TESIS DEFENDIDA POR

Rodrigo Aceves Pérez

Y aprobada por el siguiente comité:

Dr. Carlos Alberto Brizuela Rodríguez

Director del Comité

Dr. Gustavo Olague Caballero

Miembro del Comité

M.C. Jorge Torres Rodríguez

Miembro del Comité

Dr. Joaquín Álvarez Gallegos

Miembro del Comité

Dr. Jesús Favela Vara

Jefe del Departamento de Ciencias de la Computación

Dr. Luis Alberto Delgado Argote

Director de Estudios de Posgrado

13 de octubre de 2003

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN
SUPERIOR DE ENSENADA

DIVISIÓN DE FÍSICA APLICADA
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

Estudio de Operadores Genéticos para un Problema de Calendarización Multi-Objetivo

TESIS

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

MAESTRO EN CIENCIAS

Presenta:

Rodrigo Aceves Pérez

Ensenada, Baja California a octubre de 2003.

RESUMEN de la tesis de **Rodrigo Aceves Pérez**, presentada como requisito parcial para obtener el grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACIÓN. Ensenada, B. C. octubre de 2003.

Estudio de Operadores Genéticos para un Problema de Calendarización Multi-Objetivo

Resumen aprobado por:

Dr. Carlos Alberto Brizuela Rodríguez

Director de Tesis

En este trabajo se presenta un análisis experimental de las combinaciones de cuatro operadores de cruzamiento y tres de mutación, a fin de encontrar aquella combinación que genere las mejores soluciones.

Sin embargo, no es fácil decidir entre dos o más conjuntos de soluciones no-dominadas cuál es el mejor. El enfoque aquí propuesto se basa en utilizar las relaciones de dominancia entre los conjuntos para lograr tal decisión. Se propone un criterio de comparación de conjuntos de soluciones no-dominadas. Esta propuesta se hizo debido a que la mayoría de los criterios actuales están basados en la hipótesis de que las funciones objetivo son continuas, y el problema de flujo de tareas multi-objetivo tiene funciones objetivo discretas. Se analizan varios de los criterios actuales y se contrastan con el criterio aquí propuesto.

Los operadores se analizaron sobre tres algoritmos genéticos distintos, para comprobar que la supremacía de la combinación ganadora sea independiente del algoritmo. Además, se estudiaron dos conjuntos de casos de tamaño distinto para evaluar la forma en que el tamaño del problema afecta el comportamiento de los operadores genéticos.

En base a los experimentos se concluye que existe una combinación de operadores de cruzamiento y de mutación que genera el mayor promedio relativo de soluciones no-dominadas. Tal combinación es la de cruzamiento Obx con mutación Insert. Además, se encontró que la mutación Switch es la peor de las mutaciones y que los operadores de cruzamiento Osx y Twopoint se comportan de manera similar.

Para respaldar las conclusiones obtenidas en el presente trabajo, se realizó un análisis estadístico no-paramétrico utilizando la prueba de Kruskal-Wallis y la prueba de Wilcoxon de suma de rangos.

Palabras clave: Algoritmos Genéticos, Multi-objetivo, Frente Pareto, Calendarización, Flujo de Tareas.

ABSTRACT of the thesis presented by **Rodrigo Aceves Pérez**, as a partial requirement to obtain the SCIENCE MASTER degree in COMPUTER SCIENCES. Ensenada, B. C. october 2003.

Genetic Operators Analysis for a Multi-Objective Scheduling Problem

Abstract approved by:

Dr. Carlos Alberto Brizuela Rodríguez

Thesis director

This thesis deals with the experimental analysis of genetic operators for the multi-objective permutation flowshop problem. All combinations of four crossover and three mutation operators are analyzed. The aim of the work is to find which combination produces the best set of solutions (the best non-dominated front).

However, deciding the best among two or more non-dominated fronts is not an easy task. Our approach is based on studying the dominance relationships among the sets to reach such decision. In fact, we propose a criterion for comparing non-dominated sets that overcome the limitations of current criteria which were thought of for continuous objective functions, while the multi-objective flowshop problem has discrete objective functions. Several of these criteria are analyzed and contrasted with our proposed criterion.

The genetic operators were analyzed over three different genetic algorithms, to ensure that the supremacy of the winning combination is algorithm-independent. Two set of instances of different sizes were studied to assert the way problem size affects the behavior of the genetic operators.

We conclude that there is a combination of crossover and mutation operators that outperforms the others. This combination is composed of the Obx crossover and the Insert mutation. Besides, we found that Switch mutation is the worst of all mutations and that Osx and Twopoint crossover operators have similar behavior.

In order to support the conclusions attained, a statistical non-parametric analysis was performed applying the Kruskal-Wallis and the Wilcoxon's sum of ranks tests.

Keywords: Genetic Algorithms, Multiobjective, Pareto front, Scheduling, Flowshop.

A mi familia y a Myrna

Agradecimientos

Agradezco ante todo a mi familia por haberme apoyado en mi decisión de estudiar una maestría lejos de casa. A mi padre por ponerme el ejemplo del estudio y la dedicación a la ciencia, a mi madre por su cariño incondicional y sus recetas de cocina, a mis hermanas y a mi cuñado por su apoyo y a mi sobrina por ser mi sobrina.

También agradezco a Myrna por haber hecho de este tiempo en Ensenada un tiempo valioso. Por compartir los desvelos, las hambrunas, las presiones y todo lo demás que vino con la maestría. Pero sobretodo porque gracias a ella nunca estuve solo estando tan lejos de mi familia.

Y un agradecimiento muy especial a mi asesor Carlos Brizuela, por su paciencia para conmigo; por el enorme tiempo que dedicó a mis escritos y a mis presentaciones; por la manera en que me transmitió sus conocimientos sobre el tema, pero sobre todo por su amistad.

Gracias a Everardo Gutiérrez por sus críticas (siempre constructivas) hacia mi trabajo de tesis. A Domitilo por su ayuda en matemáticas durante los cursos y a mis compañeros de generación por su amistad.

