BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

ALGORITMO GENÉTICO PARA LA OPTIMIZACIÓN MULTIOBJETIVO DE PROBLEMAS DE CALENDARIZACIÓN CON TRANSFERENCIA CERO (FLOWSHOP)

Trabajo de tesis para obtener el grado de Maestro en Ciencias de la Computación que presenta

María Magdalena Apasra Bandala Garcés

Asesor

Dra. María Auxilio Osorio Lama

Puebla, Pue. **Noviembre 2005**

Introducción

Diferentes problemas del mundo real tienen más de un objetivo a optimizarse, generalmente estos están en conflicto entre sí, a este tipo de problemas se le llama problemas de optimización multiobjetivo. Los problemas de optimización multiobjetivo requieren de técnicas alternativas de solución pues los métodos clásicos no son apropiados para este tipo de problemas ya que generalmente ofrecen una única solución cuando los problemas multiobjetivo requieren de múltiples soluciones. Existen diversos métodos de solución para problemas multiobjetivo, sin embargo los algoritmos genéticos han demostrado ser una herramienta muy poderosa para dar solución a este tipo de problemas ofreciendo mayor flexibilidad debido a la posibilidad de aplicarlos a un gran número de problemas.

El objetivo principal del presente trabajo es aportar una alternativa de solución al problema optimización Multiobjetivo del problema de calendarización de trabajos con transferencia cero, proponiendo un algoritmo genético basado en el método de partición de Pareto que ha sido usado por otras propuestas y se ha observado un buen comportamiento en el mismo. El problema de calendarización de trabajos con transferencia cero como se sabe es difícil de resolver pues es un problema combinatorio y considerado de la clase NP-completo. En el programa se implementaron diferentes operadores genéticos de cruza y mutación para seleccionar aquellos que proporcionen una mejor convergencia del algoritmo. Además se implementó una forma diferente de selección en donde no se usa la selección por ruleta, este tipo de selección garantiza diversidad en la población al no permitir hijos iguales a los padres dando como resultado mayor exploración del espacio objetivo.

El trabajo está organizado en 4 capítulos. Se presenta en el primer capitulo el modelo del problema de calendarización con transferencia cero caso mono objetivo, para mejor comprensión del caso multiobjetivo, además de las principales técnicas que se han planteado para solucionar al problema de calendarización con transferencia cero para un objetivo. El capitulo 2 se refiere al problema de Optimización Multiobjetivo donde se presenta los conceptos y técnicas de la teoría de Optimización Multiobjetivo que se han usado para darle solución, también se plantea el problema Multiobjetivo de calendarización con transferencia cero, sus antecedentes y los principales algoritmos que han aparecido para su solución. El capitulo 3 se refiere al algoritmo propuesto presentando los conceptos y parámetros usados para el diseño e implementación computacional. En el capitulo 4 muestra la aplicación en problemas concretos y conclusiones.

1 Calendarización de Trabajos con Transferencia Cero (*Flowshop*)

1.1 Calendarización

La calendarización es un proceso de optimización por el cual recursos limitados son asignados en actividades paralelas y secuénciales en un tiempo promedio. Tales situaciones se presentan en la industria, ingeniería, instituciones educativas, transporte y muchos campos más, donde ayudan a resolver problemas como la planeación de producción, organización en mano de obra, horarios, control de procesos etc. Encontrar una buena calendarización permite la optimización de algunos criterios como tiempo, costo, espacio en almacenes y otros.

Resolver este tipo de problemas requiere la búsqueda de una solución óptima entre una gran cantidad de posibles soluciones. Típicamente la tarea es compleja limitando la utilidad práctica de la combinatoria, programación matemática y otros métodos analíticos para resolver el problema de calendarización eficientemente. Tales problemas son llamados problemas de optimización combinatoria; el problema de calendarización está clasificado como aquellos problemas especialmente difíciles de resolver ya que, al tratar de solucionarlo el tiempo computacional se incrementa exponencialmente con el tamaño del problema; no existe ningún algoritmo polinomial de solución, este tipo de clasificación es conocido como de la clase NP-completo.

El primer modelo formal para la calendarización de tareas es el diagrama de Gantt (ver figura 1), usado para la planeación de grandes proyectos.

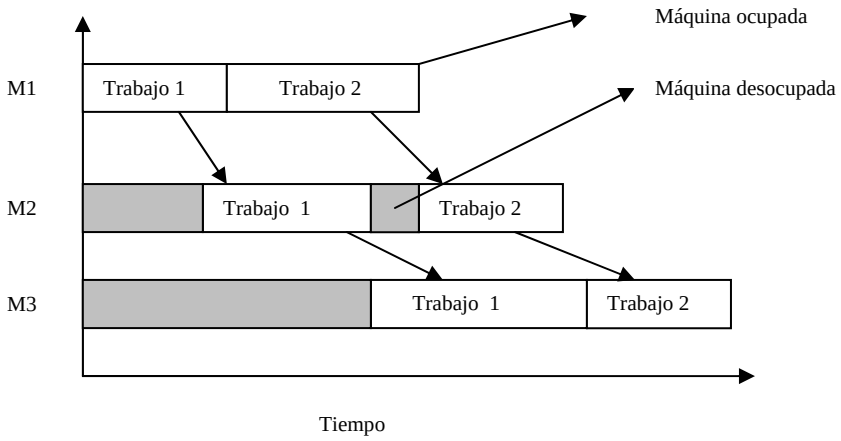

Figura 1 Diagrama de Gantt

1.2 Técnicas de solución para el problema de calendarización

Observando la importancia de la calendarización en el control de costos de producción y organización de procesos, en la década de los 40's, surgen algunos métodos, tomando fuerza en los 70's donde aparecen diversas heurísticas y metaheurísticas para dar solución al problema de calendarización. Estos métodos

son estocásticos y ahora incluyen Recocido Simulado (Simulated Annealing, SA), Búsqueda Tabú (Tabu Search, TS), Algoritmos Genéticos (GA) y técnicas híbridas.

1.2.1 **Recocido Simulado** (Simulated Annealing)

Está basado en una analogía con el proceso físico seguido para obtener sólidos con configuraciones de energía mínima. Se adapta naturalmente a problemas de minimización (el problema puede modificarse para convertirlo en uno de maximización).

Conceptualmente es un método de búsqueda por entornos donde el algoritmo selecciona aleatoriamente un candidato entre los que componen el entorno de la solución actual, si el candidato es mejor que la presente solución en términos del criterio de evaluación entonces es aceptado como solución actual; en caso contrario, será aceptado con una probabilidad que decrece según crezca la diferencia entre los costos de la solución candidata y la solución actual; cuando el candidato no sea aceptado el algoritmo selecciona aleatoriamente otro candidato y se repite el proceso.

1.2.2 **Búsqueda Tabú** (Tabu Search)

Es un procedimiento heurístico para solucionar problemas de optimización combinatoria. La búsqueda Tabú es iniciada en los años 1987 y 1989, es un método que no tiene una base teórica que lo respalde, y su principal objetivo es encontrar soluciones factibles y óptimas en problemas con alto grado combinatorio, mediante una definición adecuada de criterios de selección y parámetros; es una eficiente herramienta de solución a problemas de optimización de gran tamaño, permitiendo obtener soluciones de buena calidad con un bajo consumo de recursos computacionales. Consiste en moverse, paso a paso, desde una solución factible inicial hacia una solución que proporcione el valor mínimo de la función objetivo, definiendo una vecindad en donde se considera para cada punto solución, como un conjunto de soluciones adyacentes a ese punto.

La característica importante de la búsqueda tabú es la construcción de una lista tabú de movimientos, aquellos movimientos que no son permitidos (movimientos tabú) en la presente iteración. La razón de esta lista es de excluir los movimientos que nos pueden regresar a algún punto de una iteración anterior. Las condiciones tabú tienen la meta de prevenir ciclos e inducir la exploración de nuevas regiones.

1.2.3 Algoritmos Genéticos

Los algoritmos genéticos utilizan una estrategia de búsqueda estocástica en un espacio de soluciones potenciales del problema que trata de modelar las leyes de la evolución natural, en particular la herencia genética y la adaptación al entorno.

Consiste en partir de una población inicial de individuos, generada aleatoriamente, cada uno de los cuales representa una solución potencial del problema. Estos individuos se evalúan mediante una función (Fitness) que indica la calidad de la solución o grado de adaptación del individuo al entorno, para después hacer una selección de los mejores individuos. A partir de esta situación inicial se realizan una serie de iteraciones en cada una de las cuales se simula la creación de una nueva generación de individuos a partir de la generación anterior. Este proceso consiste en aplicar los operadores genéticos de selección, cruza y mutación sobre los individuos de la población actual. La solución que ofrece es el mejor individuo encontrado después de un determinado número de generaciones.

1.2.3.1 Codificación

Para un algoritmo genético lo primero que se requiere es determinar en qué espacio se encuentran las posibles soluciones al problema que se pretende resolver. Es necesario codificar de alguna manera el dominio del problema para obtener una estructura manejable que pueda ser manipulada por el Algoritmo.

A los códigos en general se les denomina indistintamente cromosomas, genotipo, genoma, código genético. Es frecuente que el código de los elementos del dominio del problema utilice el código binario. Por lo tanto, el algoritmo recibirá como entrada una población de códigos y a partir de ésta generará nuevas poblaciones de mejores individuos.

1.2.3.2 Función de Evaluación (Fitness)

Para determinar cuales individuos de la población corresponden a buenas propuestas de solución y cuales no, es necesario calificarlos de alguna manera. Cada individuo de cada generación de un algoritmo genético recibe una calificación o, para usar el término biológico, una medida de su grado de adaptación (Fitness). Éste es un número real no negativo tanto más grande cuanto mejor sea la solución propuesta por dicho individuo. El objetivo de este número es que permita distinguir propuestas de soluciones buenas de aquellas que no lo son.

Al determinar que, a cada individuo de la población se le asigna una y sólo una calificación, se está hablando de una función que se denomina función de adaptación cuya evaluación puede no ser sencilla y es lo que en la mayoría de los casos consume más tiempo en la ejecución de un algoritmo genético.

1.2.3.3 Selección

Una vez que se han evaluado a los individuos de una población el algoritmo debe seleccionar a los mejores individuos de la población de acuerdo a ésta evaluación. La selección ocasiona que haya más individuos buenos, explota el conocimiento que se ha obtenido hasta el momento, procurando elegir lo mejor que se haya encontrado, evaluando así el nivel de adaptación de toda la población. Además de la explotación es necesario que exista exploración. El algoritmo genético debe, no sólo seleccionar de entre lo mejor que ha encontrado, sino procurar encontrar mejores individuos.

La selección más común es la selección proporcional o por "ruleta" (Roullette Wheel Selection), éste tipo de selección es proporcional por lo siguiente: supóngase que se suman las calificaciones de todos los individuos de la población y esta suma es considerada el 100% de una circunferencia. A cada individuo se le asigna un trozo que le corresponde de ésta según su aportación a la suma de las calificaciones. Es decir, si la calificación de un individuo es x_i entonces le corresponde un segmento de circunferencia dado por la simple regla de tres:

$$s = 2\pi \frac{x_i}{\sum_j x_j}$$

Si a ésta circunferencia se le considera como una ruleta y se le coloca una lengüeta que roce el borde de ella, la probabilidad de que dicha lengüeta quede en el arco correspondiente al individuo de calificación x_i cuando la rueda se detenga tras realizar algunos giros, es:

$$p(i) = \frac{x_i}{\sum_j x_j}$$

Lo que es proporcional a la calificación x_i del individuo.

Individuo	Calificación	p(i)
1	54	12
2	67,5	15
3	45	10
4	54	12
5	63	14
6	126	28
7	40,5	9
suma	450	100

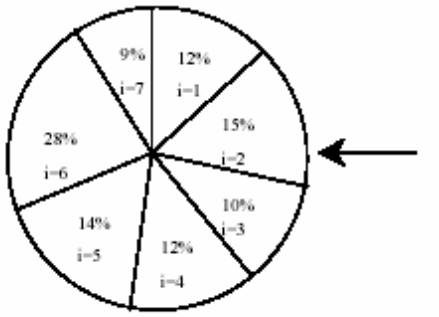

Figura 2 Selección proporcional o por ruleta

1.2.3.4 Cruza

El objetivo del operador genético de cruza es generar nuevos individuos a partir de dos individuos seleccionados en función de su grado de adaptación (Fitness).

Consiste en que dados dos individuos A y B, previamente seleccionados se mezclen, es decir, los códigos genéticos de los individuos se crucen, se fragmenten y recombinen para formar nuevos individuos. Existen muchas maneras de hacer cruzamiento, el más popular es la cruza de un punto, en este tipo de cruza dados dos individuos se elige aleatoriamente un punto de corte en el cromosoma, esto define segmentos izquierdo y derecho en cada cadena que representa al cromosoma, se procede entonces a intercambiar los segmentos de cada individuo y de esto resultan hijos híbridos (ver figura 3).

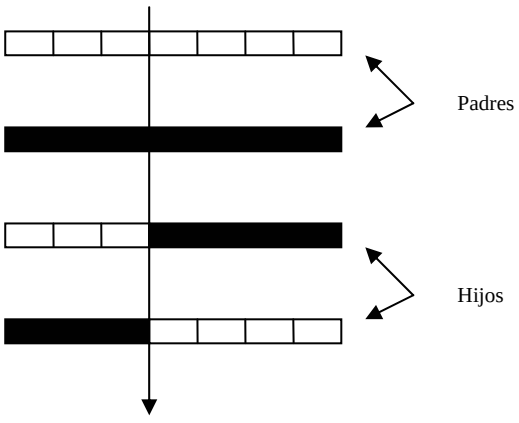

Figura 3 Cruzamiento de un punto.

1.2.3.5 Mutación

El objetivo de la mutación es generar nuevos individuos que exploren regiones del dominio del problema que probablemente no se han visitado aún (mantener la diversidad en la población).

La mutación crea nuevos individuos a partir de los ya existentes haciendo algún pequeño cambio en la cadena que represente al individuo (cromosoma), por ejemplo, un simple intercambio de elementos (ver figura 4).

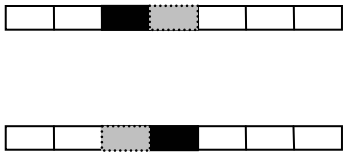

Figura 4 Ejemplo de Mutación

1.2.3.6 Elitismo

Algunos algoritmos genéticos implementan la preservación de los mejores individuos a las generaciones siguientes. Si sólo se hace selección forzando que sea más probable elegir al mejor individuo de la población pero sin guardarlo, es posible que este individuo se pierda y no forme parte de la siguiente generación. Para evitar lo anterior se fuerza la selección de los n mejores individuos de la generación para pasar intactos a la siguiente generación.

1.2.3.7 Algoritmo Genético Simple

Goldberg basado en el trabajo de Holland [13] popularizó el algoritmo genético llamándolo "Algoritmo Genético Simple" (AGS). En el AGS se considera que cada individuo de la población es representado como una cadena de bits, es decir, en código binario. El proceso de un AGS es:

1. Decidir como codificar el dominio del problema.
2. Generar un conjunto aleatorio (población inicial) de *N* posibles soluciones codificadas al problema. A ésta se le llamará la *población actual.*
3. Calificar cada posible solución (individuo) de la población actual.
4. Seleccionar dos individuos de la población actual con una probabilidad proporcional a su calificación.
5. Lanzar una moneda al aire (con una probabilidad p_c cae cara).
6. Si cayó cara mezclar los códigos de los dos individuos seleccionados para formar dos híbridos, a los que llamaremos *nuevos individuos*.
7. Si cayó cruz llamamos a los individuos seleccionados nuevos individuos.
8. Por cada bit de cada nuevo individuo lanzar otra moneda al aire (con probabilidad p_m cae cara).
9. Si cae cara cambiar el bit en turno por su complemento.
10. Si cae cruz el bit permanece inalterado.
11. Incluir a los dos nuevos individuos en una nueva población.
12. Si la nueva población tiene ya *N* individuos, llamarla población actual y regresar al paso 3, a menos que se cumpla alguna condición de terminación.
13. Si no, regresar al paso 4.

El término "lanzar una moneda al aire" se utiliza para representar un experimento de Bernoulli (aquel en el que puedan ocurrir exclusivamente dos eventos posibles, uno con probabilidad p y otro con probabilidad $1-p$). El lanzamiento de una moneda extraña en la que no necesariamente ambas caras son equiprobables.

La condición de terminación puede definirse de muchas maneras, se puede fijar un número máximo de generaciones que se pretende ejecute el algoritmo, o puede decidirse hacer alto cuando la mayoría de la población, digamos el 85% tenga una calificación 8, etc.

El algoritmo genético se caracteriza por:

1. Tamaño de población fijo en todas las generaciones.
2. Selección proporcional (de ruleta).
3. Cruzamiento de un punto. La probabilidad de cruza se mantiene fija para todas las generaciones y todas las parejas.
4. Mutación uniforme (todas las posiciones de las cadenas genéticas tienen la misma probabilidad de ser combinadas). La probabilidad de mutación permanece fija para todas las generaciones y todas las posiciones de los individuos.
5. Selección no elitista. Esto es, no se copian individuos de una generación a otra sin pasar por el proceso de selección aleatoria (en este caso proporcional).

