cual todavía se le tendría que sumar el retraso de la segunda etapa. Este fenómeno se ilustró en la Figura 20 y se corroboró con un caso de 25 × 5.

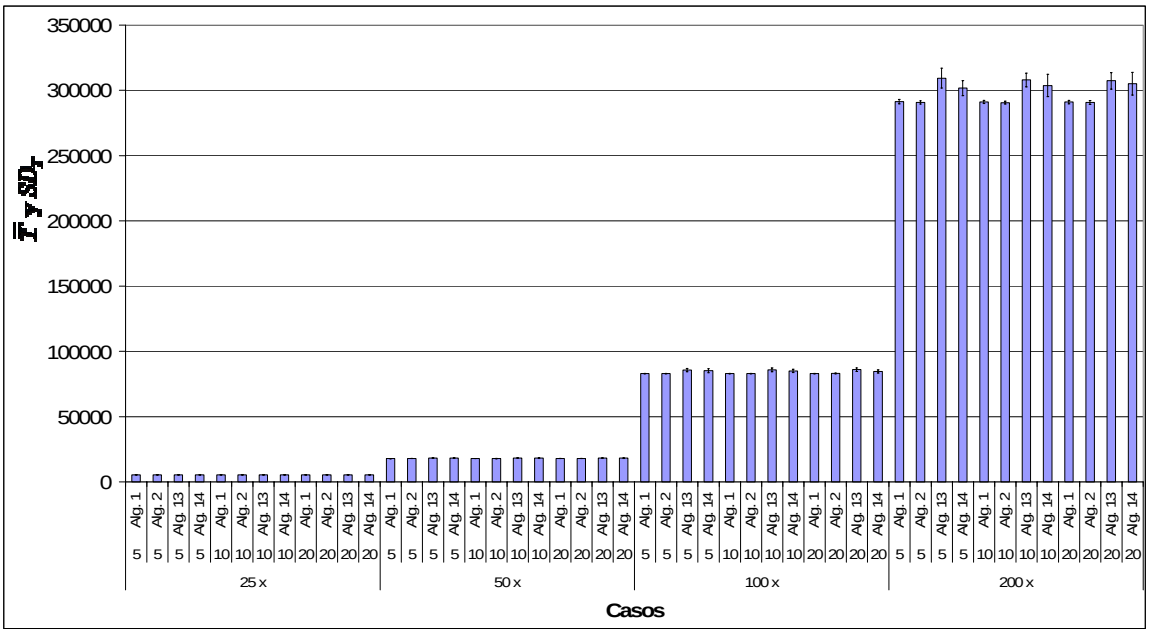


Figura 24: $\overline{T}$ y SD_T de los algoritmos 1, 2 13 y 14. Se observa que conforme se incrementa n y m los Algoritmos 13 y 14 presentan mayor SD_T.

En la Figura 24 se observa claramente que los Algoritmos 1, 2, 13 y 14 logran promedios similares para $\overline{T}$ con los casos de 25 × *m* y 50 × *m* y sus SD_T se comportan de manera similar, además de ser pequeñas (ver Tabla XI). Sin embargo, se empieza a notar una tendencia favorable hacia los Algoritmos 1 y 2 desde los casos de 100 × *m* y se confirma más adelante con los casos de 200 × *m*, en donde existe una gran diferencia tanto para $\overline{T}$ como para SD_T.

IV.2.1.6 Discusión

En seguida se discuten brevemente los resultados obtenidos en cada experimento para estudiar el retraso total.

1. De la Sección IV.2.2.2 se concluye que los Algoritmos 1 y 2 obtienen mejor desempeño que el resto de los algoritmos para minimizar $\overline{T}$ en todos los casos, además, su correspondiente SD_T también es menor en todos los experimentos realizados.
2. En la Sección IV.2.1.2 se observa que al variar únicamente el número de tareas en los experimentos con los Algoritmos 1 y 2, no hay una clara relación entre el incremento del número de tareas y el incremento de $\overline{T}$. Aunque el Algoritmo 2 resuelve el problema en menor tiempo y presenta menor desviación estándar, no es posible asegurar que este es mejor que el Algoritmo 1, ya que este último también presenta en varias ocasiones los mejores resultados y promedios de *T*.
3. Al duplicar el número de máquinas disminuye $\overline{T}$ (sección IV.2.1.3), pero no se observa una relación de proporcionalidad entre esta duplicación y $\overline{T}$. Sin embargo, los AGs consumen más tiempo para resolver el problema y es más notorio en los casos configurados con 20 máquinas en la segunda etapa. Por lo tanto, sugerimos que se efectúen experimentos con casos que contengan un número mucho mayor de máquinas en la segunda etapa y mismos tiempos de procesamiento.

4. Aunque los algoritmos 13 y 14 consumen menos tiempo en la resolución del problema, no presentan buenos resultados en comparación con los algoritmos 1 y 2, para disminuir el retraso total, estos resultados empeoran conforme se incrementa el número de tareas y máquinas (sección IV.2.1.4). El tamaño de la población utilizada por los AG interviene en la calidad de los resultados que se obtienen.
5. Con los experimentos de la sección IV.2.1.5 se confirma que el operador de cruzamiento SB2OX no logra obtener resultados con buena calidad para minimizar el retraso total. Los algoritmos 1 y 2 no muestran ser mejores el uno respecto al otro, ambos son los mejores hasta el momento.

A continuación procedemos a realizar experimentos enfocados al estudio de *Cmax*.

IV.2.2 Resultados de *Cmax*

La metodología sigue siendo la misma que se utilizó en las secciones del retraso total, sin embargo, los experimentos no se inician contrastando a todos los AGs, solo realizamos experimentos con los algoritmos 1, 2, 13 y 14. Se consideraron solamente a estos algoritmos ya que resultados de Ishibuchi y Shibata (2004) y Aceves (2003) mencionan que la combinación de operadores OBX e insert son los que arrojan mejores resultados para el problema de flujo de tareas con representación entera, lo cual ya verificamos en los experimentos para el retraso. El operador SB2OX se toma de Ruíz *et al*., (2005) porque reporta que es el operador que aporta los mejores resultados para *Cmax* y queremos saber si mantiene dicho comportamiento aplicándolo al presente problema. Consideramos al

operador de mutación swap porque en experimentos del retraso total ha mantenido resultados a la par con insert.

IV.2.2.1 *Cmax*: Experimento 1

En la Tabla XII se muestran los experimentos realizados con los Algoritmos 1, 2, 13 y 14 para el estudio del tiempo de finalización total de las tareas (*Cmax*). El parámetro T_{pobl} se establece en 50 para todos los experimentos. Los casos son exactamente los mismos que se utilizaron en los experimentos anteriores (sección IV.2.1.5), con excepción de las fechas de finalización, las cuales no intervienen para el cálculo del *Cmax*.