También agradezco a los miembros de mi comité de tesis, Dr. Gustavo Olague, M. en C. Jorge Torres y Dr. Joaquín Álvarez, por sus valiosas aportaciones que sirvieron para mejorar la calidad de este trabajo.

Ensenada, México — Rodrigo Aceves Pérez
13 de octubre de 2003.

Tabla de Contenido

Capítulo Página

Tabla de Contenido (Continuación)

Capítulo Página

Lista de Figuras

Lista de Figuras (Continuación)

Lista de Figuras (Continuación)

Lista de Figuras (Continuación)

Lista de Tablas

Tabla de símbolos

$\mathbf{K}$	Conjunto
n	Escalar
$\mathbf{a}$	Vector
$\mathbf{f}(x)$	Vector de funciones

Capítulo I

Introducción

I.1 Antecedentes y Motivación

Fue en 1913 cuando Henry Ford introdujo la línea de ensamble para producir el famoso modelo 'T'. Tal vez en ese entonces no se imaginó que no sólo los automóviles serían manufacturados en esa forma, también muchos artículos más como: televisores, refrigeradores, computadoras, etc. Algunos de los beneficios que tiene esta línea de ensamble son: reducir costos de producción, reducir tiempos de entrega, incrementar ganancias, etc.

Es posible tener líneas de ensamble que produzcan dos o más artículos utilizando las mismas máquinas (o a las mismas personas). Por ejemplo, producir hornos de microondas y hornos eléctricos; los dos están divididos en procesos similares sólo que cada uno tiene piezas distintas. De hecho, se pueden producir, con los mismos recursos, cosas tan diferentes como taladros, televisores y radios si éstos pasan por los mismos procesos de manufactura. La diferencia está en que cada artículo tiene un tiempo de procesamiento distinto en cada máquina (aunque pueden ser iguales) y estas diferencias hacen que el orden en que son manufacturados altere los tiempos de finalización de los mismos.

El párrafo anterior es una descripción, *grosso modo*, del problema de flujo de tareas.

I.1.1 El Problema

El problema de flujo de tareas consiste en encontrar una calendarización óptima, donde calendarización se refiere a la asignación en el tiempo de recursos limitados a tareas. Es un proceso de toma de decisiones que tiene la meta de optimizar uno o varios objetivos (Pinedo, 1995).

Existen distintos tipos de problema de flujo de tareas. Cada tipo de problema depende de las características de la línea de producción. Uno de ellos es el problema con almacenamiento cero. Este problema se caracteriza porque no hay un medio de almacenamiento entre dos máquinas cualesquiera, lo que ocasiona, entre otras cosas, que cuando una máquina termina su proceso no puede continuar hasta que entregue su producto a la máquina siguiente.

Por el contrario, el problema con almacenamiento infinito: entre dos máquinas cualesquiera hay un medio de almacenamiento tan grande como se requiera. Esto simplifica el problema, ya que cuando una máquina termina su trabajo lo puede dejar en el medio de almacenamiento y continuar con un nuevo trabajo aunque la máquina siguiente aún no termina el suyo.

Otro tipo de problemas surgen cuando no hay un orden único de trabajo (French, 1982). Es decir, un producto puede pasar de la máquina 1 a la máquina 3 y de ahí a la 7, etc. Y otro producto puede empezar por la máquina 3 y después pasar a la máquina 2, etc. Inclusive hay casos en que no todos los productos utilizan todas las máquinas.

También hay un tipo de problemas que tienen máquinas repetidas (Ku *et al.*, 1993). Esto es, de la máquina 1 hay más de una copia, entonces un producto puede pasar a la máquina 1(a) o a la máquina 1(b); y ambas máquinas pueden estar ocupadas al mismo tiempo. Esto se hace cuando hay un proceso en particular que es muy lento, así que para darle mayor movilidad a la línea se añaden máquinas que hacen el mismo proceso.

Otros problemas utilizan máquinas multi-funcionales. Este tipo de máquinas pueden

realizar el proceso de un conjunto de máquinas en la línea de producción. Es decir, pueden hacer el mismo trabajo que la máquina 1, la máquina 2 ó de la máquina 6, 7, 8, etc. Además estas máquinas también pueden estar repetidas. Esta máquinas ayudan a eliminar los cuellos de botella. Sin embargo, el problema de flujo de tareas se complica en este tipo de ambientes porque se debe calendarizar no sólo el orden de las tareas sino también el orden y el funcionamiento de las máquinas multi-funcionales.

En este trabajo se estudia el problema de flujo de tareas con almacenamiento infinito, con una sola secuencia para todos los trabajos, sin máquinas repetidas ni máquinas multi-funcionales.

En los problemas de producción, en general se intenta minimizar el tiempo de finalización total (*makespan*). Este tiempo es, en términos burdos, el tiempo en que se termina la producción total. Es decir, el momento en que termina el último proceso del último artículo a ser producido.

Sin embargo, puede ser de interés vender cada producto una vez que éste sea finalizado. Entonces, interesaría minimizar el tiempo promedio de flujo: el tiempo que transcurre desde que cada artículo está listo para ser fabricado hasta que es terminado.

Otro criterio importante en las líneas de producción es el tiempo promedio de retraso. Este tiempo sólo tiene sentido cuando cada producto tiene un compromiso de tiempo para ser finalizado (fecha límite). Si el producto se termina antes de la fecha límite no hay penalización, pero si se termina después de la fecha límite hay una penalización que es proporcional al tiempo de retraso. Se requiere, por lo tanto, que los tiempos de retraso sean mínimos.

En general, la minimización del *makespan* no implica la minimización del tiempo promedio de flujo, o del tiempo promedio de retraso. De hecho, cuando se tienen dos o más objetivos como estos y es importante que todos se optimicen al mismo tiempo, entonces el problema se vuelve multi-objetivo.