1.3 Calendarización de Máquinas

En esta sección se plantea el problema concreto de calendarización de máquinas en donde se explica que es la calendarización de máquinas, como se plantea y como es el problema matemáticamente.

Se considera un conjunto de tareas o trabajos que deben llevarse a cabo en ciertas máquinas (o etapas) para cumplir cierto objetivo, con distintos tiempos de inicio y fin, donde los datos de los trabajos y las máquinas son conocidos de antemano. Un trabajo puede consistir en diferentes pasos de proceso ocupando diferentes máquinas, se requiere que los trabajos se realicen en una secuencia óptima. Cuando se tienen múltiples máquinas paralelas el problema de calendarización toma el nombre de *Flowshop*.

El *Flowshop* es modelado como un conjunto de múltiples máquinas las cuales deben procesar diferentes tareas para completar ciertos trabajos donde los tiempos de proceso de cada trabajo en las máquinas son fijos y determinados, el número de máquinas y trabajos es conocido. Una estructura de precedencia es aplicada, es decir, cada trabajo después del primero tiene un antecesor y un sucesor directo, cada trabajo requiere una secuencia de procesos para ser terminado, se considera un flujo unidireccional de trabajos, siendo procesados secuencialmente en el mismo orden. Las mediciones de importancia son el tiempo total requerido para procesar todos los trabajos o tiempo total requerido por el último trabajo en la última máquina (*Makespan*), suma total de los tiempos de proceso de cada trabajo (*Flowtime*),tiempo promedio de proceso de los trabajos (*Mean Flowtime*), tiempo de retardo en los trabajos (*Tardiness*) y tiempo de máquina desocupada. El objetivo es encontrar una secuencia o calendarización de los trabajos de tal manera que optimicen cualquier criterio de los mencionados.

Matemáticamente el problema consiste en encontrar una permutación de las trabajos que minimicen el tiempo de procesamiento de todos los trabajos en las máquinas, como se nota para un problema con n trabajos existen $n!$ secuencias o permutaciones posibles, por lo tanto encontrar una solución al problema seria buscar entre las $n!$ posibles soluciones aquella que satisfaga ciertos criterios. Se mencionó

anteriormente que los problemas de calendarización y especialmente el *Flowshop* es un problema combinatorio y forma parte de los problemas NP-Completos.

Queda establecido que:
n número de trabajos que son calendarizados.
m número de máquinas.
t_{ij} tiempo de proceso para el i-ésimo trabajo en la j-ésima máquina.

Makespan y Flowtime son calculados como:

Flowtime $F = \sum_{j=1}^{n} ft_j$ donde ft_j es el tiempo de duración del j- ésimo trabajo desde el inicio en la primera máquina hasta la terminación en la última máquina.

Makespan $C_{\max} = \max_{i \le j \le n} \{ ft_j \}$ (El máximo de los Flowtime parciales).

Ejemplo. Consideremos la calendarización de trabajos con transferencia cero con cuatro trabajos y tres máquinas, los tiempos de procesamiento de los trabajos (t_{ij}) en las máquinas aparecen en la tabla 1 y en la figura 5 el diagrama de Gantt que muestra una solución (3,1,2,4) que optimiza el tiempo total requerido por el último trabajo en la última máquina (*Makespan*), el diagrama nos permite visualizar tiempo de inicio, tiempo de duración en cada máquina y tiempo de terminación de los trabajos.

Trabajos	Máquinas		
	1	2	3
1	4	2	5
2	4	3	3
3	2	1	6
4	5	3	2

Tabla 1 Tiempos de proceso t_{ij}

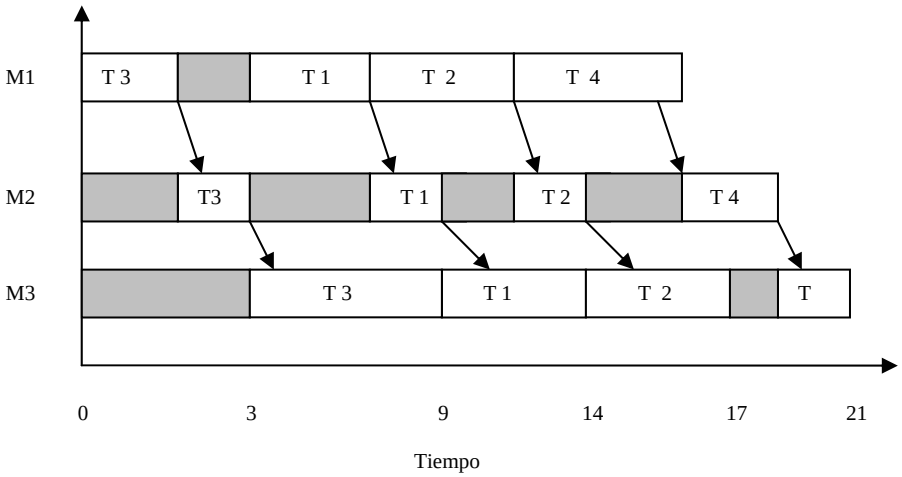

Figura 5 Diagrama de Gantt.

1.3.1 Formulación del modelo Calendarización de Trabajos con Transferencia Cero

Se tiene un conjunto de trabajos, que deben procesarse en ciertas máquinas. No todos los trabajos requieren pasar por todas las máquinas, se asume una transferencia cero o un flujo continuo entre los trabajos. El objetivo es encontrar una secuencia de los trabajos en las máquinas de tal manera que se optimice algún criterio.

El criterio más utilizado es la minimización del tiempo máximo total de todos los trabajos o tiempo completo que toma el último trabajo en la última máquina (*Makespan*).

Además:

- Cada máquina sólo puede procesar un trabajo a la vez.
- Los trabajos sólo pueden ocupar una máquina a la vez.
- El proceso de un trabajo no se puede interrumpir.
- Los trabajos son independientes y se encuentran disponibles en el instante 0.
- Cada uno de los diferentes trabajos siguen una secuencia sin parada en las máquinas.
- Cada trabajo debe ser procesado por completo.
- Los tiempos de proceso son independientes de la calendarización.
- Las máquinas ocasionalmente pueden estar desocupadas.
- El número de trabajos y de máquina es conocido y fijo.
- Los tiempos de procesos son conocidos y fijos.

Este problema puede ser resulto mediante un modelo de programación lineal mixto entero donde, si consideramos que $I = \{1,...,i\}$ conjunto de trabajos, $J = \{1,..., j\}$ conjunto de máquinas, t_{ij} es el tiempo de proceso del i-ésimo trabajo en la j-ésima máquina , t_i tiempo de inicio del i-ésimo trabajo, $t_i + \sum_{j \in J} t_{ij}$ tiempo final de cada trabajo , $t_i + \sum_{\substack{m \in J(i) \\ m \le j}} t_{im}$ terminación del i-ésimo trabajo en la j-ésimo máquina, $t_i + \sum_{m < j} t_{im}$ tiempo inicial del i-ésimo trabajo en la j-ésima máquina, $C_{ik} = \{J(i) \cap J(k)\}$ máquinas con procesos simultáneos para cada par de trabajos i, k y y_{ik} variables binarias que indican que no puede haber colisiones.

Entonces el modelo queda establecido como

Modelo Mixto-Entero

Minimizar Tiempo

s.a. $T \ge t_i + \sum_{j \in J} t_{ij} \qquad \forall i \in I$

$$t_i + \sum_{\substack{m \in J(i) \\ m \le j}} t_{im} \le t_k + \sum_{\substack{m \in J(k) \\ m < j}} t_{km} + M(1 - y_{ik}) \qquad \forall j \in C_{ik}, \forall i,k \in I, i \le k$$

$$t_k + \sum_{\substack{m \in J(k) \\ m \le j}} t_{km} \le t_i + \sum_{m < j} t_{im} + M y_{ik} \qquad \forall j \in C_{ik}, \forall i,k \in I, i \le k$$

$$y_{ik} + y_{ki} = 1 \qquad \forall i,k \in I, i < k$$

$$t_i \ge 0, i \in I, y_{ik} = \{0,1\} \qquad \forall i,k \in I, i < k$$

Para resolver el modelo mixto entero de un problema de calendarización de trabajos con transferencia cero, el tiempo aumenta exponencialmente con respecto al número de variables que se manejan es por eso que para problemas grandes este modelo no es aplicable. Se da un ejemplo de cómo se aplica el modelo en un problema de 3 trabajos y 3 máquinas.

Ejemplo. Considere los siguientes tiempos de proceso para 3 trabajos y 3 máquinas

Trabajos	Máquinas			Tiempo total Por trabajo
	1	2	3	
A	5	0	3	8
B	0	3	2	5
C	2	4	0	6

Tabla 2 Tiempos de proceso t_{ij}

El problema lineal mixto entero de acuerdo al modelo es:

Minimizar Tiempo

s.a.
$$\text{Tiempo} \ge t_A + 8$$
$$\text{Tiempo} \ge t_B + 5$$
$$\text{Tiempo} \ge t_C + 6$$
$$t_A - t_C \le -5 + M(1 - y_{AC})$$
$$t_C - t_A \le -2 + M y_{AC}$$
$$t_B - t_C \le -1 + M(1 - y_{BC})$$
$$t_C - t_B \le -6 + M y_{CB}$$
$$t_A - t_B \le -5 + M(1 - y_{AB})$$
$$t_B - t_A \le M y_{AB}$$

$$\text{Tiempo} \ge 0, t_A, t_B, t_C \ge 0$$

$$y_{AC}, y_{BC}, y_{AB} = \{0,1\}$$

M=20

1.4 Calendarización de Trabajos con Transferencia Cero para dos Máquinas

El problema de calendarización con transferencia cero para dos máquinas y como función objetivo la minimización de *Makespan* es conocido como el problema de Johnson, quien en 1954 es el primero en dar resultados significativos. La regla de Johnson [9] consiste en suponer que se tienen n trabajos con tiempos requeridos de procesamiento t_{i1} , i =1,..., n en la máquina 1 y t_{i2} , i =1,..., n en la máquina 2.

Una secuencia óptima se puede obtener con la siguiente regla para ordenar pares de trabajos.

Teorema 1: El trabajo i precede al trabajo j en una secuencia óptima sí

$$\min[t_{i1}, t_{j2}] \leq \min[t_{i2}, t_{j1}].$$

Este teorema construye una secuencia de trabajos que minimiza *Makespan* para dos máquinas. La regla de Johnson puede extenderse para 3 máquinas, reduciendo el problema a 2 máquinas.

1.5 Métodos de Solución para Calendarización de Trabajos con Transferencia Cero

Los métodos que existen para resolver el problema de calendarización no son viables para un mayor número de trabajos y máquinas, ya que los problemas combinatorios tienden a crecer exponencialmente con el tamaño del problema, por lo que desde hace más de tres décadas se han venido proponiendo métodos heurísticos que encuentran soluciones factibles para este tipo de problemas en un tiempo razonable. Las heurísticas para este problema se pueden dividir en heurísticas constructivas, las cuales construyen una secuencia y las heurísticas de mejora que inician la búsqueda proponiendo una secuencia y tratan de mejorar la secuencia ya existente.

1.5.1 Heurísticas Constructivas

a) **Page** (1961) propuso tres métodos basados en técnicas de ordenación, los tres algoritmos son: Pairing, Merging y Exchanging, obteniendo como resultado que Merging es el mejor de los tres.

b) **Dudek** y **Teuton** (1964) desarrollaron una regla de M máquinas que busca minimizar el tiempo ocioso en la última máquina utilizando una técnica parecida al algoritmo de Johnson para el caso de dos máquinas.

c) **Palmer** (1965) utiliza un índice para cada trabajo basado en los tiempos de proceso que denomina "Slope Index" que proporciona un orden de prioridad para secuenciar los trabajos.

d) **Campbell, Dudek y Smith** (1970) desarrollaron un algoritmo que es básicamente una extensión del algoritmo de Johnson. El algoritmo se conoce como **CDS** y construye un total de $m-1$ secuencias agrupando las m máquinas originales en grupos de 2 y resolviendo el problema resultante con la regla de Johnson.

e) **Gupta** (1971) argumenta que la secuenciación es realmente un problema de ordenar n elementos para minimizar Makespan. Propuso una sencilla modificación a la de Palmer. En este caso los trabajos se secuencian mediante un índice que explota las similitudes entre los problemas de ordenación y secuenciación.

f) **Dannenbring** (1977) propuso un algoritmo llamado "Rapad Access" (RA) mezcla las ideas de Johnson y Palmer. En este caso se define un problema virtual con dos máquinas, como en la heurística CDS, pero en vez de aplicar directamente la regla de Johnson, se definen primero unos esquemas de ponderación para cada una de las dos máquinas virtuales y después se aplica la regla de Johnson. RA proporciona una solución buena en un tiempo razonable.

g) **Nawaz**, **Enscore** y **Ham** (1983) se conoce como NEH está considerada como uno de los mejores métodos para Flowshop. Se basa en la idea de que los trabajos con un tiempo total de proceso más alto en las máquinas deben secuenciarse lo antes posible. La heurística se basa en múltiples inserciones en la secuencia. NEH construye la secuencia insertando un nuevo trabajo en cada paso de la iteración y hallando la mejor calendarización parcial.

h) **Hundal y Rajgopal** (1988) modificaron el algoritmo de Palmer incidiendo en el hecho de que cuando m es impar, el índice de Palmer ignora la máquina $(m+1)/2$. Los autores proponen dos índices alternativos para los trabajos. Con estos dos índices y el original se calculan tres secuencias y la mejor se devuelve como resultado.

i) **Koulamas** (1998) desarrolló una nueva heurística, a la que llamó HFC. En la primera fase, HFC hace uso extensivo de la regla de Johnson. En la segunda fase se intenta mejorar la secuencia, permitiendo que algunos trabajos adelanten a otros en las máquinas.

j) **Davoud Pour** (2001) Propone una heurística basada en el intercambio de trabajos, el método parece comportarse mejor sólo en el caso donde se considera un número grande de máquinas.

k) **Edy Bertolissi** (1999) Propone una heurística para minimizar Flowtime basado en la calendarización parcial por pares de trabajos con el propósito de encontrar los pares que mejoren el tiempo de proceso haciendo un recorrido por todos los pares.

1.5.2 Heurísticas de Mejora

a) **Dannenbring** (1977) desarrolló dos heurísticas simples de mejora: Rapid Access with Close Order Search (RACS) y Rapid Access With Extensive Search (RAES). RACS y RAES toman como solución de partida una secuencia generada con el algoritmo RA. RACS intercambia toda pareja de trabajos de una secuencia dada (n-1 pasos). Las mejores secuencias de entre las n-1 generados se devuelven como resultado. El método RAES, RACS se aplica iterativamente mientras se encuentren mejoras, esto es, RACS se aplica a una secuencia inicial, si se encuentran mejoras, RACS se vuelve aplicar, así hasta que se mejore la secuencia.

b) **Ho y Chang** (1991) desarrollaron un método que trabaja minimizando los tiempos existentes entre la finalización de un trabajo en una máquina y el comienzo del proceso de ese mismo trabajo en la máquina siguiente. A este tiempo los autores lo denominan "gap" . El algoritmo inicia calculando estos tiempos para una secuencia de partida y mediante una serie de cálculos el algoritmo intercambia los trabajos en la secuencia intentando minimizar estos tiempos. La heurística inicia de una solución proporcionada al aplicar el método CDS de Campbell**,** Dudek y Smith.

c) **Suliman** (2000) en la primera fase se genera una solución inicial con la técnica CDS. En la segunda fase se mejora esta secuencia mediante intercambio de trabajos imponiendo unas restricciones de direccionalidad. Esto es, si adelantando un trabajo en la secuencia se mejora la solución, se asume que el trabajo siempre hay que adelantarlo y nunca retrasarlo.

1.5.3 Metaheurísticas

Es necesario hacer una distinción entre heurísticas de mejora y metaheurísticas. En las primeras se mejora una secuencia aplicando un método o algoritmo mientras que en la segunda la secuencia se mejora iterativamente hasta que se alcanza un criterio de paro.