Tabla XII: Resultados para *Cmax* con los algoritmos 1, 2, 13 y 14.

Caso ($n \times m$)		**Algoritmo**	$\overline{Cmax}$	**Mejor resultado de cada AG.**	$SD_{C\max}$	$\bar{t}(seg)$	$SD_t(seg)$
25 ×	5	1	1,223	1,222	1.06	0.07	0.010
	5	2	1,223	1,222	1.06	0.07	0.007
	5	13	1,223	1,222	1.25	0.07	0.012
	5	14	1,223	1,222	1.23	0.07	0.008
	10	1	993	993	0.73	0.07	0.008
	10	2	993	993	0.73	0.07	0.008
	10	13	993	993	0.80	0.07	0.009
	10	14	993	993	0.73	0.07	0.009
	20	1	977	977	0	0.09	0.01
	20	2	978	977	4.50	0.09	0.01
	20	13	978	977	3.18	0.09	0.02
	20	14	978	977	4.35	0.09	0.01
50 ×	5	1	2,516	2,513	4.84	0.33	0.04
	5	2	2,515	2,513	4.63	0.34	0.04
	5	13	2,518	2,513	5.59	0.36	0.04
	5	14	2,518	2,513	6.25	0.35	0.05
	10	1	2,284	2,284	0.59	0.32	0.03
	10	2	2,284	2,284	0.80	0.31	0.01

Caso ($n \times m$)		Algoritmo	$\overline{Cmax}$	Mejor resultado de cada AG.	$SD_{C\max}$	$\bar{t}(seg)$	$SD_t(seg)$
	10	13	2,284	2,284	1.06	0.07	0.01
	10	14	2,284	2,284	1.06	0.07	0.007
	20	1	2,351	2,350	1.25	0.07	0.012
	20	2	2,351	2,350	1.23	0.07	0.008
	20	13	2,351	2,350	0.73	0.07	0.008
	20	14	2,351	2,350	0.73	0.07	0.008
100 ×	5	1	5,247	5,247	0.80	0.07	0.009
	5	2	5,247	5,247	0.73	0.07	0.009
	5	13	5,247	5,247	0	0.09	0.01
	5	14	5,247	5,247	4.50	0.09	0.01
	10	1	4,891	4,890	3.18	0.09	0.02
	10	2	4,891	4,890	4.35	0.09	0.01
	10	13	4,891	4,890	4.84	0.33	0.04
	10	14	4,892	4,890	4.63	0.3	0.04
	20	1	5,033	5,030	5.59	0.36	0.04
	20	2	5,032	5,030	6.25	0.35	0.05
	20	13	5,032	5,030	0.59	0.32	0.03
	20	14	5,032	5,030	0.80	0.31	0.01
200 ×	5	1	10,410	10,409	0.30	0.34	0.02
	5	2	10,410	10,409	0.76	0.34	0.02
	5	13	10,410	10,409	1.50	0.35	0.03
	5	14	10,410	10,409	1.74	0.34	0.02
	10	1	10,003	10,003	1.52	0.36	0.02
	10	2	10,003	10,003	1.54	0.37	0.04
	10	13	10,003	10,003	0.43	1.8	0.10
	10	14	10,003	10,003	0.48	1.91	0.15
	20	1	10,121	10,119	0.93	2.34	0.26
	20	2	10,121	10,119	0.40	2.19	0.15
	20	13	10,120	10,119	1.10	1.84	0.13
	20	14	10,121	10,119	1.04	1.86	0.21

En ocasiones los Algoritmos 1, 2, 13 y 14 obtienen el mismo resultado (caso 25 × 10) para $\overline{Cmax}$, pero en ocasiones los Algoritmos 1 y 2 son mejores (caso 50 × 5), sin embargo, los algoritmos 12 y 13 presentan mejor $\overline{Cmax}$ para otros casos. Aunque existan

diferencias entre los algoritmos anteriores para minimizar $\overline{Cmax}$, estas son pequeñas. Los cuatro algoritmos coinciden al obtener el mejor resultado. La $SD_{C\max}$ es similar y pequeña para los cuatro algoritmos, como se confirma en la Tabla XII.

Por otro lado, a partir del caso 100 × 5 los Algoritmos 13 y 14 muestran mayor $\bar{t}(seg)$ en la resolución del problema, aunque la diferencia no es grande en comparación con los algoritmos 1 y 2. Solamente del caso 200 × 5 en adelante es cuando $SD_t(seg)$ inicia a crecer por encima de un segundo y llega a crecer hasta cerca de tres segundos en los casos 200 × 20.

Al comparar los resultados concernientes al caso 25 × 5 con los resultados del caso 25 × 10 de la Tabla XII, se observa superioridad en el desempeño para este último, sin embargo, al comparar este caso (25 × 10) con el caso 25 × 20 se observa menor desempeño para el último en cuanto a $\overline{Cmax}$ se refiere. En general, se espera que conforme se aumentan el número de máquinas en la segunda etapa del sistema, se obtenga mejor calidad para $\overline{Cmax}$, aunque a cambio se incremente el tiempo de resolución del problema. El fenómeno ocurrido en éstos experimentos se puede atribuir a que no son las mismas 25 tareas con los mismos dos tiempos de procesamiento en cada una de ellas para los tres casos (cargas) y que el algoritmo sí encuentre el óptimo.

La idea aquí expuesta se corrobora más adelante en la Sección IV.2.2.2 del presente capítulo con experimentos en los que se mantienen las mismas tareas (idénticas) y que sólo se varió el número de máquinas.