Los problemas multi-objetivo tienen una serie de dificultades *per se*. La más importante es que no hay una solución única, sino un conjunto de soluciones que satisfacen ciertas condiciones (Steuer, 1986). Otra dificultad es que, debido a que se trabaja con conjuntos de soluciones, no es sencillo distinguir entre dos conjuntos cuál es mejor (van Veldhuizen y Lamont, 2000; Knowles y Corne, 2002; Zitzler *et al.*, 2002b), más aún, cuando estos conjuntos son generados en forma estocástica.

I.2 La Metodología

Se sabe que el problema de flujo de tareas, considerando solamente la minimización del *makespan*, pertenece a la clase que se conoce como NP-difícil (NP-hard) (Garey *et al.*, 1976). Los problemas de esta clase son muy difíciles de resolver. Es por esto que se han propuesto varias heurísticas para tratar de encontrar una solución próxima al óptimo del problema. Una de estas heurísticas es la de los algoritmos genéticos.

Los algoritmos genéticos trabajan con una población de individuos, donde cada individuo representa una solución al problema. La conexión entre los algoritmos genéticos y los problemas multi-objetivo es entonces lógica. A diferencia de los métodos tradicionales que sólo obtienen una solución por corrida, los algoritmos genéticos obtienen un conjunto de soluciones en una sola corrida. Además son una herramienta emergente que está obteniendo resultados interesantes en varios problemas, especialmente en problemas mono-objetivo (Ulder *et al.*, 1991; Syswerda, 1991).

Sin embargo, la teoría de los algoritmos genéticos no ha ido a la par con la práctica. Esto significa que los mejores parámetros del algoritmo, así como los operadores que más le favorecen para cada problema en particular, son totalmente desconocidos. En la actualidad, es práctica común tomar un algoritmo y encontrar los parámetros a base de prueba y error. Existen infinitas combinaciones de parámetros para cada algoritmo,

además de que los mejores parámetros varían de problema a problema. Pocos investigadores incluyen además la búsqueda de los operadores genéticos óptimos.

La idea en este trabajo es encontrar, de entre un conjunto de operadores de cruzamiento y mutación, aquellos que son más apropiados para el problema de flujo de tareas multi-objetivo. Este interés surge del trabajo de Brizuela *et al.* (2001), donde se plantea la pregunta ¿existe una combinación de operadores genéticos que sea superior, en términos de las relaciones de dominancia, para el problema de flujo de tareas?. Existen trabajos donde se comparan varios operadores genéticos (Starkweather *et al.*, 1991; Murata *et al.*, 1996), y en particular el trabajo de Murata *et al.* (1996) lo hace para el problema de flujo de tareas, aunque su estudio es realizado para la versión mono-objetivo del problema. Para los problemas multi-objetivo podemos citar los trabajos de Tagami y Kawabe (1999) que crean particiones del frente Pareto; Talbi *et al.* (2001) que realizan un estudio comparativo de algoritmos usados para resolver este problema; Ishibuchi y Murata (1998) que usan un algoritmo genético con una combinación lineal de pesos junto con una estrategia de búsqueda local; e Ishibuchi *et al.* (2003) quienes crean un algoritmo híbrido entre algoritmo genético y búsqueda local.

Una dificultad cuando se diseñan algoritmos para problemas combinatorios multi-objetivo es el no contar con una medida para evaluar la calidad de las soluciones generadas por los distintos algoritmos en términos de las relaciones de dominancia. En la literatura están reportados varios criterios (Hansen y Jaszkiewicz, 1998; Knowles y Corne, 2002; van Veldhuizen y Lamont, 2000) pero no son efectivos cuando se trata de evaluar soluciones para un problema combinatorio, como el problema de flujo de tareas.

I.3 Objetivo Principal

Encontrar, de entre un conjunto de operadores genéticos, la combinación de operadores de cruzamiento y mutación que generen las mejores soluciones para casos específicos del problema de flujo de tareas multi-objetivo.

I.3.1 Objetivos Específicos

1. Proponer casos tipo para el problema de flujo de tareas multi-objetivo.
2. Diseñar experimentos para la comparación de operadores genéticos.
3. Estudiar y entender los algoritmos genéticos que se van a utilizar.
4. Comprender los operadores de cruzamiento y de mutación con los que se va a experimentar.
5. Enumerar y entender los criterios para considerar una solución mejor que otra.

I.4 Organización del trabajo

El trabajo de tesis aquí presentado se encuentra organizado de la siguiente manera:

El capítulo 2 (Problema de Flujo de Tareas Multi-objetivo) presenta la formulación matemática del problema, enfatiza la dificultad que surge al trabajar con funciones multi-objetivo, enumera las relaciones entre vectores que nos sirven para definir lo que es el frente Pareto e introduce los criterios que se analizarán.

El capítulo 3 (Algoritmos Genéticos) muestra los algoritmos genéticos con los que se trabajará, así como la representación de las soluciones que se utilizará. Esta representación a su vez ayuda a entender cómo trabajan los operadores de cruzamiento y de

mutación, los cuales se discuten en el mismo capítulo. Para finalizar se presenta una breve reseña de lo que se ha hecho con los algoritmos genéticos en calendarización.

Los experimentos realizados se presentan en el capítulo 4 (Experimentos y Resultados). Se plantean dos conjuntos de casos del problema de flujo de tareas, la diferencia entre ellos es el tamaño. Para cada problema se muestra cómo son los datos de entrada, qué resultados se generan y se hace un breve análisis de los resultados obtenidos.

Por último, en el capítulo 5 (Conclusiones) se presentan las conclusiones a las que se llegó en este trabajo, así como las posibles líneas de investigación a seguir.