Se mencionan algunos trabajos importantes basados en Recocido simulado (Simulated Annealing), Búsqueda Tabú (Tabu Search), Algoritmos Genéticos (AG) y técnicas híbridas.

a) **Osman y Potts** (1989) desarrollaron un algoritmo tipo Recocido Simulado (Simulated Annealing) utilizando un vecindario de desplazamiento y una evaluación aleatoria del vecindario.

b) **Widmer y Hertz** (1989) desarrollaron un método llamado SPIRIT, que se compone de dos fases. En la primera fase se resuelve mediante una analogía con el problema del agente viajero. En la segunda fase se aplica un algoritmo tipo Búsqueda Tabú (Tabu Search) con parámetros estándar para mejorar la solución obtenida en la fase uno.

c) **Taillard** (1990) presentó un algoritmo similar al de Widmer y Hertz. Se aplica un algoritmo tipo Búsqueda Tabú pero esta vez a una solución que se inicializa mediante el método NEH.

d) **Ogbu y Smith** (1990) Aportaron un algoritmo tipo Recocido Simulado (Simulated Annealing) en el que la solución inicial se genera a partir de las heurísticas de Palmer y Dannenbring. Los autores implementaron un vecindario amplio y una probabilidad de aceptación de las soluciones independientes del cambio en el Makespan de la secuencia.

e) **Werner** (1993) implementó un método iterativo que se basa en generar caminos restringidos en la búsqueda dentro de un vecindario de posibles soluciones. La diferencia con métodos de búsqueda iterativa local es que el método de Werner opera en un vecindario muy amplio y en cada paso sólo se evalúan las secuencias más prometedoras.

f) **Chen, Vempati** y **Aljaber** (1995) desarrollaron un algoritmo genético simple. De la población inicial, m-1 individuos se generan con las m-1

secuencias del método CDS y el individuo en la posición m-ésima se genera con el método RA. El resto de los individuos se generan por simples intercambios entre los individuos anteriores. Sólo aplica un operador de cruza (no hay mutación).

g) **Reeves** (1995) desarrolló un algoritmo genético en el que la descendencia generada en cada paso no reemplaza a los padres, reemplaza aquellos individuos de la población con una función de adaptación inferior a la media. Utiliza un esquema de mutación dinámico que cambia la probabilidad de mutación de acuerdo con la diversidad de la población. La población inicial es aleatoria, exceptuando uno de los individuos, que se genera por el método NEH.

h) **Tsujimura y Kubata** (1994) proponen un algoritmo genético donde usan para representar el esquema de cromosoma la secuencia de los trabajos (cromosoma no binario). Para determinar la función de evaluación usan la inversa del Makespan.

i) **Ishibuchi**, **Misaki** y **Tanaka** (1995) presentaron dos algoritmos tipo Recocido Simulado en el que se busca un comportamiento robusto frente a la secuencia de enfriamiento.

j) **Zegordi, Itoh** y **Enkawa** (1995) presentaron una técnica híbrida incorporando conocimiento del dominio del problema en un algoritmo tipo Recocido Simulado. El algoritmo denominado SA-MDJ, usa una tabla con un índice de movimientos para los trabajos que incorpora reglas para facilitar el proceso de enfriado, de esta manera se requieren menos parámetros de control.

k) **Moccellin** (1995) presentó un Búsqueda Tabú muy parecido al de Widmer y Hertz. La única diferencia estriba en el cálculo de la solución inicial, el resto del algoritmo es muy similar al método SPIRIT.

l) **Murata, Ishibuchi** y **Tanaka** (1996) proponen un algoritmo genético donde usan cruce de dos puntos, una mutación de desplazamiento de los trabajos y la técnica elitista para obtener buenas soluciones. Después implementaron versiones híbridas del algoritmo genético mezclado con Búsqueda Tabú, Recocido Simulado y Búsqueda Local. En estos algoritmos híbridos hay una fase de mejora de las secuencias antes de la selección y el cruce, esta mejora se hace a partir de Búsqueda Tabú, Recocido Simulado o búsqueda local. Los algoritmos híbridos resultaron ser mejores que las versiones simples.

m) **Nowicki** y **Smutnicki** (1996) propusieron un Búsqueda Tabú donde tan sólo se evalúa una parte muy reducida del vecindario de una secuencia. El vecindario se reduce con la idea de bloques de trabajos, donde los movimientos se hacen de un bloque a otro.

n) **Yamada y Reeves** (1997) usan un algoritmo genético híbrido en donde se utiliza un operador de cruza que junta las ideas de cruce genético con búsqueda local.

o) **Stützle** (1998) propone el método "Iterated Local Search" o ILS. El procedimiento parte de una solución inicial, hace una búsqueda local en torno a esta solución, modifica esta solución (como si fuera una mutación), repite la búsqueda local y se considera si acepta la solución.

Si la solución se acepta se repite el proceso desde el principio mejorando en cada iteración la secuencia.

p) **Ben–Daya** y **Al-Fawzan** (1998) implementaron una Búsqueda Tabú con algunas características adicionales como intensificación y diversificación que proporcionan mejoras en los movimientos dentro de la búsqueda.

q) **Moccellin** y **Dos Santos** (2000) Presentaron un híbrido entre Búsqueda Tabú y Recocido Simulado.

r) **Ponnambalam, Aravindan y Chandrasekaran** (2001) usan un algoritmo Genético que utiliza mutación por intercambio de trabajos e inicialización de la población de manera aleatoria.

Año	Autor/es	Heurística	Tipo	Comentario
1954	Johnson	Johnson	C	Exacto para dos máquinas
1961	Page	Page	C	Basado en ordenación
1963	Dudek y Teuton		C	Basado en Johnson
1965	Palmer	Palmer	C	Índices para los trabajos
1970	Campbell, Dudek y Smith	CDS	C	Basado en Johnson
1971	Gupta	Gupta	C	Índices para los trabajos
1977	Dannenbring	RA,RACS,RAES	C/M	3 heurísticas
1983	Nawaz ,Enscore y Ham	NEH	C	Prioridad e inserción
1988	Hundal y Rajgopal	HunRa	C	Basado en Palmer
1991	Ho y Chang	HoCha	M	Minimización de "gap"
1998	Koulamas	Koula	C/M	Basado en Johnson
2000	Suliman	Sulim	M	Intercambios de trabajos
2001	Davoud Pour	Pour	C	Intercambios de trabajos

Tabla 3 Resumen de Heurísticas constructivas (C) y de mejora (M) para Flowshop

Año	Autor/es	Metaheurística	Tipo	Comentario
1989	Osman y Potts	SOAP	SA	
	Widmer y Hertz	Spirit	TS	Solución basada en el TSP
1990	Taillard		TS	
	Ogbu y Smith		SA	
1993	Werner		otro	Algoritmo de rutas
1995	Chen Vempati y Aljaber	GAChen	GA	
	Reeves	GAReev	GA	Mutación adaptativa
	Ishibuchi, Misaki, Tanaka		SA	Se considera 2 SA
	Zegordi, Itoh y Enkawa		SA	Combina conocimiento del problema
	Moccellin		TS	Basado en SPIRIT
1996	Murata, Ishibuchi y Tanaka	GAMIT	Hybrid	GA + Búsqueda Local /SA
	Nowicki y Smutnicki		TS	Bloques de trabajos
1997	Yamada y Reeves		GA	Operadores con conocimiento del problema
1998	Stützle	ILS		
	Ben Daya y Al-Fawzan		otro	Búsqueda local iterativa
			TS	Intensificación + diversificación
2000	Mocellin y Dos Santos		Hybrid	TS+SA
2001	Ponnambalam, Aravindan y Chandrasekaran	GAPAC	GA	

Tabla 4 Metaheurísticas para flowshop (Tipo: Recocido Simulado, SA; Búsqueda Tabú, TS; Algoritmos Genéticos, GA)

En las pruebas realizadas por R. Ruiz y C. Maroto [6] se observó que el método de Nawaz, Enscore y Ham (1983) (NEH) es la mejor heurística. De los métodos metaheurísticos el algoritmo Stützle (1998)(ILS) obtiene soluciones muy cercanas a las óptimas alcanzadas en tiempos razonables en los peores casos. El algoritmo genético de Reeves (1995) resulta ser muy cercano en eficiencia al anterior.

2 Optimización Multiobjetivo

En el mundo real siempre se enfrentan problemas en donde se deben tomar decisiones, tratando de llegar a más de un objetivo. Muchos de los problemas envuelven optimización simultánea de diferentes objetivos en conflicto, este tipo de problemas es llamado optimización multiobjetivo o vectorial, se observa que se encuentra en todas las áreas de estudio y fue desarrollado originalmente en el área económica.

La optimización multiobjetivo consiste en optimizar más de un objetivo sujeto a una o varias restricciones, para simplificar su solución muchos de estos problemas tienden a modelarse como mono-objetivo usando sólo una de las funciones objetivo y manejando las adicionales como restricciones.

Una definición del Problema de Optimización Multiobjetivo (MOP) formulada por Osyczka, (1985) y tomada por Coello [2] establece que un Problema de Optimización Multiobjetivo es aquél que "*Consiste en encontrar un vector de variables de decisión que satisfaga las restricciones y optimice una función vectorial cuyos elementos representan a las funciones objetivo. Dichas funciones conforman una descripción matemática del criterio de desempeño y suelen estar en conflicto entre ellas. En consecuencia, el término optimizar significa encontrar una solución que produzca valores de las funciones objetivo aceptables para el responsable de tomar las decisiones (Decisión Maker)".*

2.1 Conceptos preliminares

Restricciones.

En los problemas de optimización hay restricciones impuestas por las características particulares y los recursos disponibles para el problema a optimizar, estas restricciones son expresadas en forma matemática como desigualdades $g_i(\vec{x}) \geq 0$, i =1,..., m o las igualdades $h_i(\vec{x}) = 0$, i =1,..., p. Las restricciones $g_i(\vec{x})$ y $h_i(\vec{x})$, definen la región factible Ω, y algún punto $\vec{x}^* \in \Omega$ (variables de decisión) define la *solución factible*.

Funciones objetivo.

Las funciones objetivo son denotadas como $f_1(\vec{x}), f_2(\vec{x}),..., f_k(\vec{x})$ donde k es el número de funciones objetivo a optimizar, por lo tanto las funciones objetivo forman un vector función $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}),..., f_k(\vec{x})]^T$.

Problema Multiobjetivo.

El problema de optimización multiobjetivo se define formalmente como:

Definición 1 Hallar un vector $\vec{x}^* = [x_1^*, x_2^*,..., x_n^*]^T$ el cual satisface la m restricciones de desigualdad $g_i(\vec{x}) \geq 0$, i =1,2,..., m las p restricciones de igualdad $h_i(\vec{x}) = 0$, i = 1,2,..., p y que optimice $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}),..., f_k(\vec{x})]^T$.

Tipos de Problemas Multiobjetivo.

En los problemas de optimización multiobjetivo existen tres posibles situaciones: Minimizar todas las funciones objetivo, Maximizar todas las funciones objetivo, Minimizar algunas y Maximizar otras, por razones de simplicidad normalmente son convertidas a una forma Maximizar o Minimizar.

2.2 Técnicas de optimización Multiobjetivo.

La dificultad de algunos de estos problemas consiste en que generalmente no se puede establecer como un modelo matemático que nos lleve a una solución exacta. Las técnicas convencionales de optimización como por ejemplo el método Simplex, Recosido Simulado (Simulated Annealing) y otros son difíciles de extender para el caso multiobjetivo por que estos métodos no son diseñados para obtener múltiples soluciones.

Se han desarrollado muchos algoritmos de programación matemática para resolver problemas multiobjetivo, sin embargo presentan la desventaja de generar una solución a la vez, a pesar de que los problemas multiobjetivo suelen tener múltiples soluciones, normalmente requieren de una solución factible para iniciar la búsqueda. Los métodos heurísticos no garantizan la optimalidad, pero ofrecen una solución factible en un tiempo razonable. Una ventaja importante que presentan las heurísticas respecto a las técnicas que buscan soluciones exactas es que, por lo general, permiten mayor flexibilidad para el manejo de las características del problema. Generalmente ofrecen más de una solución (se aplican muy bien al problema multiobjetivo) lo cual permite ampliar las posibilidades de elección de quien decide, sobre todo cuando existen factores no cuantificables que no han podido ser añadidos al modelo, pero que también deben ser considerados. El inconveniente de métodos heurísticos es que por lo general no es posible conocer la calidad de la solución, es decir, cuán cerca está del óptimo la solución que ofrece la heurística.

La mayoría de las técnicas de optimización multiobjetivo están basadas en el frente de Pareto, damos algunas definiciones y conceptos acerca de la teoría de Pareto.

Optimalidad de Pareto.

Definición 2 Se dice que un punto $\vec{x}^* \in \Omega$ es un óptimo de Pareto si para toda $\vec{x} \in \Omega, I = \{1,2,...k\}$ ya sea, $\forall_{i \in I}(f_i(\vec{x}) = f_i(\vec{x}^*))$ o hay al menos una $i \in I$, tal que $f_i(\vec{x}) > f_i(\vec{x}^*)$ para minimización y $f_i(\vec{x}) < f_i(\vec{x}^*)$ para Maximización.

Dominancia de Pareto.

Definición 3 Un vector $\vec{u} = (u_1, u_2, ..., u_k)$ domina a otro $\vec{v} = (v_1, v_2, ...v_k)$ si y sólo si, $\vec{u}$ es parcialmente menor a $\vec{v}$, entonces $\forall i \in \{1,...,k\}, u_i \leq v_i \wedge \exists i \in \{1,...,k\} : u_i < v_i$. Y es denotado como $\vec{u} \preceq \vec{v}$.

Conjunto de óptimos de Pareto.

Definición 4 Para un problema multiobjetivo dado $\vec{f}(x)$ el conjunto de óptimos de Pareto (ρ^*) se define como $\rho^* := \{x \in \Omega \mid \neg \exists x' \in \Omega \vec{f}(x') \preceq \vec{f}(x)\}$

Frente de Pareto.

En el espacio solución existe una región de puntos frontera, en la cual se encuentran concentrados aquellos puntos que pueden considerarse como óptimos, conjunto de vectores no dominados. Cuando es graficado el espacio objetivo el conjunto de vectores no dominados son conocidos como el frente de Pareto. Entonces la solución al problema multiobjetivo es una familia de puntos conocido como el conjunto óptimo de Pareto (también llamado conjunto no dominado, (ver figura 6). Se define como:

Definición 5 Dado un problema multiobjetivo $\vec{f}(x)$ y un conjunto de óptimos de Pareto ρ^*, el **frente de Pareto** (ρf^*) se define como $\rho f^* := \{\vec{u} = \vec{f} = (f_1(x),..., f_k(x)) \mid x \in \rho^*\}$.

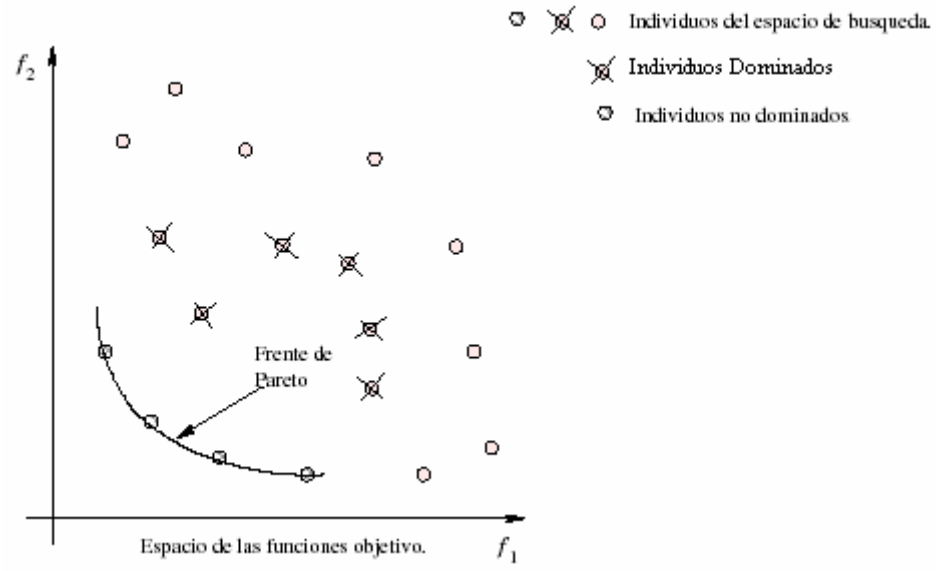

Figura 6 Frente de Pareto

2.3 Algoritmos Evolutivos de Optimización Multiobjetivo

Diversos métodos han surgido para solucionar el problema multiobjetivo, observando que la aplicación de algoritmos evolutivos han tenido mejores resultados, ya que, su éxito se relaciona con la producción de múltiples soluciones en una sola corrida a diferencia de otras heurísticas que sólo nos permiten encontrar una solución a la vez.

La computación evolutiva esta basada en las ideas de la selección natural, y su aplicación en un ambiente artificial. La selección natural es vista como un proceso de optimización, en el que paulatinamente los individuos de la población se van mejorando para adaptarse a su medio. Para la computación evolutiva un individuo es una solución potencial a un problema, codificada de acuerdo con el funcionamiento del algoritmo; y el medio donde se desenvuelve lo componen la función objetivo y las restricciones, las que nos dirán que tan apto es el individuo para

sobrevivir. La computación evolutiva involucra métodos que son poblacionales, lo que significa que trabajan con varias soluciones a la vez.

2.3.1 Clasificación de Algoritmos Evolutivos de Optimización Multiobjetivo (MOEAs)

Este tipo de algoritmos aparecen a finales de los años sesenta, pero han tenido un desarrollo significativo en las dos últimas décadas. Los primeros enfoques son clasificados como Algoritmos Evolutivos de Optimización Multiobjetivo (MOEAs) de primera generación y los enfoques actuales corresponden a la segunda generación desarrollados en la última década, como se muestra en la siguiente figura.