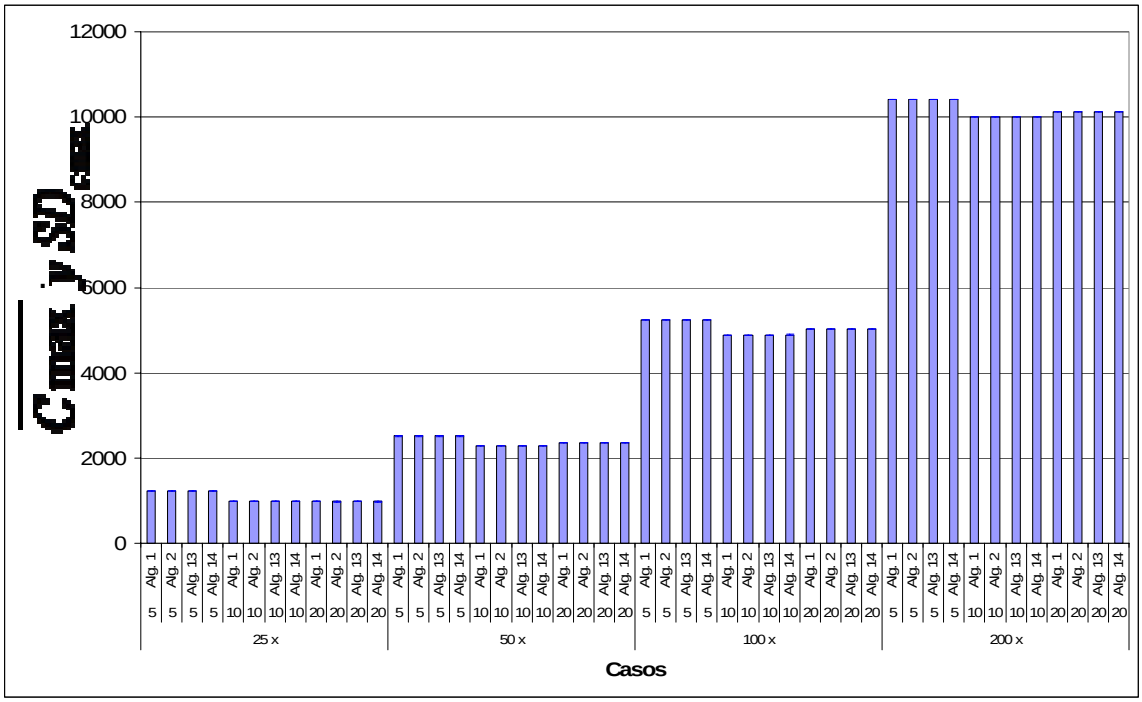


Figura 25: $\overline{Cmax}$ y $SD_{c\max}$ para los algoritmos 1,2, 13 y 14.

En la Figura 25 se puede ver que $\overline{Cmax}$ es similar para los cuatro algoritmos cuando los casos que tienen el mismo número de tareas y de máquinas. Lo mismo se observa para las desviaciones estándar las cuales, además de ser pequeñas (ver Tabla XII), no crecen de manera considerable conforme se incrementan las tareas o máquinas. Tomando en cuenta las descripciones anteriores. Podemos decir que los algoritmos tienen un desempeño similar en cuanto *Cmax* se refiere.

El operador SB2OX ha sido desarrollado por Ruíz *et al*., (2005), reportando que es el operador que mejores resultados arroja para *Cmax*, esto se confirma con los presentes experimentos.

Por otro lado, con la intención de contrastar nuestros resultados obtenidos para *Cmax*, se emplea el AG implementado en el software llamado ProdPlanner, que pertenece a Ruíz García en los mismos casos con los que experimentamos. En la Tabla XIII se muestran los resultados.

Tabla XIII: Soluciones para *Cmax* generadas a partir de ProdPlanner.

Caso ($n \times m$)	$\overline{Cmax}$	$\bar{t}(seg)$
25 × 5	1,222	15
25 × 10	993	2.75
25 × 20	977	5.25
50 × 5	2,513	3
50 × 10	2,289	5.5
50 × 20	2,350	10.5
100 × 5	5,247	6
100 × 10	4,890	11,000
100 × 20	5,030	21,000
200 × 5	10,409	12,000
200 × 10	10,003	22,000
200 × 20	10,119	42,000

La Tabla XIII muestra los tamaños de los casos, el resultado del promedio de *Cmax* y el promedio del tiempo de CPU en segundos. No se presenta $\overline{Cmax}$ ya que no existen variaciones entre las ejecuciones, es decir, no presenta desviación estándar debido a que ProdPlanner siempre mostró el mismo resultado para los $N_{\exp}$.

La Tabla XIV muestra las diferencias entre los resultados obtenidos con el AG del software ProdPlanner y los algoritmos genéticos desarrollados en el presente trabajo.

Tabla XIV: Comparación de resultados con su correspondiente desviación estándar arrojados por el AG de ProdPlanner y los algoritmos desarrollados en el presente trabajo para *Cmax*.

Caso ($n \times m$)	$\overline{Cmax}$ **ProdPlanner**	$\overline{Cmax}$ **AG**	$SD_{C\max}$ **ProdPlanner**	$SD_{C\max}$ **AG**
50 × 5	2,513	2,515	0	4
50 × 10	2,289	2,284	0	0
100 × 20	5,030	5,032	0	3
200 × 20	10,119	10,120	0	1

La Tabla XIV muestra que algunos resultados proporcionados por el AG implementado en ProdPlanner superan a los proporcionados por el algoritmo desarrollado en este trabajo y el único caso en que presenta mejor resultado el AG es para el caso 50 × 10. Sin embargo, es importante hacer notar que en la Tabla XII se presenta un promedio de 30 ejecuciones del AG ya que ProdPlanner no permite ver, manipular o controlar el tamaño de población, número de iteraciones, probabilidades de cruzamiento y mutación, todos estos parámetros se encuentran establecidos por omisión internamente en ProdPlanner y no se pueden conocer o modificar.

IV.2.2.2 *Cmax*: Experimento 2

Los casos para esta segunda serie de experimentos con *Cmax* también se generaron manteniendo los tiempos de procesamiento para el mismo número de tareas, sólo varía el número de máquinas. La intención es investigar lo sucedido en el desempeño de los algoritmos para el *Cmax* en la Sección IV.2.2.1, es decir, con un cierto número de máquinas en la segunda etapa presentó mejores resultados que con un número mayor de éstas. Los algoritmos utilizados son: 1, 2, 13 y 14. T_{plobl} se establece en 100 basándose en

Ishibuchi y Shibata (2004) y Aceves (2003). Los resultados de los experimentos se muestran en la Tabla XV.

Tabla XV: Resultados obtenidos para $\overline{Cmax}$ con los Algoritmos 1, 2, 13 y 14.

Caso ($n \times m$)		Algoritmo	$\overline{Cmax}$	Mejor resultado de cada AG.	$SD_{C\max}$	$\bar{t}(seg)$	$SD_t(seg)$
25 ×	5	1	1,222	1,222	0.36	1.13	0.01
	5	2	1,222	1,222	0.36	1.14	0.02
	5	13	1,222	1,222	0.36	1.12	0.01
	5	14	1,222	1,222	0.61	1.13	0.02
	10	1	1,222	1,222	0.36	1.23	0.01
	10	2	1,222	1,222	0	1.23	0.02
	10	13	1,222	1,222	0.40	1.21	0.02
	10	14	1,222	1,222	0.50	1.22	0.03
	20	1	1,222	1,222	0.36	1.43	0.15
	20	2	1,222	1,222	0	1.42	0.03
	20	13	1,222	1,222	0.36	1.44	0.23
	20	14	1,222	1,222	0.36	1.39	0.03
50 ×	5	1	2,513	2,513	0.54	5.09	0.15
	5	2	2,513	2,513	1.45	5.16	0.59
	5	13	2,515	2,513	3.99	5.94	0.94
	5	14	2,515	2,513	4.63	5.68	0.19
	10	1	2,513	2,513	0.91	5.24	0.12
	10	2	2,513	2,513	0.91	5.41	1.03
	10	13	2,516	2,513	4.34	5.94	0.79
	10	14	2,515	2,513	3.77	5.93	1.00
	20	1	2,514	2,513	1.91	5.74	0.79
	20	2	2,514	2,513	2.75	5.83	0.95
	20	13	2,515	2,513	2.73	6.45	1.09
	20	14	2,514	2,513	2.76	6.30	0.62
100 ×	5	1	5,247	5,247	0.30	31.59	1.07
	5	2	5,247	5,247	0	31.54	0.70
	5	13	5,247	5,247	0.18	38.54	0.98
	5	14	5,247	5,247	0	38.53	0.89
	10	1	5,247	5,247	0	31.69	0.70
	10	2	5,247	5,247	0.30	31.81	0.85
	10	13	5,247	5,247	0	38.51	0.90