Capítulo II

Problema de Flujo de Tareas Multi-objetivo

Un escenario común donde surge el problema de flujo de tareas es en las líneas de producción en serie. En este tipo de producción se tienen varias máquinas -que realizan diferentes funciones- y ellas deben producir varios artículos. Cada artículo puede tener un tiempo de procesamiento distinto en cada máquina y además necesita pasar por todas las máquinas para poder ser terminado. En el problema de flujo de tareas (versión permutación) el orden de las máquinas está fijo, esto es, la secuencia que siguen los artículos a través de las máquinas es la misma para todos. Cada máquina debe terminar un proceso iniciado antes de pasar a otro, es decir, no se permiten suspensiones. Cuando un artículo entra a la primera máquina debe continuar a la siguiente máquina (cuando ésta lo permita) y luego a la siguiente hasta pasar por todas. El paso de una máquina a otra sólo es posible cuando la máquina 'receptora' está libre y la máquina 'emisora' ha terminado de procesar al artículo. Cada artículo tiene un tiempo de finalización límite, antes del cual debe ser terminado.

El orden en que se inicia la producción de los artículos y la disponibilidad de recursos limitados crea diferentes situaciones: "bloqueo" (blocking), una máquina ha terminado un proceso pero no puede empezar el siguiente (queda bloqueada) porque la máquina siguiente aún está ocupada; "ocio" (idle), una máquina ha terminado su trabajo y lo ha liberado, pero no puede procesar el trabajo que sigue porque la máquina anterior aún no lo libera. Este tipo de situaciones hace que no se aprovechen las máquinas el 100% del tiempo, lo que lleva a retrasos en la entrega, tiempos de finalización muy grandes,

etc.

En general se intenta optimizar (minimizar por regla general) el tiempo de finalización del último artículo (*makespan*). Aunque también se puede minimizar el tiempo promedio en que cada artículo es producido, o el tiempo promedio de retraso en la producción de los artículos con respecto a ciertas fechas límite específicas para cada uno de ellos. De hecho, el problema de flujo de tareas se vuelve multi-objetivo al tratar de minimizar dos o más de estos criterios *al mismo tiempo*. Un ejemplo para comprender mejor este problema se puede ver en el Apéndice A. Una definición formal del problema se presenta a continuación.

II.1 Formulación del Problema

El problema de flujo de tareas multi-objetivo se plantea de la siguiente manera: Sea $\mathbf{K}_n$ el conjunto de los n primeros números naturales, i.e. $\mathbf{K}_n = \{1, 2, \ldots, n\}$. El problema de flujo de tareas versión permutación (permutation flowshop) consiste en un conjunto $\mathbf{K}_n$ de tareas ($n \geq 1$) que deben ser procesadas en un conjunto de máquinas $\mathbf{K}_m$ ($m \geq 1$). Cada tarea $j \in \mathbf{K}_n$ está compuesta de m operaciones. Cada operación O_{kj}, que representa la k-ésima operación de la tarea j, tiene asociado un tiempo de procesamiento p_{kj}. Cuando una máquina ha empezado a procesar una operación debe terminarla (no se permiten suspensiones temporales – *preemption*). Ninguna máquina puede procesar más de una operación al mismo tiempo. Ninguna operación puede ser procesada por más de una máquina al mismo tiempo. A cada tarea j se le asigna un tiempo de disponibilidad r_j (tiempo a partir del cual puede empezar a ser procesado) y una fecha límite d_j (tiempo para el cual debe estar finalizado). Todas las tareas deben seguir la misma ruta a través de las máquinas.

Ahora hace falta establecer una notación para poder describir las restricciones así

como las funciones objetivo a minimizar. El tiempo de inicio de la operación O_{kj} se denota como s_{kj}. Una solución del problema se define como una permutación $\pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$, con $\pi(i) \in \mathbf{K}_n$. Bajo estas condiciones una solución factible debe cumplir:

$$s_{mj} \geq r_j \quad \forall j \in \mathbf{K}_n \tag{1}$$

$$s_{k\pi(j)} + p_{k\pi(j)} \leq s_{(k+1)\pi(j)} \quad \forall k \in \mathbf{K}_{m-\mathbf{1}}, j \in \mathbf{K}_n \tag{2}$$

La ecuación (2) nos dice que el tiempo de inicio de la operación k de la tarea $\pi(j)$ ($s_{k\pi(j)}$) más su tiempo de procesamiento ($p_{k\pi(j)}$) debe ser menor o igual que el tiempo de inicio de la misma tarea $\pi(j)$ en su operación $k+1$ ($s_{(k+1)\pi(j)}$).

Todo par de operaciones, de tareas consecutivas, procesadas por la misma máquina debe satisfacer:

$$s_{k\pi(j)} + p_{k\pi(j)} \leq s_{k\pi(j+1)}$$
$$\text{para toda máquina } k \in \mathbf{K}_m, \text{ y toda tarea } j \in \mathbf{K}_{n-1} \ . \tag{3}$$

La desigualdad (3) significa que el tiempo de inicio de la operación k de la tarea $\pi(j)$ ($s_{k\pi(j)}$) más su tiempo de procesamiento ($p_{k\pi(j)}$) debe ser menor o igual que el tiempo de inicio de la siguiente tarea $\pi(j+1)$ en su operación k ($s_{k\pi(j+1)}$).

Tomaremos la figura 1 para ilustrar las restricciones y las funciones objetivo. La primera restricción no se ilustra porque se supone que todos los tiempos $r_j = 0 \ \ \forall j \in \mathbf{K}_n$, es decir, todas las tareas están listas para ser procesadas a partir de $t = 0$. La segunda restricción se muestra en la figura 2 (izquierda), donde se observa que cada tarea debe terminar la operación en la máquina actual antes de pasar a la siguiente máquina (la siguiente operación). La tercera restricción se muestra en la figura 2 (derecha), donde se muestra que una máquina debe terminar la operación de la tarea actual antes de continuar con la operación de la siguiente tarea.