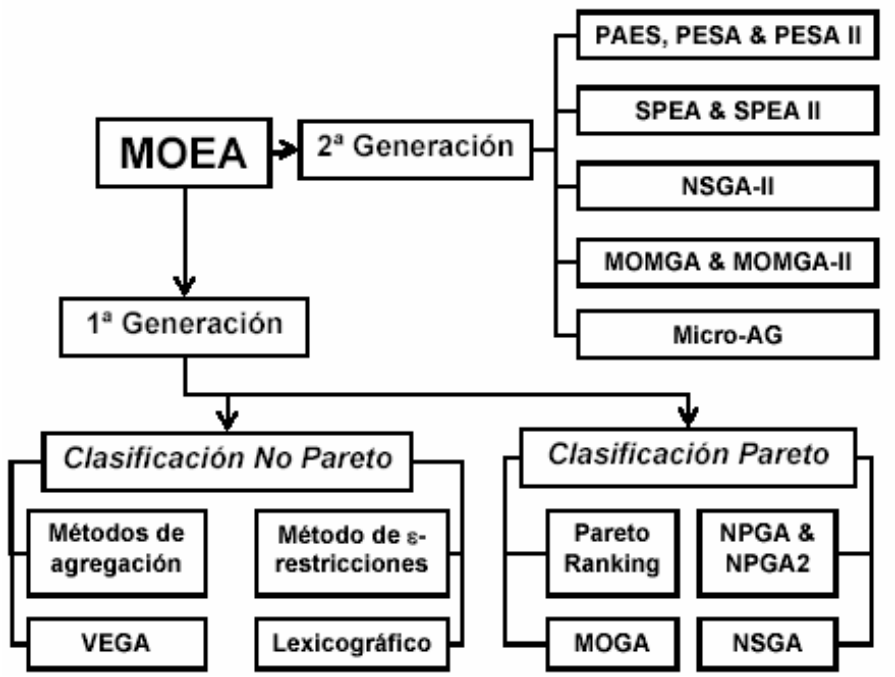

Figura 7 Clasificación según la generación

2.3.1.1 Primera Generación (Inicios de los Algoritmos Evolutivos Multiobjetivo)

MOEAs de primera generación son producto de los primeros ensayos en el área. Esta categoría engloba una gran cantidad de algoritmos evolutivos multiobjetivo, los algoritmos primera generación son subclasificados en métodos con clasificación Pareto y No Pareto (ver figura 9). Esta clasificación se refiere a la función de adaptación (Fitness), los algoritmos con clasificación Pareto son aquellos que poseen una función de adaptación basada en el criterio de dominancia de Pareto. Los algoritmos no Pareto son aquellos cuya función de adaptación no emplea el criterio de dominancia de Pareto.

En general son fáciles de implementar y con una eficiencia aceptable, sin embargo presentan cierta dificultad para generar la totalidad de la frontera de Pareto. Los siguientes son clasificación Pareto.

VEGA (Vector Evaluated Genetic Algorithm), Schaffer (1985) en VEGA, la población total es dividida en k partes iguales (suponiendo que el problema tiene k objetivos), y en cada una de ellas opera la selección tomando en cuenta sólo una función objetivo. Una vez realizada la selección, la población se mezcla para aplicar el resto de los operadores evolutivos. Todo este proceso se repite en cada generación. Un problema evidente en VEGA es que no favorece a todos los objetivos, sino que, prefiere las soluciones mejores en sólo uno de los objetivos.

MOGA (Multi-objetive Genetic Algorithm), Fonseca y Fleming, (1993). Este algoritmo asigna un rango a la población según la dominancia de los individuos, donde el rango es asignado según el número de individuos por el cual un individuo es dominado. Usa el método de nichos (Fitness-sahring) para la distribución de los individuos en la región óptima de Pareto.

NSGA (Non-dominated Sorting Genetic Algorithm), Srinivas y Deb, (1994). Se le asigna a la población un rango en base a la no dominancia. Todos los individuos no dominados en la población actual son identificados y pasan a una lista con un rango igual a 1, en la generación siguiente estos individuos serán removidos y tendrán un rango 2, este proceso continúa hasta terminar. Mantiene la diversidad en la población.

NPGA (Niched Pareto Genetic Algorithm), Horn, Nafpliotis y Goldberg, (1994). A diferencia de los anteriores, que jerarquizan la población completa, éste propone una forma de realizar torneos basados en la dominancia de Pareto, con la finalidad de evitar jerarquizar a toda la población. En NPGA cuando se enfrentan dos individuos en un torneo, se comparan ambos contra una fracción de la población (utilizan el 10%).

2.3.1.2 Segunda generación (MOEAs concepto elitista)

Los MOEAs de segunda generación introducen el concepto de elitismo (preservación de los mejores individuos a las generaciones siguientes) en dos formas principales usando selección de individuos y usando poblaciones secundarias de individuos.

Esta técnica permite mantener a los mejores individuos durante la ejecución del algoritmo, esto es, la conservación de los individuos con mejor grado de adaptación en la población o en una población auxiliar. Estos algoritmos generan la mayor cantidad de soluciones posibles, bien distribuidas y lo más cerca posible de la frontera de Pareto. Distintos investigadores han desarrollado, durante los últimos años una serie de métodos clasificados como segunda generación, enfocados a lograr los siguientes objetivos:

- Realizar la asignación de adaptación y proceso de selección, para guiar la búsqueda en dirección al conjunto óptimo de Pareto.
- Mantener la diversidad dentro de la población, para prevenir convergencias prematuras y alcanzar un frente bien distribuido.

SPEA (Strength Pareto Evolutionary Algorithm), Zitzler y Thiele, (1999). Usa una población auxiliar para seleccionar a los individuos no dominados obtenidos a lo largo del proceso evolutivo. Cada individuo del conjunto externo se le calcula un factor similar al rango de MOGA. Este rango está basado en un factor de resistencia, el cual es proporcional al número de soluciones para el cual cierto individuo es dominado.

PAES (Pareto Archived Evolution Strategy), Knowles y Corne, (1999). Almacena en un archivo externo a los individuos no dominados encontrados a lo largo del proceso evolutivo (elitismo). Cuando se encuentra un individuo no dominado es comparado

con los individuos del archivo externo, y en caso de que nuevamente sea no dominado, puede entrar al mismo archivo. El mecanismo de PAES para mantener diversidad consiste en una rejilla adaptativa.

NSGA II Deb, Agrawal, Pratap y Meyarivan, (2000). Éste mantiene las mejores soluciones halladas incluyendo la siguiente generación. El rango de cada individuo está basado en el concepto de la no dominancia como niveles correspondientes al NSGA. Usa un parámetro de distancia entre los individuos para guardar la diversidad en la población.

PESA (Pareto Enveloped-based Selection Algorithm), Corne, Knowles y Oates, (2000). Este algoritmo hace uso de una población auxiliar donde almacena las mejores soluciones encontradas. El mecanismo de grupo es basado en la división por rejillas formando cubos como en PAES. También usa un criterio de selección para éste concepto.

NSGA-II con control de elitismo, Ded y Goel (2001). Introduce un operador que controla elitismo y otro para mantener la diversidad que no requiere parámetros. Produce un mejor equilibrio entre la exploración y la explotación.

SPEA2 Zitzler, Laumanns y Thiele, (2001). Basado en SPEA, la función de evaluación esta basada en la no dominancia de Pareto más la densidad de la población en la vecindad del individuo. Introduce un operador para guardar la diversidad en la población.

PRIMERA GENERACIÓN	**Año**	**Autor**	**Características**
VEGA	1985	Schaffer	Divide a la población según el número de objetivos, prioriza un sólo objetivo.
MOGA	1993	Fonseca y Fleming	Jerarquiza la población según la dominancia de Pareto. Técnica de nichos.
NSGA	1994	Srinivas y Deb	Jerarquiza la población por capas. Mantiene la diversidad.
NPGA	1994	Horn, Nafpliotis y Goldberg	Realiza torneos entre individuos. Jerarquiza a la población en base al torneo.
SEGUNDA GENERACIÓN Elitismo + Nichos	**Año**	**Autor**	**Características**
SPEA	1999	Zitzler y Thiele	Usa una población auxiliar para los individuos no dominados. Jerarquiza la población externa.
PAES	1999	Knowles y Corne	Usa una población auxiliar para los individuos no dominados. Divide el espacio objetivo en rejillas. Selección por elitismo y Pareto.
NSGA II	2000	Deb, Agrawal, Pratap y Meyarivan	Jerarquiza la población según NSGA.
PESA	2000	Corne, Knowles y Oates	Usa una población auxiliar para los individuos no dominados. Divide el espacio objetivo en rejillas y usa un criterio de Selección por elitismo y Pareto.
NSGA-II (Elitismo controlado)	2001	Ded y Goel	Control elitismo.
SPEA2	2001	Zitzler, Laumanns y Thiele	Basado en la dominancia de Pareto. Guarda la diversidad.

Tabla 5 Resumen Según Generación

2.4 Calendarización de Trabajos con Transferencia Cero caso Multiobjetivo

Teóricamente los trabajos de calendarización multicriterio inician en la década de los 1970´s, Heck y Roberts (1972) presentan un método donde se consideran criterios secundarios. Emmons (1975) resuelve el problema de n trabajos y una máquina tomando dos funciones objetivo como dos criterios separados. Van Wassenhove y Gelders (1980) resuelven el mismo problema usando como criterios el máximo retardo y una función de costos. Van Wassenhove y Baker (1982) desarrollaron un frente eficiente (frente de Pareto) para una simple máquina usando dos objetivos. Dileepan y Sen (1988) estudiaron el problema bi-criterio para una máquina.

En el problema de calendarización multiobjetivo la consideración de más de un objetivo puede causar conflictos entre los objetivos, ya que las condiciones que favorezcan la maximización o minimización de un objetivo puede perjudicar la optimización de otro u otros objetivos, y entrar en conflicto. Consecuentemente una calendarización debe poder ser evaluada para diferentes objetivos, como *Makespan*, *Meanflowtime*, *Mean Tardiness*, *Tardiness*. La consideración simultánea de estos objetivos es un problema de optimización multiobjetivo. Pero al igual que para un simple objetivo el problema de *flowshop* multiobjetivo es un problema tipo NP-completo.

Año	Autor	Contribución
1972	Heck y Roberts	Criterios secundarios
1975	Emmons	Flow-Shop /tardiness
1980	Van Wassenhove y Gelders	Cost/tardines
1982	Van Wassenhove y Baker Dileepan	Frente de Pareto
1988	Sen y Dileepan	Multicriterio, una máquina

Tabla 6 Antecedentes de Calendarización Multicriterio

2.4.1 Métodos de solución tradicionales para el Problema Multiobjetivo de Calendarización con Transferencia cero

2.4.1.1 Método del Objetivo Pesado

Múltiples objetivos (minimización de: *Makespan*, *Mean Flowtime*, *Mean Tardiness*) son combinados en un sub-objetivo fijo con un factor de peso en una simple función objetivo Z como

$$Z = \sum_{i=1}^{N} w_i f_i$$

Los factores de peso $\{w_i\}$ en Z son números fraccionarios $(0 \le w_i \le 1)$ y la suma de todos estos números es igual a 1. Claramente el vector de peso $(w_1, w_2, w_3, ..., w_N)$ controla la optimalidad de la solución. Esto plantea un problema de decisión por sí mismo.

Matemáticamente una solución obtenida con iguales pesos para todos los objetivos puede ofrecer un menor conflicto entre objetivos, pero puesto que, en el mundo real muchas veces esto no ocurre, ya que cuando se formula Z puede haber

funciones objetivo con cierta prioridad. La desventaja de usar esta técnica es que puede haber un valor relativo de mayor prioridad que controle a los demás. Esta técnica no garantiza la optimalidad de Pareto.

2.4.1.2 Método distancia de funciones

En este método hace la conversión de múltiples objetivos en un sólo objetivo, usando un vector de nivel de demanda ($\bar{y}$) el cual es definido como el vector objetivo. La función de un objetivo se forma desde los objetivos individuales tomados de la siguiente manera:

$$Z=\left[\sum |f_i - y_i|^r\right]^{1/r}, 1 \le r \le \infty$$

Usualmente una métrica euclidiana r=2 con $\bar{y}$ siendo mayor que el óptimo de los diferentes objetivos. La solución para este problema es dependiente de los valores específicos de $\bar{y}$. El método distancia de funciones es similar al método de peso, la diferencia está, en la especificación de las funciones objetivo. Aunque Z sea minimizada con éxito el método no garantiza la optimalidad de Pareto.

2.4.1.3 La formulación Mini-Max

Esta es una variación del método de la distancia de funciones, ya que intenta minimizar una relativa desviación de cada función objetivo desde su óptimo. Se plantea como sigue:

$$\text{Minimizar } F(x) = \max [Z_j(x)],\ j = 1,2,\ldots,n$$

$Z_j(x)$ es calculada para valores óptimos ($\bar{f}_j$>0) como sigue:

$$Z_j(x) = \frac{f_i(x) - \bar{f}_j}{\bar{f}_i},\ j = 1,2,\ldots n$$

Este método produce las mejores soluciones posibles cuando los objetivos { $f_j(x)$ } tienen igual prioridad para ser optimizados, caso que es muy difícil de encontrar en problemas reales.

2.4.2 Desventajas de los Métodos Clásicos

En cada uno de los métodos mencionados, diferentes objetivos son combinados para formar un objetivo usando los conocimientos del problema. La optimización de un sólo objetivo puede ser una aproximación cercana al óptimo de Pareto, pero el resultado es una solución de un *simple punto*. Además los métodos clásicos son muy difíciles de analizar pues se requiere el conocimiento de la prioridad individual de los objetivos (vector).

El principal inconveniente y lo más crítico es que se debe tener de manera explicita cada uno de las prioridades de cada función objetivo (disponibles) con la finalidad de facilitar el análisis del resultado de una sola función objetivo.

2.5 Algoritmos Genéticos en Flowshop Multiobjetivo

2.5.1 Murata, Ishibuchi y **Tanaka** (1996) proponen un algoritmo genético, en donde minimizan Makespan, total Tardiness y Flowtime. Utilizan un esquema de suma de pesos en donde los objetivos individuales forman un número, para luego usar un simple algoritmo genético. Aplican elitismo (preservando los mejores valores solución para cada objetivo).

2.5.2 NSGA (Nondominated Sorting Genetic Algorithm) NSGA usa la asignación de rango a los individuos de la población para luego hacer selección en base al rango, mantiene diversidad en la población, codifica el cromosoma como la secuencia de los *n* trabajos, aplica la cruza de un punto y muta intercambiando trabajos adyacentes. La reproducción se da con los miembros no dominados usando el rango asignado a los individuos. NSGA no preserva las mejores soluciones de una generación a la siguiente. NSGA emplea dos estrategias (1) usa la no dominancia de Pareto para buscar los mejores individuos y (2) usa una función de adaptación que permite obtener las más soluciones no dominadas como sea posible.

2.5.3 ENGA (Elitist Nondominated Genetic Algorithm) básicamente trabaja igual que NSGA la diferencia está en el proceso de selección. Usa para la representación de cromosomas la secuencia de los trabajos en las máquinas, cada inicio de secuencia es una permutación aleatoria de enteros de 1 a *n* sin repetición. Minimiza *Makespan* de la secuencia, tiempo promedio de flujo (*Mean Flowtime*), tiempo promedio de retardos (*Mean Tardiness*). La selección y la reproducción se basan en la no dominancia, usa cruza de un punto, la mutación la realiza intercambiando trabajos adyacentes. Ordenamiento: 1) Le asigna a todos los miembros de la población una bandera con valor 0 significando que estos miembros son clasificados. 2) cada individuo es comparado con otros individuos de la población en base a la no dominancia, si algún miembro es no dominado por algún otro individuo en la población se le asigna una bandera con valor 2 (significa no dominado) en otro caso al individuo se le asigna el valor de 1.

El procedimiento es repetido permitiendo la clasificación de los miembros de la población. Cuando la población es clasificada por completo es ordenada en base al número de frente (usando el método de la burbuja). Usa como criterio de paro el 50% de la población ordenada, constituyendo los miembros de la siguiente generación

2.5.4 Algoritmo Genético con el Método de partición de Pareto (Takanori Tagami y Tohru Kawabe) (1998) Proponen un algoritmo genético utilizando el método de partición de Pareto. El propósito del método es generar un conjunto de óptimos de Pareto que están distribuidos en la vecindad del espacio solución, el método controla la convergencia de las soluciones, esto se logra por que el espacio objetivo es dividido en regiones con el propósito de localizar a las soluciones no dominadas en cada región. Entonces este método asigna a todos los individuos no dominados un grado de adaptación diferente, sobre la base de la distribución en regiones pre-especificadas (ver figura 8).

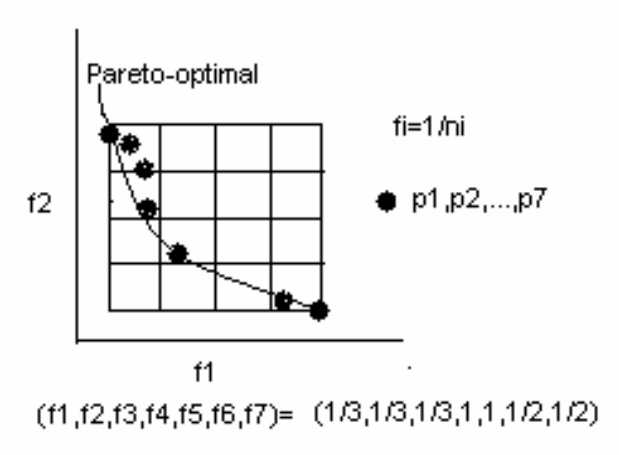

Figura 8 Método de partición de Pareto

3 Algoritmo Propuesto

El objetivo principal de este capitulo es presentar un algoritmo genético para darle solución al problema Multiobjetivo de calendarización de trabajos con transferencia cero, a partir del algoritmo genético aportado por Takanori Tagami y Tohru Kawabe (1998) [7] basado en el método de partición de Pareto, citado en el capitulo anterior. La idea general del algoritmo es generar una segunda población a partir de la población inicial mediante la selección, cruza y mutación de mejores individuos para después unir estas dos poblaciones y hacer una selección de individuos integrado la nueva población y reiniciar el proceso, se continúa así hasta un cierto número de generaciones. La función de adaptación es la que aplica el método de partición de Pareto que usa el concepto de de Rango Multiobjetivo aportado por Fonseca y Fleming [15] con la idea de localizar en el espacio objetivo a las soluciones no dominadas.