Caso ($n \times m$)		Algoritmo	$\overline{Cmax}$	Mejor resultado de cada AG.	$SD_{C\max}$	$\bar{t}(seg)$	$SD_t(seg)$
	10	14	5,247	5,247	0.18	38.48	0.73
	20	1	5,247	5,247	0	32.11	0.49
	20	2	5,247	5,247	0	32.19	0.54
	20	13	5,247	5,247	0.18	39.06	0.81
	20	14	5,247	5,247	0.25	39.33	0.85
200 ×	5	1	10,409	10,409	0.52	236.75	4.47
	5	2	10,409	10,409	0.61	238.40	5.67
	5	13	10,409	10,409	0.43	300.72	8.93
	5	14	10,409	10,409	0.44	307.63	37.17
	10	1	10,410	10,409	0.65	231.89	4.43
	10	2	10,409	10,409	0.44	233.32	6.38
	10	13	10,409	10,409	0.37	296.61	7.92
	10	14	10,410	10,409	0.68	299.17	7.34
	20	1	10,409	10,409	0.54	231.80	3.09
	20	2	10,409	10,409	0.40	234.05	11.55
	20	13	10,409	10,409	0.55	299.03	7.17
	20	14	10,409	10,409	0.53	298.74	7.16

En la Tabla XV se observa que los algoritmos 13 y 14 logran menor desempeño para $\overline{Cmax}$ (por ejemplo, los casos 50 × 5 y 50 × 10), aunque la diferencia es pequeña en comparación con los algoritmos 1 y 2. Conforme se incrementan las tareas y las máquinas le cuesta más trabajo a los algoritmos 13 y 14, sin embargo, la diferencia no es significativa. Los cuatro algoritmos coinciden al obtener el mejor resultado.

La desviación estándar de $\overline{Cmax}$ es pequeña para los cuatro algoritmos en todos los casos. En los casos 50 × *m* los algoritmos 13 y 14 tienden a obtener una desviación mayor a la de los algoritmos 1 y 2, sin embargo, dicha diferencia no es considerable. Es curioso notar que con la actual población en varias ocasiones los cuatro algoritmos obtienen una desviación cero y la mayoría de las ocasiones la obtiene el Algoritmo 2 (caso 100 × 5).

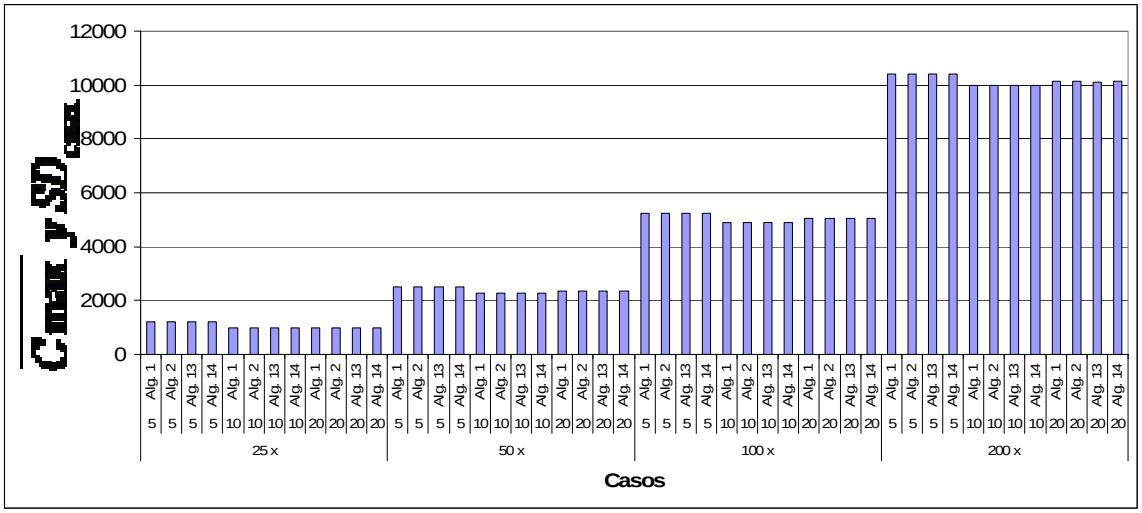


Figura 26: $\overline{Cmax}$ y $SD_{C\max}$ de los AGs 1, 2, 13 y 14. Para los actuales casos se varió únicamente el número de máquinas en la segunda etapa.

En la Figura 26 se ve que el promedio de *Cmax* es el mismo en todos los casos con cualquiera de los cuatro algoritmos. Las respectivas desviaciones estándar son cero (ver Tabla XV). Por esto se puede decir que los cuatro algoritmos muestran el mismo desempeño para minimizar *Cmax.*

Finalmente, y con respecto a la Sección 4.2.2.1 se puede concluir que el tiempo de finalización total no puede ser menor al tiempo de las tareas que se ejecutaron en la primera etapa, es decir, la suma de los tiempos de procesamiento que la permutación de tareas arroja es una cota inferior, y por tanto, no se podrá minimizar el *Cmax* por debajo de este tiempo, pues hay que considerar los tiempos de procesamiento en la segunda etapa.

IV.2.2.3 Discusión

1. De acuerdo a lo observado en los experimentos de la sección IV.2.2.1, los algoritmos 1, 2, 13 y 14 muestran el mismo desempeño para minimizar *Cmax*, sin embargo, no sucede lo mismo para los algoritmos 13 y 14 al tratar de minimizar el retraso total.
2. Con una población de tamaño 100, los algoritmos 1, 2, 13 y 14 obtienen el mismo resultado al optimizar el *Cmax*, aunque los algoritmos 13 y 14 tienden a consumir más tiempo para la resolución del problema, esto se acentúa conforme se incrementan el número de tareas y máquinas en la segunda etapa. Al variar únicamente el número de máquinas, se obtienen los mismos resultados con los cuatro algoritmos (sección IV.2.2.2).
3. Los algoritmos aquí propuestos compiten en calidad con el AG empleado en ProdPlanner.