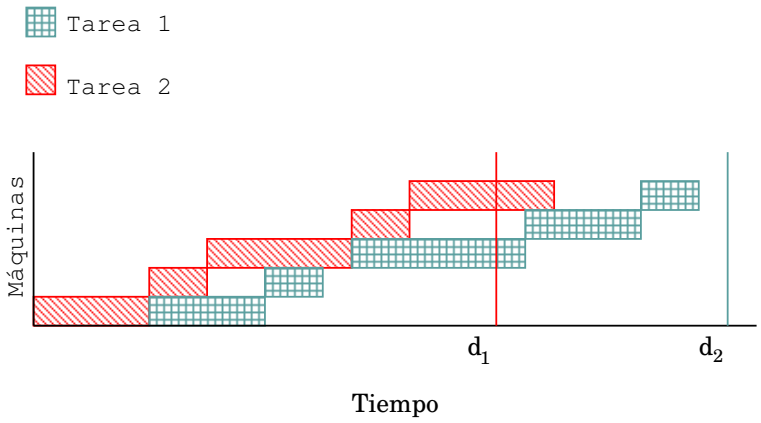

Figura 1: Diagrama de Gantt para un problema de 2 tareas y 5 máquinas

En la figura 3 se ilustra la primera función objetivo que queremos minimizar, que es el *makespan*, i.e. el tiempo de finalización de la última tarea,

$$f_1 = cmax = s_{m\pi(n)} + p_{m\pi(n)} \tag{4}$$

La segunda función objetivo se ilustra en la figura 4 y es el tiempo promedio de flujo (mean flow time). Este es el tiempo promedio que las tareas permanecen en el área de trabajo,

$$f_2 = \overline{fl} = \frac{1}{n}\sum_{j=1}^{n} fl_j, \tag{5}$$

donde $fl_j = \{s_{mj} + p_{mj}\} - r_j$, i.e. el tiempo que la tarea j permanece en el taller después de ser liberada.

Y en la figura 5 se ilustra el tercer objetivo a minimizar, que es el promedio del retardo (mean tardiness)

$$f_3 = \overline{T} = \frac{1}{n}\sum_{j=1}^{n} T_j \tag{6}$$

donde $T_j = \max\{0, L_j\}$, y $L_j = s_{mj} + t_{mj} - d_j$.

Entonces, el problema de flujo de tareas multi-objetivo es:

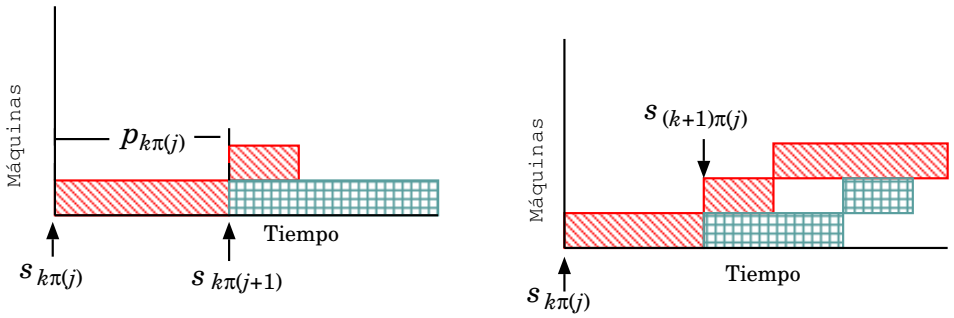

Figura 2: Ilustración de las restricciones para el problema de flujo de tareas. Izquierda: Restricción 2. Derecha: Restricción 3

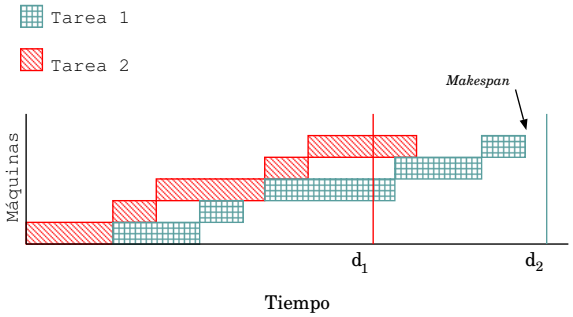

Figura 3: Ilustración del makespan para un problema de 2 tareas y 5 máquinas

$$\text{Minimizar}(f_1, f_2, f_3)$$
$$\text{sujeto a } (1) - (3). \tag{7}$$

Está demostrado que este problema, considerando solamente el *makespan* como función objetivo, es NP-difícil (Garey *et al.*, 1976). Como el problema definido por (7) es al menos tan difícil como el problema del *makespan*, entonces el problema (7) es también NP-difícil. Esto se demuestra en la proposición 1.

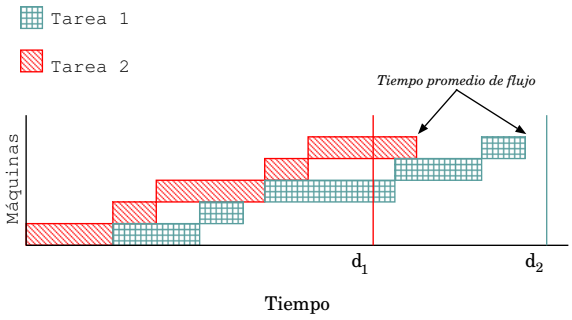

Figura 4: Ilustración del tiempo promedio de flujo para un problema de 2 tareas y 5 máquinas

Proposición 1. *El problema definido por* (7) *es NP-difícil.*

Prueba. Para demostrar que un problema A es NP-difícil, se debe demostrar que algún problema NP-difícil se reduce polinomialmente a A. Entonces, tomamos el problema dado por (4), el cual sabemos que es NP-difícil (Garey *et al.*, 1976) y vemos que se reduce polinomialmente a (7). La reducción polinomial se puede hacer de la siguiente manera:
Se toma un caso cualquiera de (4) y se mapea a un caso de (7) en el que se permite que (5) y (6) tomen valores arbitrarios. Por lo tanto

$$(4) \text{ se reduce polinomialmente a } minimizar(f_1, \cdot, \cdot)$$

lo cual es simplemente (4), por lo tanto la demostración está completa. □

Sin embargo, minimizar más de una función objetivo al mismo tiempo no es algo trivial. Surgen complicaciones que no existen en los problemas mono-objetivo. Veamos qué significa minimizar (f_1, f_2 y f_3).