Se hace uso de diferentes operadores genéticos para comparar y seleccionar aquellos operadores que arrojen mejores resultados. La selección de individuos para la cruza no se hace de la forma tradicional, se propone una selección mediante la creación de un subconjunto elite, este subconjunto se integra tomando a los mejores individuos de la población en donde la selección para la cruza se hace de manera aleatoria hasta devolver el mismo número de individuos de la población inicial. El tamaño de la población es fijo en todas las generaciones, el criterio de paro esta dado para un número determinado de generaciones.

El algoritmo que aquí se propone toma únicamente del original la función de adaptación (método de partición de Pareto).

El problema consiste en minimizar 2 objetivos Makespan y Mean Flowtime

Minimiza $F(x) = (f_1(x), f_2(x))$

$$f_1(x) = C_{\max} \qquad \text{(Makespan)}$$

$$f_2(x) = \frac{1}{JOBS} \sum_{j=1}^{JOBS} f_j \qquad \text{(Mean Flowtime)}$$

Donde

$C_{\max} = \max_{i \le j \le n} \{ f_j \}$ Makespan y $\sum_{j=1}^{JOBS} f_j$ (Flow Time)

3.1 Modelo Conceptual del algoritmo

El comportamiento general del algoritmo consiste en la combinación de dos poblaciones donde la segunda población es generada a partir de la población inicial (1 ver figura 9) mediante la aplicación de los operadores genéticos (2) para después combinar ambas poblaciones (3) y seleccionar a los mejores individuos de la población resultante (4), este proceso continúa hasta un determinado número de generaciones. El hecho de obtener una combinación de las poblaciones permite

mayor exploración en el espacio objetivo. Conceptualmente el algoritmo se comporta de la siguiente manera

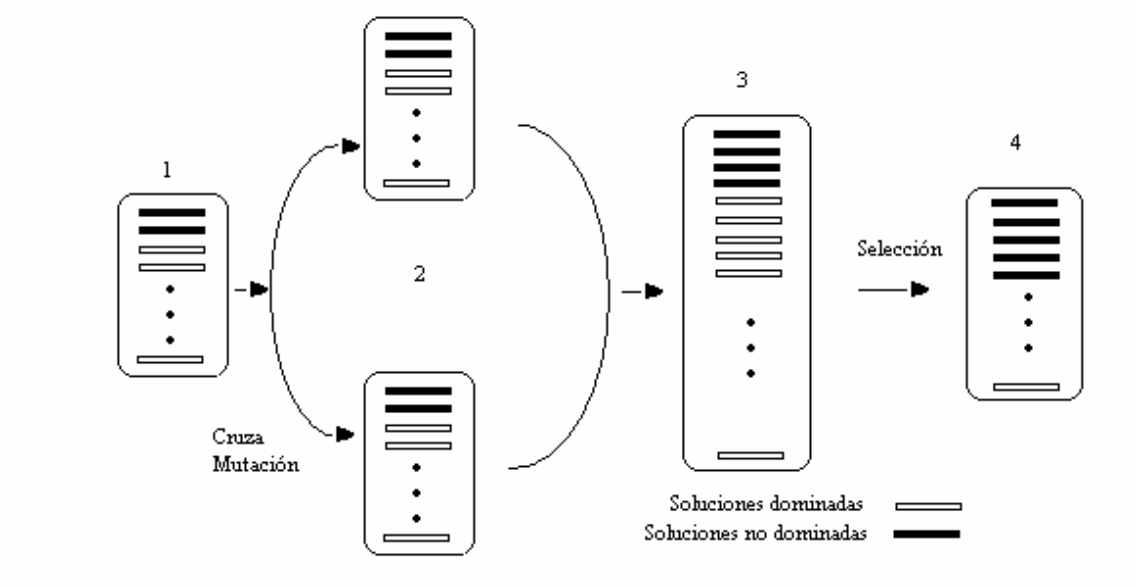

Figura 9 Modelo Conceptual del algoritmo propuesto

3.1.1 Método de partición de Pareto.

La idea central del algoritmo es utilizar el concepto de la no dominancia de Pareto para localizar a los individuos en la frontera de Pareto usando una rejilla, donde cada función objetivo tiene cierto número de regiones y el objetivo es colocar las soluciones a lo largo del frente de Pareto dentro de la rejilla.

El método consiste en que el espacio objetivo es dividido en regiones, los puntos extremos superiores de cada función objetivo corresponden a las mejores soluciones para cada función objetivo. El grado de adaptación de un individuo es definido como $f_i = 1/n_i$, los valores de n_i denotan el número de soluciones no dominadas en la región donde se encuentra el individuo i.

Cabe mencionar que existen algoritmos que también trabajan con un método parecido, en donde, dividen el espacio solución en una especie de cubos que forman una rejilla donde el objetivo es alojar a los individuos no dominados de una manera uniforme y distribuida a lo largo del frente de Pareto, éstos algoritmos no son aplicados para el problema que nos ocupa. Tales algoritmos son PESA (Pareto Enveloped-based Selection Algorithm Corne, Knowles y Oates, 2000) [11], PESA II [12], PAES (Pareto Archived Evolution Strategy, Knowles y Corne, 1999) [10]. Citados en el capitulo 2, éstos y el algoritmo que estamos presentando usan el concepto de Rango Multiobjetivo.

3.2 Formulación del algoritmo Genético

Se presenta el comportamiento general del algoritmo y se dan los detalles de los elementos que lo componen, tales como: función de evaluación, operadores genéticos y matriz de retardos. Se explican los conceptos usados y como se aplican en el algoritmo.

Algoritmo propuesto

Paso 1 Generar población inicial aleatoriamente $P(t)$ de M individuos, generación $t = 0$.

Paso 2 Calcular f_i (Fitness) de cada individuo de la población acorde al Rango Multiobjetivo.

Paso 3 Generar una nueva población $P'(t)$ a partir de $P(t)$ usando el operador de cruza.

Paso 4 Aplicar el operador de mutación a la nueva población $P'(t)$.

Paso 5 Unir las poblaciones $P(t)$ y $P'(t)$.

Paso 6 Calcular Fitness a $P(t)$ y $P'(t)$.

Paso 7 Seleccionar a los M mejores individuos en base al Fitness.

Paso 8 Si la condición de paro es verdadera parar y regresar los mejores individuos encontrados, de lo contrario $t = t+1$ ir al paso 2.

3.3 Características del algoritmo

3.3.1 Codificación

Se usa un cromosoma no binario, en donde cada cromosoma es representado por una secuencia de trabajos, es decir, si consideramos n trabajos cada individuo p_i es definido como una secuencia de enteros cuyos elementos se encuentran en el rango $[1,\ldots,n]$.

Este tipo de cromosoma fue usado por Tsujimura y Kubota [9] donde toman la permutación de los trabajos como el esquema de representación de los cromosomas para resolver el problema de Flowshop. Así el cromosoma [1,5,3,2,4] implica que la secuencia de los trabajos es T_1, T_5, T_3, T_2, T_4. Queda claro que no puede haber alelos iguales pues no seria una cadena válida para Flowshop.

La población es generada de manera aleatoria como un conjunto de cadenas de enteros donde cada cadena es una posible calendarización de los trabajos.

3.3.2 Función de evaluación (Fitness)

La función de evaluación f_i de un individuo p_i es calculada en dos pasos, primero se calcula el Rango Multiobjetivo para cada individuo de la población para después calcular el Fitness correspondiente.

a) Rango Multiobjetivo (Multiobjetive Ranking)

Se usa la idea de Fonseca y Fleming [15] quienes proponen una técnica donde la jerarquía de un individuo está dada por el número de individuos que lo dominan en la

población, tomando esto en cuenta los individuos no dominados tendrán la jerarquía 1, y se desciende proporcionalmente conforme éstos sean dominados por más individuos. La posición en la jerarquía o rango está dada por

$$(p_i, t) = 1 + x \qquad (*)$$

Donde:

p_i i-ésimo individuo en la generación t

x Número de individuos que dominan al individuo p_i en la generación t

Aquí se toma esta idea y el rango de un individuo p_i queda establecido como:

$$r_i = 1 + x$$

A partir de la definición 3 dada en el capitulo 2 sección 2.2, la dominancia de Pareto nos permite calcular el Rango Multiobjetivo para cada individuo de la población. Suponga que se tienen los vectores a, b, c, d y e, en la gráfica se muestra la dominancia entre estos vectores.

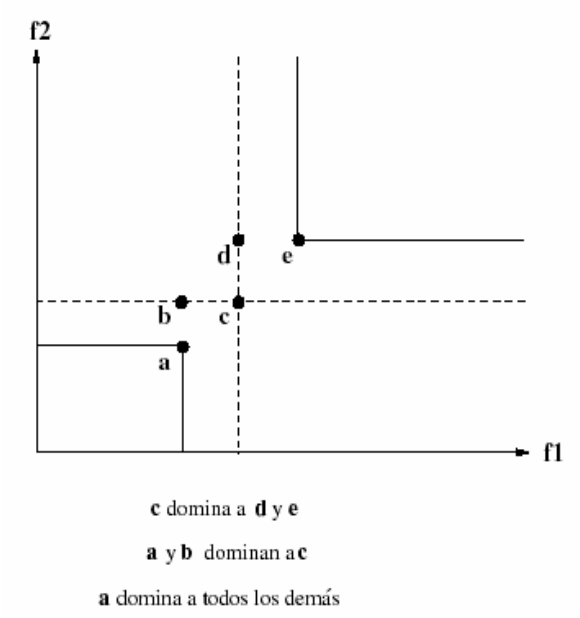

Figura 10 Ejemplo de Dominancia de Pareto

El rango de los vectores según (*) es $r_a = 1 + 0 = 1$; $r_b = 1 + 1 = 2$; $r_c = 1 + 2 = 3$; $r_d = 1 + 3 = 4$; $r_e = 1 + 4 = 5$.

b) Función de evaluación

La función de evaluación (Fitness) f_i de cada individuo p_i es definido como:

$$f_i = (f_{\max} - f_s * (r_i - 1))$$

$$f_s = \frac{1}{M - 1}(f_{\max} - f_{\min})$$

Donde:

$f_{\max}$ máxima calificación (10)
$f_{\min}$ mínima calificación (1)
M tamaño de la población
r_i Rango Multiobjetivo del individuo p_i

3.3.3 Operadores genéticos

3.3.3.1 Selección

La selección de individuos para la reproducción se lleva a cabo mediante la creación de un subconjunto élite. El algoritmo selecciona de la población actual aquellos individuos con mayor calificación ($f_i \geq 8$) para después elegir aleatoriamente de este subconjunto aquellos que han de cruzarse, los individuos seleccionados pueden tener diferente f_i. En este tipo de selección no se necesita el parámetro de probabilidad de cruza, pues los elementos del subconjunto elite se cruzan entre sí hasta devolver una población de igual tamaño que la original.

3.3.3.2 Cruza

Hacemos notar que la cruza debe llevarse a cabo con un método que permita generar cadenas hijas válidas ya que, siendo el cromosoma no binario se puede incurrir a la repetición de trabajos en una misma cadena generando así, cadenas inválidas.

Se implementaron dos tipos de Cruza: Cruza de un punto con reparación de cromosoma y cruza por intercambio de subsecuencias (SXX).

Cruza de Un Punto con Reparación de Cromosoma

Consiste en seleccionar un punto de manera aleatoria dentro del cromosoma de cada padre y a partir de éste se intercambian los materiales genéticos para dar origen a nuevos individuos. Como se nota, este método de cruza al aplicarlo directamente por el tipo de cromosoma que se tiene generaría cadenas hijas inválidas, para garantizar que no suceda esto se usa un método de reparación. El esquema de cromosoma reparado es implementado por Jayaram [16] como sigue:

1. Seleccionar de manera aleatoria un punto de corte (cp) dentro de los cromosomas padre
2. Copiar las cadenas padre en las cadenas hijas hasta el punto de corte
3. En el hijo 1 las posiciones faltantes se llenan con los elementos de la cadena padre 2 que no aparecen en la cadena hija 1, en el mismo orden que aparecen en la cadena padre 2
4. El hijo 2 las posiciones faltantes se llenan con los elementos de la cadena padre 1 que no aparecen en la cadena hija 2, en el mismo orden que aparecen en la cadena padre 1

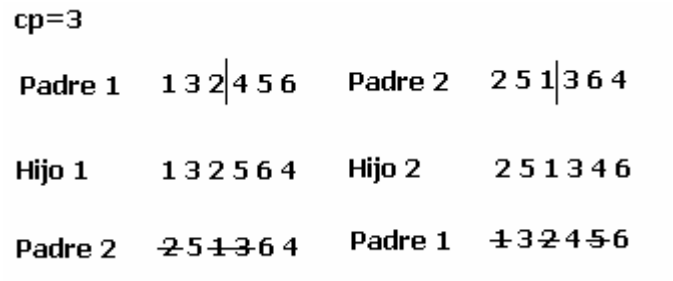

Figura 11 Ejemplo de Cruza de Un Punto con Reparación de Cromosoma

Intercambio de Subsecuencias (SXX)

El método de intercambio de subsecuencias consiste en dadas dos cadenas padre que representan secuencias de trabajos, las subsecuencias son intercambiables sí contienen el mismo conjunto de trabajos, es decir, buscar subcadenas de elementos iguales y hacer el intercambio de dichas cadenas para generar dos cadenas hijas válidas como se muestra en la figura 12.

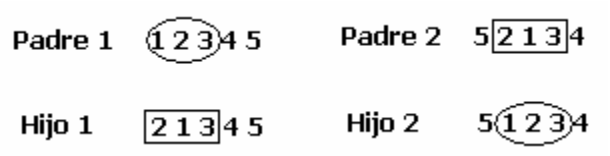

Figura 12 Cruza por Intercambio de Subsecuencias

3.3.3.3 Mutación

Ya que la codificación del cromosoma es no binario el operador de mutación se lleva a cabo por intercambio de posiciones, y se procede como sigue: elegir aleatoriamente dos alelos de la cadena que representa el cromosoma e intercambiar posición entre ellos.

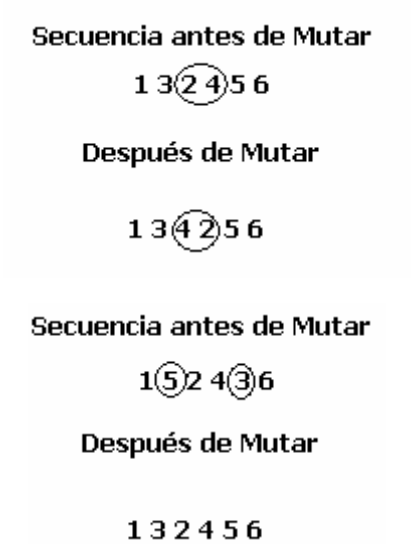

Figura 13 Ejemplos de Mutación

3.4 Matriz de retardos

En el problema de calendarización de trabajos con transferencia cero para calcular Makespan y total Flowtime se debe tomar en cuenta los retardos que se generan al término de un trabajo y el inicio de otro en una máquina, de la contribución de Andreas Fink y Stefan Voß [4] se extrae la ecuación que nos permite calcular los retardos asociados a un determinado problema. Los retardos pueden calcularse con

$$d_{ij} = \max_{1 \le j \le m} \left\{ \sum_{h=1}^{j} t_{ih} - \sum_{h=2}^{j} t_{k,h-1} \right\}$$

Ejemplo: Consideremos la calendarización de trabajos con transferencia cero con cinco trabajos y cuatro máquinas, donde la secuencia óptima es {3,2,5,4,1} y los tiempos de proceso de los trabajos en las máquinas se muestra en el recuadro(Tabla 7). Calcular Makespan y Flowtime

Trabajos	Máquinas 1	2	3	4	Tiempo total
1	15	24	11	18	68
2	22	5	19	13	59
3	16	20	7	16	59
4	12	25	16	17	70
5	15	15	18	25	73

Tala 7 Tiempos de proceso t_{ij}

Cálculo de Retardos. Dada la secuencia {3,2,5,4,1} calcular los retardos d_{32}, d_{25}, d_{45}, d_{41}:

$$d_{32} = \max_{1 \le j \le 4} \left\{ \sum_{h=1}^{j} t_{3h} - \sum_{h=2}^{j} t_{2,h-1} \right\} = 16$$

$$d_{25} = \max_{1 \le j \le 4} \left\{ \sum_{h=1}^{j} t_{2h} - \sum_{h=2}^{j} t_{5,h-1} \right\} = 22$$

$$d_{54} = \max_{1 \le j \le 4} \left\{ \sum_{h=1}^{j} t_{5h} - \sum_{h=2}^{j} t_{4,h-1} \right\} = 20$$

$$d_{41} = \max_{1 \le j \le 4} \left\{ \sum_{h=1}^{j} t_{4h} - \sum_{h=2}^{k} t_{1,h-1} \right\} = 22$$

Calcular Makespan y Flowtime: Para obtener los valores de Makespan y Flowtime se deben sumar los retardos correspondientes más el tiempo de duración del trabajo en cuestión, como se muestra en la tabla 8.