Después de haber realizado la descripción de experimentos y de los resultados obtenidos, pasaremos al capítulo de conclusiones de este trabajo.

Capítulo V

Conclusiones y Perspectivas de Investigación

V.1 Resumen

En este trabajo se presentó un estudio experimental de los AGs, combinando operadores de cruzamiento OBX, PPX, OSX, Two Point y SB2OX; con los operadores de mutación insert, swap y switch. Se combinó cada uno de los operadores de cruzamiento con cada uno de los operadores de mutación. Se probó cada combinación con una variación del algoritmo evolutivo estándar en diferentes conjuntos de casos del problema de flujo de tareas mono-objetivo. Para cada elemento de cada conjunto del problema de flujo de tareas mono-objetivo se hicieron 30 corridas. Se buscó la combinación que genera las mejores soluciones en términos del retraso total (*T*) o del tiempo máximo de finalización de las tareas (*Cmax*). La regla de despacho que se utilizó para la asignación de recursos entre las etapas de máquinas es la de *list-scheduling*.

V.2 Conclusiones

- El operador de mutación switch es el que en promedio muestra el peor desempeño de todas las combinaciones de operadores. El bajo desempeño se debe principalmente a que sólo altera dos genes contiguos del cromosoma, por tanto su influencia en la configuración de los hijos resultantes es mínima.
- En términos de minimizar el retraso total, se encontró que el operador de cruzamiento OBX es superior en base a que logró obtener el menor promedio y menor desviación estándar.
- Los operadores de mutación insert y swap con el operador de cruzamiento OBX, presentan resultados similares.
- Conforme se incrementa el número de tareas y máquinas, le cuesta más trabajo a los algoritmos disminuir el retraso total y consumen más tiempo para la resolución del caso.
- El operador de cruzamiento SB2OX muestra un desempeño similar al de OBX y los operadores de mutación insert y swap para minimizar el tiempo máximo de finalización de las tareas, pero no así para minimizar el retraso total.
- Se ha verificado que los mismos operadores de cruzamiento y de mutación se cumplen tanto para el caso mono-objetivo como para el caso multi-objetivo.

Después de todos los experimentos realizados con un conjunto de operadores genéticos podemos decir que se cumplió con el objetivo de la tesis, ya que se encontró que existen dos operadores de cruzamiento (OBX y SB2OX) en combinación con dos

operadores de mutación (insert y swap), que generan mejores soluciones para casos específicos del problema de flujo de tareas de dos etapas, en donde la primera etapa tiene sólo una máquina. Además, a diferencia de Lee y Kim (2004) quienes realizaron experimentos con 15 tareas y 5 máquinas en la segunda etapa como máximo, en este trabajo se realizaron experimentos con casos de 200 tareas y 20 máquinas en la segunda etapa.

V.3 Aportaciones

- Se demostró experimentalmente que para un conjunto de casos específicos del problema de flujo de tareas (versión permutación) el operador OBX y los operadores de mutación insert y swap generan mayor calidad en las soluciones.
- Se desarrolló un conjunto de programas con los que se pueden realizar experimentos con los AGs (ver Apéndice C), diferentes reglas de despacho que permiten experimentar con diferentes casos de prueba. Dichos programas permiten optimizar diferentes criterios, así como generar distintos casos de fechas límite.

V.4. Trabajo futuro

- Investigar las razones teóricas del por qué OBX combinado con insert y swap son los operadores genéticos que producen mayor calidad en las soluciones.
- Determinar una cota inferior tanto para el retraso total como para el tiempo de finalización total de las tareas, con el fin de usar dicha cota para validar la calidad de los resultados obtenidos, o bien usarla para modificar el criterio de paro.
- Aplicar los operadores de cruzamiento y mutación de este trabajo a otros criterios de optimización, a fin de saber con qué criterios son más eficientes dichos operadores.
- Realizar un estudio del problema 1 + *m* usando otras heurísticas de aproximación y compararlas con los AGs aquí propuestos.
- Extender el uso de los AGs para los casos *m* + 1 y *m* + *n* del flujo de tareas.
- Realizar un análisis estadístico de los resultados para comprobar si diferencias son o no significativas.

Bibliografía

Aceves, R. (2003). "Estudio de Operadores Genéticos para un Problema de calendarización Multi-Objetivo," *Centro de investigación Científica y de estudios Superiores de Ensenada.* Tesis de Maestría.

Allaoui, H. y Artiba, A. (2006). Scheduling two-stage hybrid flow shop with availability constraints. *Computers & Operations Research,* **33**:1399–1419.

Ben-Daya M. y Al-Fawzan M. (1998). "A tabu search approach for the flow shop scheduling problem," *European Journal of Operacional Research* **109** 88-95.

Bertel, S. y Billaut, J.-C. (2004). A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. *European Journal of Operational Research,* 159: 651-662.

Bierwirth, C. Mattfeld, D. C. and Kopfer, H. (1996). "On permutations representations for scheduling problems," *Lecture Notes in Computer Science,* **1141:** 960-970.

Brindle, A. (1981). "Genetic Algorithms for Function Optimization," *PhD thesis, department of Computer Scince, Univrsity of Alberta, Edmonton, Alberta.*

Cartwright, H. M. y A. L. Tuson. (1994). "Genetic algorithm and flowshop scheduling: Towards the development of a real-time process control system," Fogarty, T. C., editor, "Proceedings of the AISB Conference on Evolutionary Computing:, Leeds, England, **865,** of LNCS,. Springer-Verlag. 277-290 p.

Chang, J., Yan, W. y Shao, H. (2004). Scheduling a two-stage no-wait hybrid flowshop with separated setup and removal times. *Proceeding of the 2004 American Control Conference Boston, Massachusetts June 30.* 1412-1416.

Chen B., Glass C. A., Potts C. N. and Strusevich V. A., (1996). "A new heuristic for three machine flow shop scheduling," *Operations Research* **44** 891-898.

Coello, C. A., D. A. van Veldhuizen, y G. B. Lamont (2003). "*Evolutionary algorithms for solving multi-objective problems*". Kluwer Academic Publishers, London. 580 pp.

Companys, R. (1966). "Métodos heurísticos en la resolución del problema del Taller Mecánico," *Estudios Empresariales,* **66:** (2).

Dimopoulos C. y Zalzala A. M. S. (2000). "Recent Developments in evolutinary computation for manufacturing optimization: Problems, solutions, and comparisons," *IEEE Trans. on Evolutionary Computation* **4** 93-113.