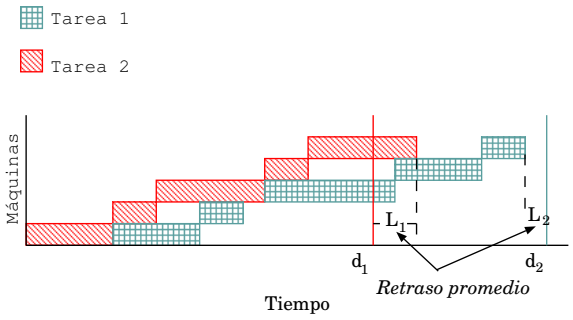

Figura 5: Ilustración del promedio del retraso para un problema de 2 tareas y 5 máquinas

II.2 Problemas Multi-objetivo

Un problema multi-objetivo tiene varios componentes:

- **Variables de decisión**. Son las componentes de un vector $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ cuyos valores deben ser escogidos para la optimización.

- **Restricciones**. Son parte inherente del problema, pueden ser restricciones físicas, de tiempo, o de cualquier otro tipo. Estas restricciones *deben* ser satisfechas para que la solución sea factible. Pueden ser restricciones de desigualdad:

$$g_i(\mathbf{x}) \geq 0 \qquad i = 1, 2, \ldots, k \tag{8}$$

  o de igualdad:

$$h_i(\mathbf{x}) = 0 \qquad i = 1, 2, \ldots, p \tag{9}$$

- **Funciones objetivo**. Son las funciones que se deben optimizar (maximizar o minimizar). Se puede ver como un vector de la forma

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})]^T$$

Las variables de decisión y las funciones objetivo definen dos espacios a considerar:

- El espacio de las variables de decisión $\mathbf{x}$ (decision space), el cual es de dimensión n. Cada punto en este espacio corresponde a un vector $\mathbf{x}$.
- El espacio de criterios $\mathbf{f}(\mathbf{x})$ (criteria space), el cual es de dimensión m. Cada punto es la correspondiente imagen, a través de $\mathbf{f}(\cdot)$, de una solución $\mathbf{x}$ al problema.

Definición 1. El *Punto Ideal* o *Vector Ideal* es el punto $\mathbf{z}^*$ que está compuesto por los mejores valores objetivo que se pueden obtener. Esto es, $z_j^* = \max\{f_j(\mathbf{x})|\mathbf{x} \in \mathbf{A}\}$ (si se está maximizando), $\mathbf{A}$ es el conjunto (sin restricciones) de todos los posibles valores de $\mathbf{x}$, $j = 1, 2, \ldots, m$, donde $m > 1$ es el número de funciones objetivo.

Basándonos en la definición 1 se puede presentar el concepto de funciones objetivo en conflicto.

Definición 2. *Dos funciones objetivo están en conflicto* si la distancia Euclidiana desde el punto ideal al conjunto de los mejores valores de las soluciones factibles es diferente de cero.

Definición 3. *Tres o más funciones objetivo están en conflicto* si están en conflicto por pares.

El hecho de que las funciones objetivo estén en conflicto hace que se tengan varias soluciones para el mismo problema multi-objetivo. Es decir, mientras un problema mono-objetivo se resuelve encontrando el mejor valor para una función dada; un problema multi-objetivo se resuelve encontrando un conjunto de vectores que no pueden ser comparados entre sí. Para aclarar esto último presentamos a continuación las relaciones que pueden darse entre dos vectores.

II.3 Relación entre vectores

Para dos vectores cualesquiera $\mathbf{a} = [a_1, a_2, \ldots, a_m]^T$ y $\mathbf{b} = [b_1, b_2, \ldots, b_m]^T$ existen cuatro posibles relaciones entre ellos (para ejemplificar supondremos minimización, pero en caso de maximización sólo se cambia el signo de *menor que* por el signo de *mayor que*):

1. El vector $\mathbf{a}$ *domina* al vector $\mathbf{b}$ ($\mathbf{a} \prec \mathbf{b}$). Para cada componente de $\mathbf{b}$ los componentes de $\mathbf{a}$ son iguales o mejores ($a_i \leq b_i \forall i = 1, 2, \ldots, m$; y $a_i \neq b_i$ para al menos un i). La figura 6 (extrema izquierda) muestra un ejemplo.

2. El vector $\mathbf{a}$ *es dominado* por el vector $\mathbf{b}$ ($\mathbf{a} \succ \mathbf{b}$). Para cada componente de $\mathbf{a}$ los componentes de $\mathbf{b}$ son iguales o mejores ($a_i \geq b_i \forall i = 1, 2, \ldots, m$; y $a_i \neq b_i$ para al menos un i). En la figura 6 (centro izquierda) se ve un ejemplo.

3. El vector $\mathbf{a}$ *es igual* al vector $\mathbf{b}$ ($\mathbf{a} = \mathbf{b}$). En cada uno de los componentes de $\mathbf{a}$, los componentes de $\mathbf{b}$ son los mismos ($a_i = b_i \forall i = 1, 2, \ldots, m$). La figura 6 (centro derecha) muestra un ejemplo.

4. El vector $\mathbf{a}$ *es no comparable* con el vector $\mathbf{b}$ ($\mathbf{a} \succ\!\prec \mathbf{b}$). Este caso ocurre cuando ninguno de los tres primeros casos sucede. En este caso se dice que ambos vectores son igual de buenos (o igual de malos) ($a_i \leq b_i$ y $a_j \geq b_j$ para algún $i, j = 1, 2, \ldots, m$; y $\mathbf{a} \neq \mathbf{b}$). Esta situación se ilustra en la figura 6 (extrema derecha).