Secuencia	Retardos	Duración por trabajo	Tiempo final de trabajo
3	0	59	59
2	16	59	75
5	38	73	111
4	58	70	128
1	80	68	148 Makespan
			521 Flowtime

Tabla 8 Cálculo de Makespan y Flowtime

3.5 Desarrollo del Algoritmo Genético

En la sección anterior se dio una idea general del proceso que se sigue para la aplicación del algoritmo, cabe mencionar que lo importante del algoritmo es el uso del concepto de Rango Multiobjetivo, pues es lo que permite identificar a los mejores individuos. El Makespan y Mean Flowtime son los valores que indican que tan buena es una secuencia o permutación de los trabajos (dado que el problema es de minimización se deben encontrar los valores mínimos de Makespan y Mean Flowtime).

Para desarrollar el algoritmo la idea central es, dada una población calcular Makespan y Mean Flowtime (funciones objetivo) a cada individuo de la población usando la matriz de retardos para después calcular la función de evaluación (Fitness) y así, aplicar el método de partición de Pareto que permite colocar a los mejores individuos en el frente de Pareto en el espacio objetivo, ver figura 14.

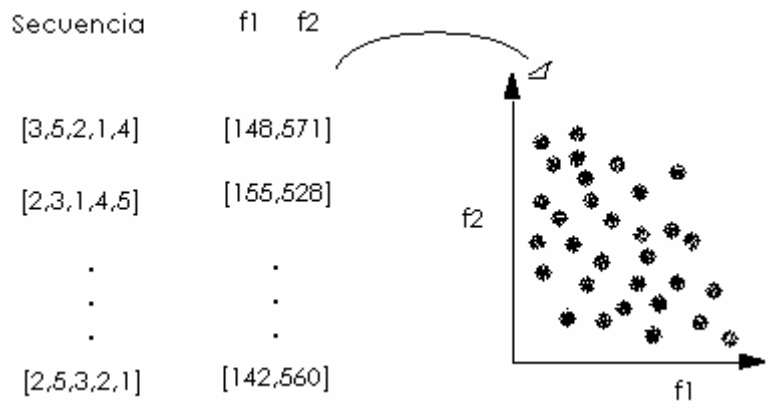

Figura 14 Ejemplo de desarrollo del algoritmo

3.5.1 Diseño

El diseño está basado en el modelo conceptual (figura 9) presentado en la sección 3.1, donde se observa el funcionamiento del algoritmo. Se debe hacer uso de espacio en memoria para guardar las poblaciones que han de unirse para formar la población final que contendrá aquellos individuos que interesan.

Se presenta el Diagrama de flujo y el pseudocódigo del algoritmo. En todos los módulos o procedimientos se tiene que:

n Número de trabajos que son calendarizados

m Número de máquinas

t_{ij} Tiempo de proceso para el i-ésimo trabajo en la j-ésima máquina

T_{ij} Matriz de tiempos t_{ij}

D_{ij} Matriz de retardos d_{ij}

p_i i-ésimo individuo de la población

r_i Rango multiobjetivo del i-ésimo individuo

ft_j Duración del trabajo j desde el inicio en la primera máquina hasta la terminación en la última máquina

f_i Calificación asociada al i-ésimo individuo por la función de evaluación

M Tamaño de la población (p_s)

$F = \sum_{j=1}^{n} ft_j$ Flowtime

$MF = \frac{1}{JOBS} \sum_{j=1}^{n} f_j$ Mean Flowtime

$C_{\max} = \max_{i \le j \le n} \{ f_j \}$ Makespan

$P(t)$ Población en la generación t

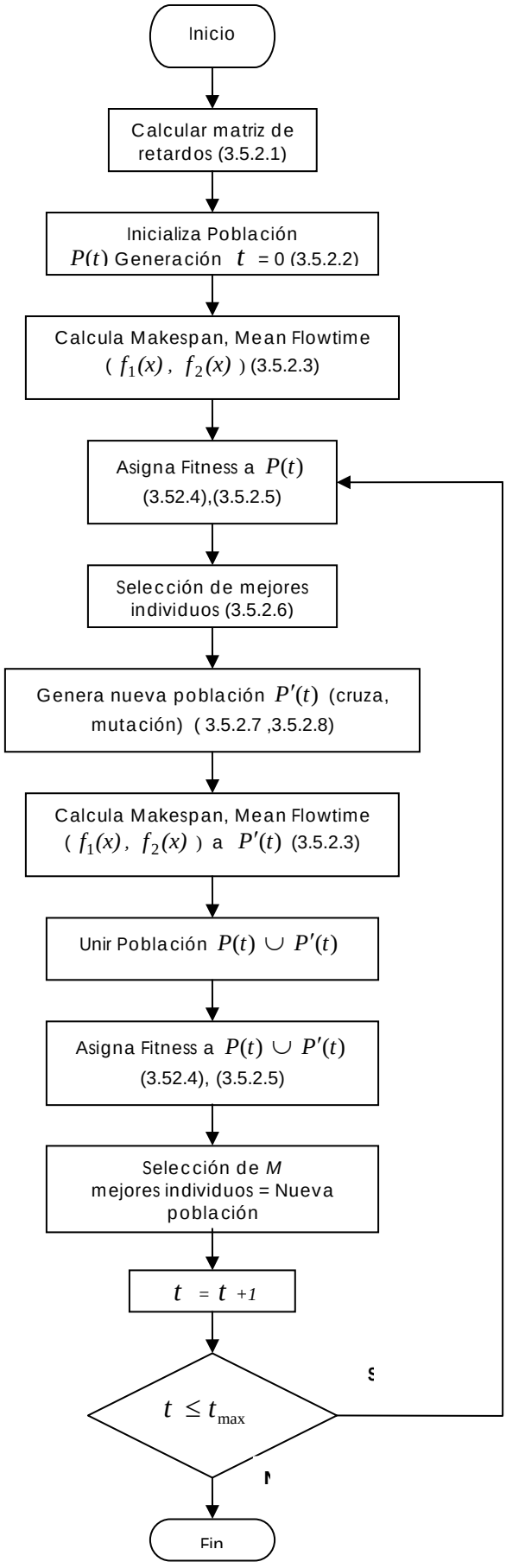

Figura 15 Diagrama de flujo

Pseudocódigo del algoritmo Genético

inicio
Calcular Matriz de retardos
Inicializar $P(t)$ *de tamaño* M , t *=0*
Calcular $f_1(x)$, $f_2(x)$
Mientras $t \leq$ *Número de generaciones* $t_{\max}$ *hacer*
Evalúa población $P(t)$
Seleccionar p_i *cuyo* $f_i \geq$ *8.0 para cruza*
Generar población $P'(t)$ *Aplicando el operador de cruce y mutación a* $P(t)$
Calcular $f_1(x)$, $f_2(x)$ *a* $P'(t)$
$PP(t) = P(t) \cup P'(t)$
Evalúa $PP(t)$
Seleccionar a los Mejores individuos
Integrar la nueva generación reemplazando la anterior, $t = t + 1$
fin mientras
fin

3.5.2 Módulos

Se presentan los módulos que conforman el sistema, en cada módulo se da una breve explicación y su pseudocódigo correspondiente.

3.5.2.1 Retardos

Se encarga de calcular la matriz de retardos que permite obtener Makespan y Flowtime. Para encontrar los retardos generados en una secuencia se debe contar con los tiempos de proceso de los trabajos en las máquinas matriz de tiempos (T_{ij}), así como el número de máquinas (m) y número de trabajos (n).

Procedimiento retardos
inicio
Para $i = 1,...,n$ *hacer*
Suma=0
Para $j = 1,...,m$ *hacer*
Suma=suma+ t_{ij}
a_{ij} = *suma*
$b_{ij+1} = a_{ij}$
fin
fin
Para $k = 1,...,n$ *hacer*
Para $i = 1,...,n$ *hacer*
Para $j = 1,...,m$ *hacer*
$v_j = a_{kj} - b_{ij}$
fin

```
max = v_1
    para j = 1,...,m hacer
        si max < v_i entonces
            max= v_i
        finsi
    fin
    d_ki = max
  fin
fin
```

3.5.2.2 Genera Población

La población inicial es generada aleatoriamente, como secuencias o permutaciones de enteros entre $i = 1,...,n$.

El procedimiento no permite la repetición de elementos en la secuencia pues sería un individuo inválido, cada vez que se genera un número aleatorio para insertarlo a la secuencia, se verifica que no éste presente en la secuencia para insertarlo, sí el número no está presente se inserta de lo contrario, genera otro número lo verifica, así sucesivamente hasta terminar la secuencia.

```
Procedimiento genera población
Inicio
  Contador =1
  Mientras k < tamaño Población
     j =1;
     mientras j < n
        bandera=0
        x = random(1,n)
        para i = 1,...,n
            si x = v_i
                bandera=bandera+1
            finsi
        fin para
        si bandera=0
              v_j = x
              j = j + 1
        finsi
     fin mientras
     secuencia_k = v
     k = k + 1
  fin mientras
fin
```

3.5.2.3 Makespan y Mean Flowtime

Calcula $C_{\max} = \max_{i \le j \le n} \{f_j\}$ y $MF = \frac{1}{JOBS} \sum_{j=1}^{n} f_j$ de las secuencias generadas. Es necesario contar con matriz de retardos (D_{ij}) y matriz de tiempos (T_{ij}).

El procedimiento general del módulo es: dada una secuencia de trabajos obtener los retardos asociados (Ver ejemplo 1), sumar los retardos a la duración del trabajo (f_j) correspondiente en la secuencia para luego obtener el Mean Flowtime.

Procedimiento makespan, flowtime
Inicio
Retardo=0
Para $i = 1,...,n$ *hacer*
$suma = 0$
Para $j = 1,...,m$ *hacer*
$suma_i = suma + t_{ij}$
fin
z=retardo + $suma_i$
$v_i = z$
retardo=retardo+ $d_{i,i+1}$
fin
mk=z
ft=0
Para $i = 1,...,n$ *hacer*
ft = ft+ v_i
fin
fin

3.5.2.4 Ranking

Este procedimiento está basado en rango multiobjetivo y asigna a cada vector de Makespan y Flowtime un rango o jerarquía según la dominancia de Pareto como se vio en sección 3.3.2. Sólo requiere los vectores que han de compararse $[mk_{ij}, ft_{i,j+1}]$ y $[mk_{i+1}, ft_{i+1,j+1}]$.

Procedimiento ranking
inicio
$r_1 = 0$
Para $j = 1,...,n$ *hacer*
para $k = 1,...,n-j$
$x = mk_{j1} - mk_{j+k,1}$
$y = ft_{j2} - ft_{j+k,2}$
Si $x \ge 0 \ \wedge \ y \ge 0$
$r_j = r_j + 1$
fin
else
si $x \le 0 \ \wedge \ y \le 0$

$$r_{j+k} = r_{j+k} + 1$$

finelse
fin
fin
fin

3.5.2.5 Función de adaptación (Fitness)

Se encarga de asignar una calificación según la posición del individuo en el espacio objetivo haciendo uso del módulo anterior para obtener el rango de cada individuo y colocarlo en la malla generada en el espacio objetivo, r_i contiene el rango del i -ésimo individuo.

Procedimiento fitness
inicio

$f_{\min} = 1$

$f_{mzx} = 10$

$$f_s = \frac{1}{M-1}(f_{\max} - f_{\min})$$

Para $i = 1,...,M$ *hacer*

$$f_i = (f_{\max} - f_s * (r_i - 1))$$

fin
fin

3.5.2.6 Selección

Este proceso genera un subconjunto elite con aquellos individuos para los cuales $f_i \geq 8$, en este subconjunto se guarda la secuencia (v_i) con su respectivos valores objetivos ([mk_i, ft_j])

Procedimiento Selección
inicio
Para $i = 1,...,M$ *hacer*
Si $f_i \geq 8$
guarda $v_i \in \{Elite\}$
finsi
fin
fin

3.5.2.7 Cruza

Realiza el intercambio de información genética con los individuos de mejor calificación y son cruzados por una selección aleatoria de los padres. Se utiliza la

cruza de un punto con reparación de cromosoma en donde se toma un punto de corte para después llevar a cabo el intercambio, como en la sección 3.3.3 figura 11.

Procedimiento Cruza
Inicio
pc=x
cont=1
mientras (cont < M)
Para $i = 1,..., pc$ *hacer*
h1=p1
h2=p2
fin
p=1
para $i = 1,..., n$ *hacer*
k=0
para $j = 1,...n$ *hacer*
si $h1_j = p2_i$
k=k+1
finsi
fin
si k = 0
$h1_{pc+p} = p2_i$
p=p+1
finsi
fin
p=1
para $i = 1,..., n$ *hacer*
k=0
para $j = 1,...n$ *hacer*
si $h2_j = p1_i$
k=k+1
finsi
finpara
si k = 0
$h2_{pc+p} = p1_i$
p=p+1
finsi
finpara
fin
fin

3.5.2.8 Muta

Este proceso realiza la mutación una vez que se ha hecho la cruza, intercambiando las posiciones de dos elementos de la secuencia elegidos aletaroriamente.

Procedimiento muta
inicio
Para $i = 1,..., M$ *hacer*
x *= random(1,n)*
y = random(1,n)
temp= v_x
$v_x = v_y$

$v_y = temp$

fin

fin

3.6 Implementación Computacional

3.6.1 Lenguaje de programación

El algoritmo está implementado en lenguaje de programación C (gcc Linux/3.2.2) bajo plataforma Linux versión 5.1. El programa trabaja con problemas que van de 15 trabajos hasta 25 y de 10 a 15 máquinas.

3.6.2 Especificaciones del sistema

El programa se llama Genético, necesita un archivo que contenga la matriz de tiempos en el subdirectorio que se corra, con el nombre de tiempo.dat

3.6.3 Datos de entrada

El sistema pide como datos de entrada el número de trabajos, número de máquinas y número de generaciones. Los valores que tomará como correctos son:
Número de trabajos de 15 a 25
Número de máquinas de 10 a 15
Número de generaciones de 50 a 1000

4 Pruebas y Conclusiones

4.1 Pruebas

Las pruebas se realizaron con más de 20 problemas de diferentes fuentes (http://www.msic.ac.uk/info.html, [17], [4] notando un buen comportamiento en el algoritmo), aunque aquí solamente se reportan dos ejemplos documentados y dos ejemplos de 30 y 49 trabajos con 15 máquinas para hacer un comparativo con los resultados obtenidos con el algoritmo propuesto. En la siguiente sección se hace un análisis de resultados evaluando los tiempos de ejecución.

Para validar el desempeño del algoritmo se requiere de otro (u otros) método que evalúen los problemas con los mismos datos para tener igualdad de condiciones en ambos métodos y tener un parámetro de comparación de la calidad de las soluciones, en nuestro caso no se tiene implementado otro método de solución a problemas multiobjetivo, por lo tanto se hará la comparación con el método Bertilissi [4] (método monobjetivo citado en el capitulo 1 sección 1.5) que ofrece la permutación que minimiza Mean Flowtime, el otro objetivo (Makespan) se calcula a partir del Flowtime obtenido. En cada ejemplo se muestra una tabla comparativa de resultados. Se muestran detalladamente dos problemas con 20 trabajos, 10 máquinas y 25 trabajos, 15 máquinas procesados en el programa. Mostramos la matriz de tiempos, matriz de retardos, los valores mínimos y máximos encontrados según el tamaño de la población y generación, así como las mejores secuencias encontrados. Se muestra también una gráfica donde se aprecia los valores que se acumulan hacia el frente de Pareto para cada problema. Las matrices de tiempos y problemas de mayor tamaño se tomaron de [9].