Dudek R. A., Panwalkar S. S. and Smith M. L., (1992) "The lessons of flowshop scheduling research," *Operations Research* **40** 7-13

French, S. (1982). "Sequencing and scheduling: An introduction to the mathemathics of the job shop," *Ellis Horwood Limited, Chichester, England.* 245 pp.

Garey, M.R., Johnson D. S., y Sethi R. (1976). "The complexity of flowshop and jobshop scheduling," *Mathematics of Operations Research,* 1(2):117-129 p.

Garey M. R., johnson D. S. (1979). "Computers and Intractability: A Guide to the Theory of NP-Completeness," San Francisco, CA, W. H. Freeman, 340.

Gen, M. y Cheng, R. (1997) "Genetic algorithms and Engineering Design." (John Wiley & sons, New York).

Gen, M. y R. Cheng 2000. \Genetic algorithms & engineering optimization". John Wiley & Sons, New York.

Goldberg, D. E. (1989). *Genetic Algothims in Search, Optimization, and Machina Learning* (Addison-wesley, reading).

Guirchoun, S., Martineau, P., y Billaut, J.C. (2005). Total completion time minimization in a computer system with a server and two parallel processors". *Computers & Operations Research*, 32:599–611.

Gupta, J. N. D. (1988). Two-Stage, Hybrid Flow shop Scheduling Problem. *Journal of the Operational Research Society*, 39(4):359–364.

Gupta, J. N. D. y Tunc, E. A. (1991). Schedules for a two-stage hybrid flowshop with parallel machines at the second stage. *International Journal of Production Research*, 29(7):1489–1502.

Gupta, J. N. D. y Tunc, E. A. (1998). Minimizing tardy jobs in a two-stage hybrid flowshop. *International Journal of Production Research*, **36(9)**:2397–2417.

Holland H. J. (1962). "concerning efficient adaptative systems," In M. C. Yovits, G. T. Jacobi, and G. D. Goldstein, editors, *Self-Organizing Systems* pages 215-230. Spartan Books, Washington, D.C.

Holland, J. H. (1975). "A daptation in natural and artificial system." The University of Michigan Press, Ann Arbor, Michigan. 211 pp.

Ishibuchi H. y Shibata, Y. (2004). Titulo del libro, Single-Objetive and Multi-Objetive Evolutionary Flowshop Scheduling. Department of Industrial engineering, Osaka Prefecture University. 1-1 Gakuen-cho, Sakai, Osaka 599-8531, Japan.

Ishibuchi H., Misaki S. and Tanaka H. (1995). "Modified Simulated annealing algirithms for the flor shop sequencing problem," *European Journal of Operacional Research* **81** 388-398.

Jain, N. y T. P. Bagachi. (2000). "Hybridized GAs: Some new results in Flowshop scheduling," URL. citeseer.nj.nec.com/jain00hybridized.html.

Jin, Z.H., Ohno, K. Ito, T. y Elmaghraby, S.E. (2002). Scheduling Hybrid Flowshops in Printed Circuit Board Assembly Lines. *Production and Operations Management Society*, **11**:216-230.

Johnson S. M., (1954) "Optimal two- and three-stage production schedules with setup times included," *Naval Research Logistics Quarterly* **1**:61-68.

Ku, H. M. A., D. Rajagolapan, y I. A. Karimi (1993). "Scheduling in batch processes". *Chem. Eng. Prog.*, 83(8):35-45 p.

Lee, G.C. y Kim, Y.D. (2004). A branch-and-bound algorithm for a two-stage hybrid flowshop scheduling problem minimizing total tardiness. *International Journal of Production Research*, **42(22)**:4731 -4743.

Lee, C-Y. y Vairaktarakis, G.L. (1994). Minimizing makespan in hybrid flowshops. *Operational Research Letters*, **16**:149–58.

Lin, H. y Liao, C. (2003). A case study in a two-stage hybrid flow shop with setup time and dedicated machines. *International Journal of Production Economics*, **86**:133–143.

Mitchell, T. (1996). "An introduction to genetic algorithms." The MIT Press, Cambridge, Massachussetts. 224 pp.

Morita, H. y Shio, N. (2005). Hybrid Branch and Bound Method with Genetic Algorithm for Flexible Flowshop Scheduling Problem. *JSME International Journal*, **48(1)**:46-52.

Murata, T. Ishibuchi, H. y Tanaka, H. (1996). "Genetic Algorithms for flowshop scheduling problems," *Copmputer and Industrial Engineering* **30:** 1061-1071.

Nawaz M., Enscore E. y Ham I. (1983). "A heuristic algorithm for the m-machine n-job flow shop sequencing problem," *OMEGA* **11:** 91-95.

Osman I. H. y Potts C. N. (1989). “Simulated annealing for permutation flor-shop scheduling,” *OMEGA* **17:** 551-557.

Park Y. B., Pedgen C. D. y Enscore E. E. (1984). “A survey and evaluation of estatic flowshop scheduling heuristics,” *International Journal of Production Research* **22:** 127-141.

Portmann, M. C., Vignier, A., Dardilhac, D., y Dezalay, D. (1998). Branch and bound crossed with GA to solve hybrid flowshops. *European Journal of Operational Research*, **107**:389-400.

Reeves, C. R. (1995). “A genetic algorithm for flowshop sequencing,” Computers and Operations Research **22 (1),** 5-13.

Riane, F., Artiba, A. y Elmaghraby, S.E. (2002). Sequencing a hybrid two-stage flowshop with dedicated machines. *International Journal of Production Research, **40(17)***:4353-4380.

Rudolph Gunter, (1994). “Convergence Analysis of Canonical Genetic Algorithms,” IEEE *Transactions on Neuronal Networks*, **5**:96-101, January 1994.

Ruíz, R. y Maroto C. (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research **169**:781-800.

Ruíz, R. Maroto C. y Alcaraz, J. (2005). “Solving the flowshop scheduling problemwith sequence dependent setup times using advanced metaheuristics,” *European Journal of Operacional Reserch* **165**: 34-54.

Sriskandarajah, C. y Sethi, S. P. (1989). Scheduling algorithms for flexible flowshops: Worst and average case performance. *European Journal of Operational Research*, **43**:143–160.

Syswerda, G. (1991). “Handbook of genetic algorithms,” International Thomson Publising, capitulo 21: *Schedule optimization using genetic algorithms*, 332-249 p.

Taillard E. (1990). “Some efficient heuristic methods for the flow shop sequencing problem,” *European Journal of Operacional Research* **47:** 65-74.

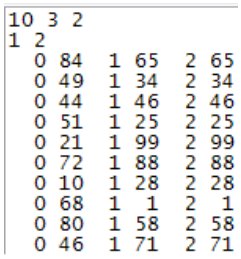
Ulder, N. L., E. H. Aarts, H. J. Bandelt, P. J. M Van Laarhoven, y E. Pesch (1991). “Genetic Local Search algorithm for the traveling salesman problem,” En: Schwefel, H. P. y R. Manner, editors, “Parallel Problem Solving from Nature”, Dortmund, Germany, volume 496 of LNCS. Berlin: springer-Verlag. 109-116p.