Estas relaciones son de gran utilidad, ya que cada solución de un problema multiobjetivo se puede ver como un vector de m componentes, donde m es el número de funciones objetivo. Cada componente del vector es el valor que toma la función objetivo respectiva. Por ejemplo, para un problema con dos funciones objetivo, un vector solución sería $\mathbf{a} = [a_1, a_2]^T$, donde $a_1 = f_1(\mathbf{x})$ y $a_2 = f_2(\mathbf{x})$.

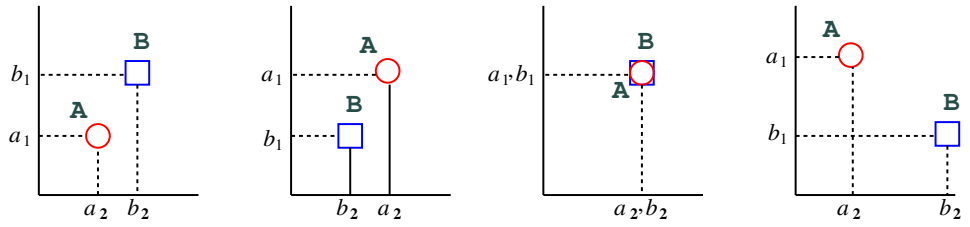

Figura 6: Relación de dominancia entre dos vectores de longitud m (minimizar). Extrema izquierda $\mathbf{a} \prec \mathbf{b}$. Centro izquierda $\mathbf{a} \succ \mathbf{b}$. Centro derecha $\mathbf{a} = \mathbf{b}$. Extrema derecha $\mathbf{a} \succ\prec \mathbf{b}$

Ahora bien, lo que queremos encontrar al resolver un problema multi-objetivo, es un *conjunto de soluciones*. Una solución no es suficiente (como en el caso mono-objetivo) porque, como se mencionó antes, se tienen objetivos en conflicto. El conjunto que queremos encontrar se denomina Frente Pareto y lo definimos a continuación.

II.4 Frente Pareto

La solución a un problema multi-objetivo es un conjunto de soluciones que tienen el mejor valor posible en todas las funciones objetivo, y que no pueden ser mejores entre ellas. Este conjunto se llama Frente Pareto, soluciones *no-inferiores, admisibles* o *no-dominadas* (Steuer, 1986).

Definición 4. Para un problema de optimización multi-objetivo, el *conjunto óptimo de Pareto* ($\mathbf{P}_{true}$) es

$$\mathbf{P}_{true} := \{\mathbf{x} \in \mathbf{A} | \nexists \mathbf{y} \in \mathbf{A} \quad \mathbf{f}(\mathbf{y}) \prec \mathbf{f}(\mathbf{x})\}$$

A es el conjunto de soluciones que cumplen con (8) y (9).

Definición 5. Para un problema de optimización multi-objetivo y un conjunto óptimo de Pareto $\mathbf{P}_{true}$, el *frente Pareto* se define como:

$$\mathbf{PF}_{true} := \left\{ \mathbf{u} = \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})]^T \,|\mathbf{x} \in \mathbf{P}_{true} \right\}$$

Entonces, un problema multi-objetivo se define de la siguiente manera:

Definición 6. *Problema multi-objetivo.* Encontrar los vectores $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ que satisfacen las k restricciones de desigualdad:

$$g_i(\mathbf{x}) \geq 0 \qquad i = 1, 2, \ldots, k \tag{10}$$

las p restricciones de igualdad:

$$h_i(\mathbf{x}) = 0 \qquad i = 1, 2, \ldots, p \tag{11}$$

y optimizan el vector de funciones

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})]^T \tag{12}$$

donde optimizar $\mathbf{f}(\mathbf{x})$ significa encontrar el conjunto óptimo de Pareto.

La diferencia entre el conjunto óptimo de Pareto y el frente Pareto es que el primero se encuentra en el espacio de decisión y el frente Pareto se encuentra en el espacio de criterios. Es por esto que las soluciones a un problema multi-objetivo se presentan (por lo general) en el espacio de criterios.

En resumen, al enfrentarse a un problema multi-objetivo, lo que se quiere obtener es el conjunto de soluciones que se mapean al frente Pareto. Si se está tratando de resolver un problema multi-objetivo con funciones continuas, lo que se quiere encontrar es una gama de puntos que cubran (discretamente) lo más posible del frente (continuo) Pareto.

Se han utilizado varios métodos para tratar de resolver este tipo de problemas: combinación lineal de pesos (Zadeh, 1963), método de la constante ε (Osyczka, 1984), método para resolver problemas secuenciales multi-objetivo (Goicoechea *et al.*, 1982), entre otros.

Cuando se cuenta con dos o más métodos distintos para resolver un problema multi-objetivo se debe saber cuál produce los mejores resultados. Esto no es tan fácil como en el caso mono-objetivo, en el que cada método produce una solución única y el mejor método es el que produce la solución menor o mayor (minimizar o maximizar). En el caso de multi-objetivo, la comparación debe realizarse entre *conjuntos de soluciones*. Para tratar de comparar estos conjuntos se han propuesto varios criterios, los cuales son explicados en la siguiente sección.

II.5 Criterios para la Evaluación de Conjuntos No-dominados

El término de métrica como se utiliza en Knowles y Corne (2002) y en van Veldhuizen y Lamont (2000) no es lo mismo que se define en análisis funcional (Gamelin y Greene, 1983), donde toda métrica se define como:

Definición 7. Sea $\mathbf{E}$ un conjunto. *Una distancia o métrica* sobre $\mathbf{E}$ es una función $d : \mathbf{E} \times \mathbf{E} \rightarrow \mathbb{R}$ que cumple las siguientes propiedades:
$\forall x, y, z \in \mathbf{E}$

1. Positividad: $d(x, y) \geq 0$
2. No degeneración: $d(x, y) = 0 \Rightarrow x = y$
3. Simetría: $d(x, y) = d(y, x)$

4. Desigualdad del triángulo: $d(x, y) + d(y, z) \geq d(x, z)$

Debido a que las "métricas" propuestas hasta ahora no cumplen con estas cuatro propiedades, nosotros las llamaremos de aquí en adelante criterios. Los criterios que se utilizan son un mapeo que toma como argumento un conjunto y le asigna un número real. Varios de estos criterios son en realidad funciones de utilidad (ver Steuer, 1986, página 139). Además, estos criterios están pensados para evaluar distintos aspectos de los conjuntos de soluciones:

1. Cercanía al frente Pareto. Evalúa qué tan alejado del frente Pareto real están las soluciones obtenidas. Los criterios que caen en esta categoría son: Razón de Error, Distancia Generacional y Error Máximo del Frente Pareto.