4.1.1 Resolver para 20 trabajos y 10 máquinas con una matriz de tiempos mostrada en la siguiente tabla

TRABAJOS	MÁQUINAS										TIEMPO
	1	2	3	4	5	6	7	8	9	10	TOTAL
1	74	72	54	57	52	60	4	8	40	8	429
2	99	77	58	50	31	67	19	96	93	29	619
3	15	10	85	2	92	53	60	63	11	94	485
4	63	18	44	30	80	94	63	28	50	55	525
5	94	42	20	92	73	62	45	86	76	11	601
6	16	15	97	30	7	31	82	10	28	13	329
7	51	57	19	87	81	17	8	27	93	72	512
8	18	46	17	12	54	54	90	52	69	82	494
9	77	29	52	40	83	53	44	49	87	60	574
10	19	95	57	94	93	19	36	14	82	74	583
11	25	28	72	89	5	87	15	87	14	20	442
12	92	98	56	35	64	15	95	22	67	61	605
13	47	80	88	77	77	60	63	66	8	10	576
14	38	84	99	21	13	73	0	26	68	99	521
15	77	65	38	88	95	9	13	13	42	55	495
16	22	81	30	45	63	94	44	78	98	57	612
17	6	3	43	96	51	39	79	28	50	27	422
18	32	28	61	62	18	76	92	50	5	26	450
19	28	97	87	10	79	35	55	72	98	32	593
20	16	73	45	90	76	86	75	38	58	49	606

Matriz de Tiempos

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1		74	197	154	101	204	130	222	111	127	150	74	99	114	77	131	205	168	118	130
2	209		228	170	128	303	243	247	163	163	197	113	129	236	192	154	245	195	148	160
3	64	15		49	15	169	60	116	15	15	63	15	15	63	45	26	101	61	15	21
4	104	63	135		63	209	123	191	63	63	110	63	63	116	85	88	154	128	63	65
5	209	113	210	170		302	243	247	163	163	193	113	94	236	192	143	245	182	127	120
6	16	16	103	47	16		31	77	22	16	75	16	16	16	16	25	119	68	16	39
7	91	51	183	140	58	196		202	97	89	93	51	61	90	72	117	162	112	80	92
8	73	18	103	24	18	178	65		18	45	72	18	18	72	54	42	99	70	36	48
9	153	77	183	126	77	258	167	188		87	152	77	77	160	134	103	179	150	78	90
10	162	74	246	203	110	267	162	265	160		161	77	67	161	143	180	213	176	136	134
11	41	25	107	89	58	141	88	159	56	43		25	25	80	34	81	162	105	25	80
12	184	92	233	190	125	289	197	254	147	171	193		143	190	165	168	237	186	162	174
13	185	157	257	214	136	296	238	291	180	145	237	132		230	173	191	254	228	167	158
14	100	45	196	140	86	205	115	181	115	107	168	38	94		81	118	212	161	96	132
15	106	79	251	208	115	207	149	270	165	123	153	82	95	121		185	216	180	141	139
16	191	58	221	142	41	296	208	212	128	128	190	78	56	201	172		217	188	92	87
17	14	6	87	44	6	121	48	116	6	6	24	6	6	41	6	21		40	6	14
18	46	32	112	58	32	173	99	168	41	41	98	32	32	91	34	50	131		33	49
19	180	64	202	146	76	277	214	218	134	134	171	84	85	207	163	123	216	169		123
20	185	98	215	151	78	290	210	260	130	130	184	101	42	203	166	145	223	185	125	

Matriz de retardos

La siguiente tabla muestra los valores máximos y mínimos encontrados por el programa para Makespan y Mean Flowtime, como se observa en la tabla (9) los mejores valores se obtienen para poblaciones y generaciones de mayor tamaño en un número de 10 a 15 corridas. Los mejores valores obtenidos son para p_s=100 y $t_{\max}$=500.

p_s	$t_{\max}$	Valor mínimo encontrado		Valor máximo encontrado	
		$C_{\max}$	MF	$C_{\max}$	MF
50	100	2250	1303	2304	1293
	300	2262	1311	2363	1312
	400	2184	1296	2350	1294
	500	2146	1242	2255	1320
70	100	2189	1297	2250	1291
	300	2155	1269	2489	1349
	400	2167	1278	2349	1334
	500	2118	1270	2163	1281
100	100	2152	1283	2435	1329
	300	2141	1287	2252	1267
	400	2150	1308	2223	1322
	500	2137	1267	2180	1282
	500	2118	1270	2198	1296
	500	2123	1256	2184	1296
	500	2124	1253	2190	1310
	500	2128	1258	2185	1269
	500	2129	1256	2187	1276

Tabla 9 Valor mínimo y máximo según p_s y $t_{\max}$

En la tabla 10 mostramos las mejores secuencias encontradas por el programa con sus correspondientes valores de Makespan y Mean Flowtime obtenidas en 20 corridas.

Secuencia	**Makespan**	**Mean Flowtime**
8 4 9 19 16 5 20 13 10 12 2 18 15 14 7 11 3 17 1 6	2118	1270
8 4 3 18 16 5 20 13 10 12 9 19 2 17 1 7 11 15 14 6	2155	1269
6 8 4 9 16 5 20 13 12 18 19 2 17 11 3 10 7 14 15 1	2144	1260
6 14 12 2 16 5 20 13 17 18 3 9 19 8 4 10 7 11 15 1	2131	1274
6 14 12 2 16 5 20 13 17 11 3 18 9 19 8 4 10 7 15 1	2142	1263
6 14 12 2 8 17 11 3 18 4 9 19 16 5 20 13 10 7 15 1	2123	1256
6 14 12 2 18 17 11 3 8 4 9 19 16 5 20 13 10 7 15 1	2124	1253
6 8 4 3 18 15 14 7 10 12 9 17 11 19 2 1 6 5 20 13 1	2143	1267
6 8 4 16 5 20 13 10 12 18 3 9 19 2 17 11 15 14 7 1	2128	1260
6 8 4 9 16 5 20 13 10 12 2 19 18 3 17 11 15 14 7 1	2128	1258
6 8 4 9 16 5 20 13 10 12 2 18 19 3 17 11 15 14 7 1	2129	1256
6 14 7 11 3 8 4 20 13 10 12 17 9 19 2 16 5 18 15 1	2131	1270
6 8 4 19 2 16 5 21 13 17 9 3 10 12 18 15 14 7 11 1	2139	1280
6 8 4 9 16 19 5 20 13 10 2 12 3 17 18 15 14 7 11 1	2136	1283
6 14 7 11 3 8 4 20 13 10 12 18 9 19 2 16 5 17 15 1	2150	1268
6 14 12 2 16 5 20 13 17 11 3 18 9 19 8 4 10 7 15 1	2142	1263

Tabla 10 Mejores Secuencias Encontradas

Secuencia Optima Según Bertolissi	Makespan	Mean Flowtime
6 8 17 3 18 11 4 7 14 1 9 16 15 19 20 10 5 2 12 13 **Mejor Secuencia Encontrada con el Algoritmo Propuesto**	2659	1480
8 4 9 19 16 5 20 13 10 12 2 18 15 14 7 11 3 17 1 6	2118	1270

Tabla 11Tabla comparativa de resultados

A continuación mostramos la gráfica que nos permite observar como se acumulan los vectores no dominados hacia el frente de Pareto. La gráfica demuestra la calidad de las soluciones encontradas por el programa.

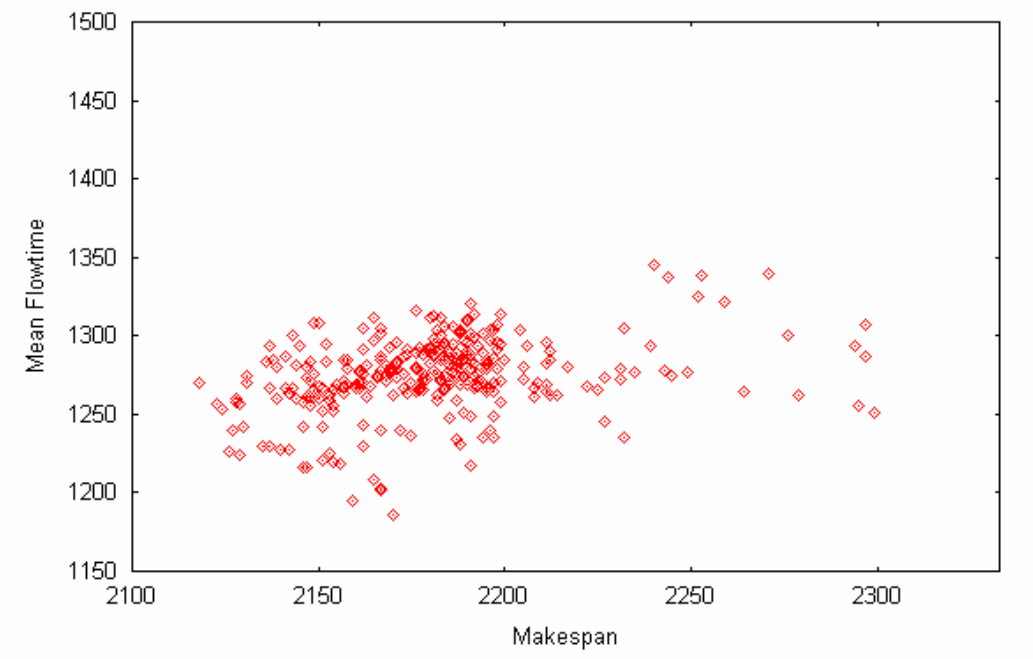

Figura 16

4.1.2 Resolver para 25 trabajos y 15 máquinas con una matriz de tiempos mostrada en la siguiente tabla.

TRABAJOS	MÁQUINAS															TIEMPO
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	TOTAL
1	74	72	54	57	52	60	4	8	40	8	85	45	74	67	48	748
2	99	77	58	50	31	67	19	96	93	29	27	6	85	22	48	807
3	15	10	85	2	92	53	60	63	11	94	44	71	19	99	94	812
4	63	18	44	30	80	94	63	28	50	55	78	83	8	68	65	827
5	94	42	20	92	73	62	45	86	76	11	80	53	29	3	70	836
6	16	15	97	30	7	31	82	10	28	13	63	55	24	26	49	546
7	51	57	19	87	81	17	8	27	93	72	1	19	30	80	86	728
8	18	46	17	12	54	54	90	52	69	82	47	96	90	14	12	753
9	77	29	52	40	83	53	44	49	87	60	2	88	26	18	30	738
10	19	95	57	94	93	19	36	14	82	74	94	7	90	40	39	853
11	25	28	72	89	5	87	15	87	14	20	24	91	93	41	36	727
12	92	98	56	35	64	15	95	22	67	61	12	98	73	6	10	804
13	47	80	88	77	77	60	63	66	8	10	63	74	90	1	56	860
14	38	84	99	21	13	73	0	26	68	99	9	72	42	43	27	714
15	77	65	38	88	95	9	13	13	42	55	51	36	27	78	12	699
16	22	81	30	45	63	94	44	78	98	57	26	85	61	82	71	937
17	6	3	43	96	51	39	79	28	50	27	49	30	73	53	85	712
18	32	28	61	62	18	76	92	50	5	26	1	53	33	79	17	633
19	28	97	87	10	79	35	55	72	98	32	2	58	87	86	12	838
20	16	73	45	90	76	86	75	38	58	49	61	90	23	92	24	896
21	47	8	30	52	53	96	26	76	18	65	32	18	35	71	36	663
22	65	90	73	28	34	58	20	10	46	94	38	34	0	17	72	679
23	6	62	42	42	95	26	90	83	82	54	47	44	80	22	2	777
24	39	12	28	15	77	22	46	17	19	18	83	5	28	55	96	560
25	4	75	57	1	24	46	99	43	17	22	36	22	12	22	38	518

Matriz de tiempos

En la siguiente tabla se muestran los valores máximos y mínimos de Makespan y Mean Flowtime encontrados por el programa. Los mejores valores se obtuvieron con p_s = 100 y $t_{\max}$ =500, como en el ejemplo anterior.

p_s	$t_{\max}$	Valor mínimo encontrado		Valor máximo encontrado	
		$C_{\max}$	MF	$C_{\max}$	MF
50	100	3194	1850	3864	2041
	300	3193	1916	3679	1986
	400	3169	1878	3858	2075
	500	3152	1878	3797	2050
70	100	3195	1853	3715	2123
	300	3177	1922	3631	1968
	400	3198	1977	3709	1975
	500	3113	1857	3440	1929
100	100	3136	1859	3444	1874
	300	3100	1851	3534	1925
	400	3118	1828	3659	1940
	500	3057	1781	3705	1957
	500	3053	1806	3168	1825
	500	3031	1810	3118	1828
	500	3022	1812	3163	1888
	500	3007	1812	3127	1876

Tabla 12 Valor mínimo y máximo según p_s y $t_{\max}$

En la siguiente tabla se muestran las mejores secuencias halladas después de 15 corridas.

Secuencia	Makespan	Mean Flowtime
24 6 14 1 7 18 10 12 9 20 16 8 4 3 19 13 11 2 23 5 21 17 15 22 25	3098	1865
24 6 14 1 7 18 10 12 9 20 16 8 4 13 19 11 3 23 5 2 21 17 15 22 25	3058	1860
24 6 7 11 14 1 15 22 21 17 9 19 5 8 23 13 3 4 10 12 16 20 2 18 25	3096	1805
24 6 25 7 1 14 15 22 12 9 21 17 11 3 23 8 4 10 19 16 20 13 5 2 18	3098	1763
24 6 25 1 21 17 11 19 5 3 8 4 13 9 2 23 16 20 10 12 14 7 15 18 22	3070	1765
24 21 3 8 4 5 2 17 15 18 1 14 6 7 11 19 23 16 20 13 10 12 9 22 25	3097	1848
24 21 17 14 7 18 15 6 25 1 11 19 5 3 8 4 2 23 16 20 13 10 12 9 22	3071	1765
24 6 18 21 17 15 22 14 1 7 11 19 2 8 4 23 5 3 16 20 10 12 13 9 25	3080	1794
24 6 21 17 11 19 23 5 14 15 22 7 18 1 2 3 8 4 16 20 13 10 12 9 25	3064	1793
24 6 21 17 11 3 23 12 14 15 22 7 18 1 2 1 9 8 4 16 20 10 13 5 9 25	3096	1783
24 6 21 17 11 3 23 5 14 15 22 7 18 1 2 19 8 4 16 20 13 10 12 9 25	3071	1790
24 21 17 11 1 14 22 7 18 9 4 3 19 8 23 5 2 16 20 13 10 12 15 6 25	3053	1806
24 21 17 11 3 4 16 20 10 13 12 9 19 8 23 5 2 1 14 22 7 18 15 6 25	3007	1812
24 21 17 11 3 4 16 20 10 13 12 9 19 8 23 5 2 7 18 22 1 14 15 6 25	3022	1812
24 21 17 11 3 4 16 30 10 13 12 9 19 8 23 5 2 7 18 15 1 14 22 6 25	3031	1810
24 18 21 17 2 4 16 20 13 9 7 11 19 8 23 5 3 10 12 14 22 1 15 6 25	3084	1894
24 21 18 1 7 11 19 23 16 20 3 8 13 5 2 17 9 4 10 12 14 22 15 6 25	3070	1855
24 21 18 1 7 11 19 23 16 8 3 20 13 5 2 17 9 4 10 12 14 22 15 6 25	3074	1858

Tabla 13 Mejores Secuencias Encontradas

Secuencia Optima Según Bertolissi	Makespan	Mean Flowtime
24 6 25 17 21 18 11 3 7 8 14 22 1 23 4 9 15 19 5 10 16 20 2 13 12	3723	2125.36
Mejor Secuencia Encontrada con el Algoritmo Propuesto		
24 21 17 11 3 4 16 20 10 13 12 9 19 8 23 5 2 1 14 22 7 18 15 6 25	3007	1812

Tabla 14 Tabla comparativa de resultados

En la siguiente gráfica se muestra como los vectores no dominados se acumulan hacia el frente de Pareto, mientras el resto de la población queda acumulada por encima de éstos.

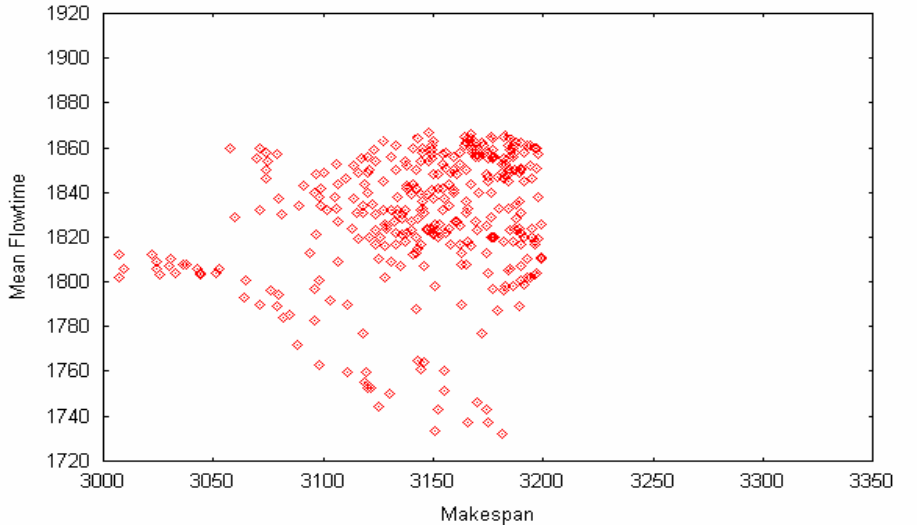

Figura 17

4.2 Análisis de Resultados

Para validar la calidad de las soluciones se probaron problemas de mayor tamaño (ejemplos disponibles en Bagchi [9]) de 30, 49 trabajos con 15 máquinas donde el problema para 30 trabajos 15 máquinas solo se dispone de Makespan y para el ejemplo de 49 trabajos 15 máquinas minimiza Makespan, Mean Flowtime y Tardiness. En la tabla se muestra los valores promedio obtenidos en Makespan y MeanFlowtime

Resultado Obtenido con	*n*	*m*	Makespan	Mean Flowtime
Bagchi	49	15	3795	2337.63
Algoritmo Propuesto			5006	2742
Bagchi	30	15	2504	No disponible
Algoritmo Propuesto			3739	2128
Bertolissi	25	15	3723	2125.36
Algoritmo Propuesto			3027	1810
Bertolissi	20	10	2659	1480
Algoritmo Propuesto			2123	1261

Tabla 15 Resumen Comparativo de Resultados

Por los resultados obtenidos se concluye que el algoritmo que aquí se presenta no es recomendable para un número mayor de trabajos pues conforme el número de trabajos aumenta la convergencia es menor. Se probaron también ejemplos de menor tamaño tomados de diferentes fuentes observando buenos resultados. Por lo que el algoritmo se recomienda para un número de trabajos y máquinas menor o igual a 25.