Vazquez, J.A. y Salhi, A. (2005). Performance of Single Stage Representation Genetic Algorithms in Scheduling Flexible Flow Shops. *IEEE 0-7803-9363-5/05/2005*.

Wang, L. y Li, D. (2002). A Scheduling Algorithm for Flexible Flow Shop Problem. Proceedings of the 4th World Congress on Intelligent Control and Automation. June 10-14, 2002.

Apéndice A

Ejemplo de un Caso Estándar



```
10 3 2
1 2
  0 84  1 65  2 65
  0 49  1 34  2 34
  0 44  1 46  2 46
  0 51  1 25  2 25
  0 21  1 99  2 99
  0 72  1 88  2 88
  0 10  1 28  2 28
  0 68  1  1  2  1
  0 80  1 58  2 58
  0 46  1 71  2 71
```

Figura 27. Tiempos de procesamiento para el caso de 10 tareas y 2 máquinas en la segunda etapa,

El primer renglón que pertenece al encabezado de la Figura 27 describe el número de tareas totales (diez) que contiene el archivo, las máquinas totales en el sistema (tres), el número de etapas de máquinas (dos). El segundo renglón describe el número de máquinas en cada etapa: una en la primera y dos máquinas en la segunda etapa. Posteriormente, la primera columna indica el índice de la máquina en la primera etapa (cero) seguida de la columna que indica el tiempo de procesamiento de cada tarea para esa etapa. El resto de las columnas se interpretan en forma análoga considerando la segunda etapa. Cada tarea tiene el mismo tiempo de procesamiento en cualquiera de las máquinas de la segunda etapa, debido a que éstas son idénticas.

Apéndice B

Ejemplo de un Archivo Salida

En la Figura 28 se muestra un fragmento de uno de los archivos de salida. En éstos se guardan los resultados obtenidos al ejecutar el AG. Los comentarios en el encabezado indican el objetivo a minimizar, mientras que el número 1 en el cuarto renglón indica la cantidad de columnas que se encuentran debajo de él. Cada renglón de la columna es la salida de una ejecución del algoritmo.

```
; Main
; Minimize
; Tardiness
1
  5973.000
  5972.000
  5967.000
  5967.000
  5967.000
  5993.000
  5967.000
  5996.000
  5993.000
  5967.000
  5978.000
  5967.000
  5967.000
  5967.000
  5993.000
  5967.000
  5967.000
  5967.000
  5993.000
  5993.000
  5967.000
  5984.000
  5980.000
  5967.000
  5967.000
  5972.000
  5993.000
  5984.000
  5978.000
  5967.000
```

Figura 28. Fragmento de un archivo que muestra la columna de salida por el AG.

Apéndice C

Pseudocódigo

Las convenciones que se enumeran en seguida son aplicadas para el pseudocódigo del programa y fueron tomadas con ligeras variaciones de las propuestas por Cormen *et al.*, (1990). Se anexa un segmento de pseudocódigo que se utiliza para el ordenamiento por inserción sólo para mostrar el uso de las convenciones.

ORDENAMIEN TO_INSERSI ON(A)

1. **para** $j \leftarrow 2$ **hasta** *length*$[A]$ **incremento** $j \leftarrow j+1$
2. **hacer** $key \leftarrow A[j]$
3. ▷ Insertar $A[j]$ en la secuencia ordenada $A[1 \otimes j-1]$.
4. $i \leftarrow j-1$
5. **mientras** $i > 0$ y $A[i] > key$
6. **hacer** $A[i+1] \leftarrow A[i]$
7. $i \leftarrow i-1$
8. $A[i+1] \leftarrow key$

- La sangría indica el inicio de un bloque de instrucciones. Por ejemplo, el cuerpo del ciclo **para-hasta-incremento** (for) que comienza en la línea 1 contiene de la línea 2 a la 8, y el cuerpo del ciclo **mientras** (while) comienza en la línea 5 y contiene de la línea 6 a la 7 pero no a la línea 8. El estilo de sangría que se utiliza también aplica para la sentencia **si-entonces-si_no** (if-then-else).
- Las estructuras de los ciclos **mientras**, **para**, **repetir** (repeat) junto con las estructuras condicionales **si**, **entonces** y **si_no** se interpretan de forma similar al uso que se les da en ANSI C.

- El símbolo "$\triangleright$" indica que el resto de la línea es un comentario.
- Una asignación múltiple se realiza de la siguiente manera: $i \leftarrow j \leftarrow \ell$, en donde el valor de la expresión ℓ es asignado a i y a j, ésta asignación en forma equivalente se lleva a cabo bajo la siguiente forma: $j \leftarrow \ell$ seguida por la asignación $i \leftarrow j$.
- Las variables tales como i, j y *key* son locales al procedimiento. Las variables globales son declaradas después de los prototipos de procedimientos y antes de las funciones de procedimientos indicadas con la palabra **GLOBAL**.
- Los elementos de los arreglos son acezados especificando el nombre del arreglo y seguido por el índice encerrado entre corchetes cuadrados, lo mismo se cumple para cada matriz. Por ejemplo, $A[i]$ indica el *i*-ésimo elemento del arreglo A, para la matriz $A[i][j]$, significa, el elemento del *i*-ésimo renglón y la *j*-ésima columna de la matriz A. La notación "$\otimes$" es usada para indicar un rango de valores dentro del arreglo. Así, $A[1 \otimes j]$ indica el subarreglo de A que consiste de j elementos $A[1], A[2], ..., A[j]$.
- Los datos son organizados en *objetos*, que a su vez tienen *atributos* o *campos*. Un campo en particular es accedido utilizando el nombre de dicho campo seguido por el nombre del objeto encerrado por corchetes cuadrados. Por ejemplo, un arreglo es tratado como un objeto con el atributo *lenght* que indica el número de elementos contenidos en el mismo arreglo, es decir *length* $[A]$.
- Los parámetros son pasados al procedimiento por valor y separados por coma "," entre ellos mismos. El procedimiento llamado recibe una copia de los parámetros, y si a

dicho parámetro se le asigna un valor fuera del procedimiento, no altera nada dentro del procedimiento llamado.

- Los operadores de tipo bool "*and*" y "*or*" son de corto alcance. Cuando se evalúa la expresión "x *and* y" primero se evalúa x. Si x es evaluado como *FALSO*, entonces la expresión completa no puede ser evaluada como *VERDADERO* y por lo tanto no se evalúa a y, de otra manera, si x es avaluada como *VERDADERO*, entonces se evalúa a y para determinar la expresión completa. Para la expresión "x *or* y", se evalúa la expresión y sólo si x se evalúa como *FALSO*.
- El símbolo igual " = ", indica una comparación entre dos datos u objetos, se utiliza para determinar si ambos son iguales. Los símbolos menor que "<", mayor que ">", menor o igual que "≤" y mayor o igual que "≥" tienen la misma aplicación que en ANSI C.
- El procedimiento aleatorio regresa un número al azar entre 0 (cero) y 1 (uno) si no recibe parámetro alguno (aleatorio()), en caso contrario, regresa un número al azar entre 0 (cero) y el número dado menos 1 (*n*-1).

```
1.    ▷ PROTOTIPOS DE PROCEDIMIENTOS

2.    GENERAR_POBLACION()
3.    GENERAR_INDIVIDUO()
4.    EVALUAR_PADRES()
5.    ESTRATEGIA_DE_ASIGNACION()                ▷ VER NOTA 1.
6.    COPIAR_HIJO_A_PADRE(i, r)
7.    GUARDAR_SUPERINDIVIDUO()
8.    CRUZAMIENTO()
9.    REALIZAR_CRUZA(i)                         ▷ VER NOTA 2.
10.   COPIAR_PADRES_A_HIJOS()                   ▷ VER NOTA 3.
11.   MUTACION()
12.   REALIZAR_MUTACION()                       ▷ VER NOTA 4.
```

```
13.   REEMPLAZAR_PADRES()                          ▷ VER NOTA 5.
14.   LEER()                                       ▷ VER NOTA 6.

15.   ▷ VARIABLES GLOBALES

16.   GLOBAL
17.      Tam_poblacion
18.      Tot_tareas
19.      super_individuo
20.      anterior_super_individuo
21.      retrazo_total
22.      poblacion_padres[1⊗Tam_poblacion][1⊗Tot_tareas]
23.      poblacion_hijos[1⊗Tam_poblacion][1⊗Tot_tareas]
24.      aptitud_de_hijos[1⊗Tam_poblacion]
25.      aptitud_super_individuo
26.      PC
27.      PM

28.   ▷ PROCEDIMIENTO PRINCIPAL

29.   INICIO Programa - "Algoritmo Genético"

30.      aptitud_super_individuo←999999
31.      LEER(Tam_poblacion)
32.      LEER(Tot_tareas)
33.      LEER(PC, PM)
34.      LEER(criterio_paro)
35.      GENERAR_POBLACION()
36.      MIENTRAS contador<criterio_paro
37.         EVALUAR_PADRES()
38.         GUARDAR_SUPERINDIVIDUO()
39.         CRUZAMIENTO()
40.         MUTACION()
41.         REEMPLAZAR_PADRES()
42.         SI super_individuo = anterior_super_individuo ENTONCES
43.            cambio←0
44.         SI_NO
45.            cambio←1
46.            anterior_super_individuo←super_individuo
47.            contador←1
48.         SI cambio=0 ENTONCES
49.            contador←contador + 1

50.     FIN

51.     ▷ PROCEDIMIENTO
52.     INICIO GENERAR_POBLACION()

53.        PARA i←1 HASTA Tam_poblacion INCREMENTO i←i+1
54.           GENERAR_INDIVIDUO()

55.     FIN
```

```
56.    ▷ PROCEDIMIENTO
57.    INICIO GENERAR_INDIVIDUO()

58.       PARA i←1 HASTA Tot_tareas INCREMENTO i←i+1
59.         individuo[i]←aleatorio(Tot_tareas)

60.    FIN

61.    ▷ PROCEDIMIENTO
62.    INICIO EVALUAR_PADRES()

63.       PARA i←1 HASTA Tam_poblacion INCREMENTO i←i+1
64.          retrazo_total←0.0                      ▷ VER NOTA 7.
65.          ESTRATEGIA_DE_ASIGNACION()
66.          aptitud_de_hijos[i]← retrazo_total     ▷ VER NOTA 7.
67.       PARA i←1 HASTA Tam_poblacion INCREMENTO i←i+1
68.          r1←aleatorio(Tam_poblacion)
69.          r2←aleatorio(Tam_poblacion)
70.          SI aptitud_de_hijos[r1]<aptitud_de_hijos[r2] ENTONCES
71.             COPIAR_HIJO_A_PADRE(i,r1)
72.          SI_NO
73.             COPIAR_HIJO_A_PADRE(i,r2)

74.    FIN

75.    ▷ PROCEDIMIENTO
76.    INICIO COPIAR_HIJO_A_PADRE(NUMERICO i,NUMERICO r)

77.       PARA j←1 HASTA Tot_tareas  INCREMENTO j←j+1
78.          poblacion_padres[i][j]←poblacion_hijos[r][j]

79.    FIN

80.    ▷ PROCEDIMIENTO
81.    INICIO GUARDAR_SUPERINDIVIDUO()

82.       PARA i←1 HASTA Tam_poblacion INCREMENTO i←i+1
33.          SI aptitude_de_hijos[i] < aptitud_super_individuo ENTOCES
84.             aptitud_super_individuo←aptitude_de_hijos[i]

85.    FIN

86.    ▷ PROCEDIMIENTO
87.    INICIO CRUZAMIENTO()

88.    COPIAR_PADRES_A_HIJOS()
89.    PARA i←1 HASTA Tam_poblacion INCREMENTO i←i+2
90.       SI aleatorio() < PC
91.          REALIZAR_CRUZA(i)
92.       SI_NO
93.          COPIAR_HIJO_A_PADRE(i,i)
94.          COPIAR_HIJO_A_PADRE(i+1,i+1)

95.    FIN
```

```
96.     ▷ PROCEDIMIENTO
97.     INICIO MUTACION()

98.        PARA i←1 HASTA Tam_poblacion INCREMENTO i←i+1
99.           SI aleatorio() < PM ENTONCES
100.             REALIZAR_MUTACION()

101.    FIN
```

1. ESTRATEGIA_DE_ASIGNACION(): Se aplica una estrategia de asignación de recursos a las tareas que lo solicitan para poder ser procesados en la segunda etapa.

2. REALIZAR_CRUZA(): Aplica alguno de los operadores de cruzamiento descritos.

3. COPIAR_PADRES_A_HIJOS(): Se copian idénticamente todos los elementos de los padres a los hijos.

4. REALIZAR_MUTACION(): Aplica alguno de los operadores de mutación descritos.

5. REEMPLAZAR_PADRES(): Se copian idénticamente todos los individuos hijos sobre los padres.

6. LEER(): Se leen los datos básicos de entrada para que pueda trabajar el programa.

7. retraso_total: Almacena el retraso total en curso que se está optimizando. En el caso que se quiera optimizar la finalización total de las tareas debe cambiarse por "*Cmax*".