Para hacer un análisis sobre el comportamiento del programa en tiempo según el tamaño de la población y el número de generaciones para los ejemplos expuestos con 25 trabajos, 15 máquinas y 20 trabajos, 10 máquinas se muestran los tiempos de ejecución del programa observados para ambos problemas. Los tiempos de ejecución (CPU) fueron obtenidos con las funciones de reloj de lenguaje C.

En las tablas 16 y 17 se muestra el tiempo promedio de ejecución en segundos para distintos tamaños de población (p_s) con generaciones de 100 a 1000, para ambos ejemplos.

En la tabla 18 se puede ver que a un número mayor de trabajos y máquinas el tiempo promedio de ejecución aumenta según el tamaño de la población y el número de generaciones, aunque el tiempo es menor para p_s = 50 y $t_{\max}$ = 100 en las pruebas realizadas se muestra que los mejores resultados se obtienen con un tamaño de p_s = 100 y $t_{\max}$ = 500 con un tiempo de ejecución igual para ambos ejemplos.

La gráfica muestra el comportamiento del programa en tiempo según el tamaño de población y el tiempo promedio de ejecución para distintos números de generaciones. Como se puede ver el tiempo promedio de ejecución se incrementa para problemas de mayor tamaño.

p_s	t_{max} 100	200	300	400	500	600	700	800	900	1000	Tiempo Promedio de ejecución
50	0.02	0.02	0.02	0.02	0.02	0.02	0.03	0.03	0.03	0.03	0.024
70	0.03	0.03	0.03	0.03	0.03	0.04	0.04	0.04	0.04	0.04	0.035
100	0.04	0.04	0.04	0.05	0.05	0.06	0.06	0.06	0.06	0.06	0.052

Tabla 16 Tiempo de ejecución en C para n =25 y m =15 (una corrida)

p_s	t_{max} 100	200	300	400	500	600	700	800	900	1000	Tiempo Promedio de ejecución
50	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.018
70	0.02	0.02	0.02	0.03	0.03	0.03	0.03	0.03	0.04	0.04	0.029
100	0.04	0.04	0.04	0.04	0.05	0.05	0.05	0.05	0.05	0.05	0.046

Tabla 17 Tiempo de ejecución en C para n =20 y m =10 (una corrida)

n	m	p_s =50 p_m =0.1 t_{max} [100,1000]	p_s =70 p_m =0.1 t_{max} [100,1000]	p_s =100 p_m =0.1 t_{max} [100,1000]
20	10	0.018	0.029	0.04
25	15	0.024	0.035	0.052

Tabla 18 Tiempo promedio en segundos de ejecución (una corridas)

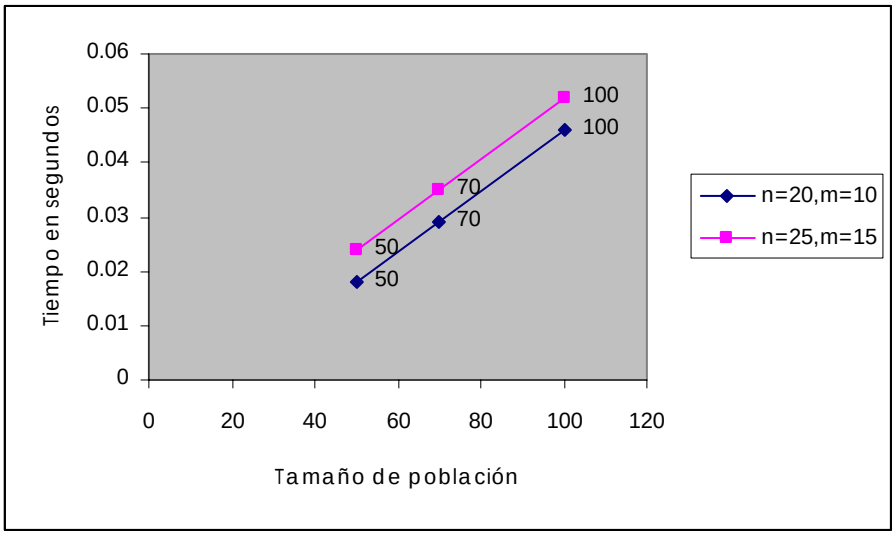

Figura 18

4.3 Conclusiones

Los buenos resultados de un algoritmo genético dependen en gran parte de la buena elección de los operadores genéticos, pues son estos los que permiten o limitan la exploración del espacio solución del problema en especial cuando la codificación de la población es no binaria como en este caso, es por eso que se considera una aportación importante de este trabajo el algoritmo en general, el tipo de selección que se propone y la comparación de operadores de cruza y mutación. A continuación damos una conclusión de resultados de cada operador y la parametrización que se utiliza.

4.3.1 Selección

El tipo de selección implementada da como resultado diversidad en la población, pues no reproduce hijos iguales a los padres, además se nota que en una población un aproximado del 65%, 70% son individuos aptos para la cruza.

4.3.2 Cruza

Se implementaron dos métodos de cruza: Intercambio de subsecuencias (SXX) y Cruza de un punto con reparación de cromosoma.

El método de Intercambio de subsecuencias conlleva un trabajo computacional mayor por el número de comparaciones que se llevan a cabo para encontrar las subsecuencias con elementos iguales, cuando las cadenas contienen mayor número de elementos (20 a 25) las cadenas hijas no presentan un gran cambio entre ellas y sus padres correspondientes por lo que limita la exploración, ya que, genera cadenas muy parecidas y en algunos casos iguales. Esto se debe a que es difícil encontrar subsecuencias de más de 5 caracteres para que la cadena cambie significativamente y si se toman subsecuencias muy pequeñas las cadenas hijas son distintas pero muy parecidas a sus padres y casi se tendría una mutación, por lo que se usa el método Cruza de un punto con reparación de cromosoma.

Se observo que para más de 10 trabajos el método Cruza de un punto con reparación de cromosoma además de ofrecer una buena exploración lleva un trabajo computacional menor garantizando mejores resultados. En la figura 19 se muestra la aplicación de ambos métodos.

4.3.3 Mutación

Se hicieron pruebas intercambiando trabajos adyacentes y aleatorios observando que dan mejores resultados las cadenas obtenidas de mutar por intercambio de trabajos elegidos aleatoriamente.

4.3.4 Inversa

Se implemento el operador de inversa en donde se invierte todo la cadena. No se obtuvieron buenos resultados por lo que no se incluye en el programa final.

4.3.5 Parametrización

Los parámetros considerados en este algoritmo son: tamaño de la población (p_s), probabilidad de mutación (p_m), número de regiones particionadas en el método de partición de Pareto $(p_s)^2$ y número máximo de generaciones ($t_{\max}$). Las pruebas se hicieron con los valores que se presentan en la tabla 19.

Parámetro	Valor
p_s	50, 70, 100
$(p_s)^2$	$(50)^2$, $(70)^2$, $(100)^2$
p_m	0.1
$t_{\max}$	100, 300,400,500

Tabla 19

Los mejores resultados se logran con una población de 100 individuos, probabilidad mutación igual a 0.1 en 500 generaciones como se muestra en la tabla 20.

Parámetro	Mejor Valor
p_s	100
$(p_s)^2$	$(100)^2$
p_m	0.1
$t_{\max}$	500

Tabla 20

Referencias

[1] S. Kobayashi, I. Ono, and M. Yamamura, "*An Efficient Genetic Algorithm for Job Shop Scheduling Problems*", Procedings of the Sixth Interntional Conference on Genetic Algorithms University of Pittsburgh, 1995, p. 506-522.

[2] C. A. Coello, D. A. Veldhuizen, and G.B. Lamont, *Evolutionary Algorithms for Solving Multiobjective Problems*. Kluwer Academic Publishers, USA, 2002.

[3] D. Greiner, "*A summarized overview of evolutionary multiobjetive algorithms and new approaches*", CEANI, USA, 2000.

[4] A. Fink, and V.B.Stefan, "*Solving the Continuos Flow-Shop Scheduling Problem by Metaheuristics*". Technical University of Braunschweig, Gemany, 2001.

[5] B. Pérez, y M.A. Osorio, "*Análisis Comparativo de Heurísticas para Problemas de Calendarización de Trabajos con Transferencia Cero*" Memorias del Primer Congreso Mexicano de Computación Evolutiva, CIMAT, 2003, p. 43-54.

[6] R. Ruiz, y C. Morato, "*Evaluación de Heurísticas para el problema del taller de flujo*". Reporte Técnico, Departamento de estadística e Investigación Operativa Aplicadas y Calidad. Universidad Politécnica de Valencia, España, 2003.

[7] T. Tagami & T. Kawabe, "*Genetic Algorithm with a Pareto Partitioning Method for Multiobjetive Flowshop Scheduling*", Technical Report, Kagawa Junior College, Japan; University of Tsukuba, Japan, 1998.

[8] R. Z. Ríos, y J. F. Bard, "*Heurísticas para Secuenciamiento de Tareas en Líneas de Flujo*". Reporte Técnico, Universidad Autónoma de Nuevo León, 1999.

[9] T. Bagchi, *Multiobjetive Scheduling by Genetic Algorithms*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1999.

[10] J. D. Knowles and D. W. Corne "*Approximating the Nondominated Front using the Pareto Archived Evolution Strategy "* . Evolutionary Computation, 8(2) pp149-172 Massachusetts Institute of Technology, 2000.
http://iridia.ulb.ac.be/~jknowles/pubs.html

[11] D. W. Corne, J. D. Knowles, M. J. Oates, "*The Pareto-Envelope based Selection Algorithm for Multiobjetive Optimization"*. Parallel Problem Solving from Nature-PPSN VI, Springer Lecture Notes in Computer Science, Springer, Berlin, 2000.

[12] D. W. Corne, N. R. Jerram, J. D. Knowles, M. J. Oates "*PESA-11: Region-based Selection in Evolutionary Multiobjetive Optimization".* Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), Morgan Kaufmann Publishers.

[13] A. Kuri, J. Galaviz, *Algoritmos Genéticos*, Instituto Politécnico Nacional, Universidad Nacional Autónoma de México, Fondo de Cultura Económica, México, 2002

[14] H. Ishibuchi, T. Yoshida, T. Murata, *"Balance between Genetic Search and Local Search in Memetic Algorithms for Multiobjective Permutation Flowshop Scheduling "*. Deparment of Industrial Engineering, Osaka Prefecture University, Japan; Deparment of Informatics, Faculty of Informatics, Kansai University, Japan.

[15] C. M. Fonseca and P. J. Fleming. *Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization*. In Stephanie Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms, pages 416-432, San Mateo, California , 1993. University of Illinois at Urbana –Champaign, Morgan Kauffman Publishers.

[16] k., Jayaram (1998) *Multiobjetive Production Scheduling*, M Tech Thesis, Deparment of industrial and Management Engineering, Indian Institute of Technology Kanpur.

[17] D. L. Bricker, Dept. of Industrial of Iowa, Iowa City, Iowa 52242 e-mail: dbricker@icaen.uiowa.edu

Apéndice

Partes importantes de código

```
retardo(int job, int mach){

 int mt[N][M], mb[N][M], i, j,k,l,m,max,v1[N],suma,x,y,h;

 FILE *fpt;


 fpt=fopen("tiempo.dat","r");


 for(x=1;x<job;x++){
                for(y=1;y<mach;y++){
       fscanf(fpt,"%d",&h);
                        mt[x][y]=h;
                }
 }

for(i=1;i<job;i++){//N=job
                suma=0;
                for(j=1;j<mach;j++){
                        suma=suma+mt[i][j];
                        ma[i][j]=suma;
                    mb[i][j+1]=ma[i][j];
                 }
 }
 for(y=1;y<job;y++){
       vma[y]=ma[y][mach-1];
                }
 for(k=1;k<jobt;k++){
                for(i=1;i<job;i++){
                    for(j=1;j<gato;j++){
                     v1[j]=ma[k][j]-mb[i][j];
                  }
                    max=v1[1];
                    for(m=1;m<mach;m++){
                        if(max<v1[m]){
                            max=v1[m];
                        }
                    }
                    d[k][i]=max;
                }
 }
 fclose(fpt);
 return(0);
}

makft(int w, int uno)
{
   int reta, x,y,z,ft,a, vres[N],i,l,k,g,m;
   int vs[N],meanft;


   for(k=1;k<w;k++){
                for(l=1;l<uno;l++){
                  vs[l]=ms[k][l];
                }
                reta=0;
                  for(i=1;i<uno;i++){
                      x=vs[i];
                      y=vs[i+1];
                     z=reta+vma[x];
                      vres[i]=z;
                      reta=reta+d[x][y];
                }
               mft[k][1]=vres[uno-1];
                ft=0;
                  for(a=1;a<uno;a++){
                        ft=ft+vres[a];
                  }
                   meanft=ft/uno-1;
                mft[k][2]=meanft;
                 }
    return(0);
```

```
}

rankingm(int l){

  int j,k,x,y,a;

    for(a=0;a<l;a++){
                        marca[a]=0;
              }
              for(j=1;j<l;j++){
                        for(k=1;k<l-j;k++){
                                  x=(mft[j][1]-mft[j+k][1]);
                                  y=(mft[j][2]-mft[j+k][2]);
                                  if((x>=0 && y>=0)){
                                            marca[j]=marca[j]+1;
                                  }
                                  else{
                                            if( (x<=0 && y<=0)){
                                                      marca[j+k]=marca[j+k]+1;
                                            }
                                  }
                        }
              }
              return(0);
}

fitness(int z){

              int i,t;
              float  fs,fmx,fmin,p,k,x;


              fmx=10.0;
              fmin=1.0;
              x=(z-1.0);
              k=(1.0/x);
              fs=k*(fmx-fmin);
              for(i=1;i<z;i++){
             p=fs*marca[i];
               f[i]=(fmx-p);
              }

              return(0);
}

cruza(int dos){

   int contador, n1,n2, k1,k2,k,l,j,i,a,cp,p;
   int vp1[N], vp2[N],vh1[N], vh2[N];


     srand(time(NULL));

     cp=dos/2;
     contador=1;
     while(contador<S+1){
       n1= (1 + (rand() % S-1));
       n2= (1 + (rand() % S-1));
      if(n1 != n2 && vaf[n1]>0 && vaf[n2]>0){
                                  k1=vaf[n1];
                             k2=vaf[n2];
                                  for(j=1;j<N;j++){
                                   vp1[j]=ms[k1][j];
                                   vp2[j]=ms[k2][j];
                                  }
                           for(a=1;a<cp+1;a++){
                              vh1[a]=vp1[a];
                            vh2[a]=vp2[a];
                             }

          p=1;
          for(i=1;i<dos;i++){
                                            k=0;
                                for(j=1;j<dos;j++){
                                      if(vh1[j]==vp2[i]){
                                                        k=k+1;
                                       }
                                       }
                                       if(k==0){
                                                  vh1[cp+p]=vp2[i];
                                                  p=p+1;
```

```
                                        }
                                }

        p=1;
        for(i=1;i<dos;i++){
                        k=0;
                          for(j=1;j<dos;j++){
                                  if(vp1[i]==vh2[j]){
                                                  k=k+1;
                                   }
                                  }
                                  if(k==0){
                                          vh2[cp+p]=vp1[i];
                                          p=p+1;
                                  }
                                }
        for(l=1;l<dos;l++){
             pp[contador][l]=vh1[l];
           pp[contador+1][l]=vh2[l];
         }

        contador=contador+2;
}
   }

 return(0);
}

genpo(int cero)
{

  int i,j,k,ban,cont,vt[N], num;
  int contador,ii,jj;

  srand(time(NULL));
  contador=1;
  while(contador < S){
              for(i=0;i<N;i++){
                      vt[i]=0;
              }
              cont=1;
              while(cont<cero){
                      ban=0;
                    num=(1+(rand() % (cero-1)));
                      for(j=1;j<cero;j++){
                              if(num==vt[j]){
                                      ban=ban+1;
                              }
                      }
                      if(ban==0){
                              vt[cont]=num;
          cont=cont+1;
                      }
              }
              for(k=1;k<cero;k++){
                      ms[contador][k]=vt[k];
              }
    ++contador;
  }
  return(0);
}
```

AGRADECIMIENTOS

BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

CONSEJO NACIONAL DE CIENCIA Y TECNOLOGÍA

Dra. María Auxilio Osorio Lama

Agradezco a mis revisores:

Dra. Lourdes Sandoval Solís

Dra. Darnes Vilariño Ayala

M.C. Rogelio González Velásquez

Dr. Abraham Sánchez López

Contenido