2. Distribución. Evalúa qué tanto del frente Pareto está siendo cubierto por las soluciones obtenidas. El criterio $\mathcal{S}$ es un ejemplo.

3. Dispersión. Dice qué tan equidistantes sobre el frente Pareto están las soluciones obtenidas. Por ejemplo, el criterio *spacing*.

Sin embargo, no hay un criterio que evalúe los tres aspectos a la vez. Por el contrario, es un problema multi-objetivo decidir qué conjunto de soluciones es el mejor debido a que puede haber dos o más conjuntos mejores dependiendo del criterio seleccionado.

Antes de comenzar a enumerar los diferentes criterios, se necesitan algunas definiciones (Hansen y Jaszkiewicz, 1998).

Si se tienen dos conjuntos $\mathbf{A}$ y $\mathbf{B}$, $ND(\mathbf{A}) = \{$Conjunto de soluciones No-Dominadas de $\mathbf{A}\}$ se define:

Definición 8. Superioridad débil – Weak Outperformance.
$\mathbf{A}O_W\mathbf{B} \Leftrightarrow ND(\mathbf{A} \bigcup \mathbf{B}) = \mathbf{A}$ y $\mathbf{A} \neq \mathbf{B}$. $\mathbf{A}$ *supera débilmente a* $\mathbf{B}$ si todos los puntos

de **B** son 'cubiertos' por los puntos de **A** ('cubiertos' significa domina o es igual) y hay al menos un punto en **A** que no está en **B**. Esto se ilustra en la figura 7 (Izquierda).

Definición 9. Superioridad fuerte – Strong Outperfomance.
$\mathbf{A}O_S\mathbf{B} \Leftrightarrow ND(\mathbf{A} \bigcup \mathbf{B}) = \mathbf{A}$ y $\mathbf{B} \backslash ND(\mathbf{A} \bigcup \mathbf{B}) \neq \emptyset$. **A** *supera fuertemente a* **B** si todos los puntos en **B** son cubiertos por los puntos de **A** y algún punto en **B** es dominado por un punto en **A**. La figura 7 (Centro) muestra un ejemplo.

Definición 10. Superioridad completa – Complete Outperformance.
$\mathbf{A}O_C\mathbf{B} \Leftrightarrow ND(\mathbf{A} \bigcup \mathbf{B}) = \mathbf{A}$ y $\mathbf{B} \bigcap ND(\mathbf{A} \bigcup \mathbf{B}) = \emptyset$. **A** *supera completamente a* **B** si cada punto en **B** es dominado por un punto en **A**. La figura 7 (Derecha) ilustra esta definición.

Como se ve de las definiciones 8, 9 y 10: $\mathbf{A}O_C\mathbf{B} \Rightarrow \mathbf{A}O_S\mathbf{B} \Rightarrow \mathbf{A}O_W\mathbf{B}$. Esto es, *superioridad completa* es la más estricta de las relaciones y *superioridad débil* es la menos estricta.

Hansen y Jaszkiewicz (1998) proponen estos tipos de compatibilidad:

Definición 11. Compatibilidad débil – Weak Compatibility.
Una relación de comparación $\leq_R$ es *débilmente compatible* con una medida de desempeño R si para cada par de conjuntos no-dominados $\mathbf{A}, \mathbf{B}$ con $\mathbf{A} \leq_R \mathbf{B}$, R evaluará que **A** no es mejor que **B**.

Definición 12. Compatibilidad – Compatibility.
Una relación de comparación $\leq_R$ es *compatible* con una medida de desempeño R si para cada par de conjuntos no-dominados **A** y **B**, tales que $\mathbf{A} \leq_R \mathbf{B}$, R evaluará que **A** es mejor que **B**.

Definición 13. Monotonía (débil). Al agregar un punto no-dominado a un conjunto no-dominado dado **A**, se mejora (no se empeora) su evaluación.

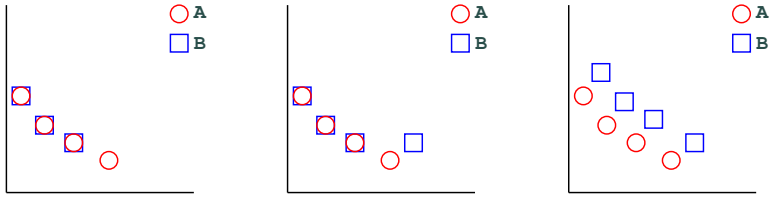

Figura 7: Relaciones de superioridad. Izquierda: Superioridad débil ($\mathbf{A}O_W\mathbf{B}$). Centro: Superioridad fuerte ($\mathbf{A}O_S\mathbf{B}$). Derecha: Superioridad completa ($\mathbf{A}O_C\mathbf{B}$)

Las definiciones anteriores nos sirven para saber qué características tiene un criterio R en particular. Si, por ejemplo, el criterio R es compatible con O_W significa que al tener dos conjuntos ($\mathbf{A}$ y $\mathbf{B}$) cuyas soluciones se ven como en la figura 7 (Izquierda), R va a evaluar correctamente que $\mathbf{A}$ es el mejor conjunto. Si por el contrario, R no es compatible con O_W entonces evaluará que el mejor conjunto es $\mathbf{B}$.

Además de las definiciones anteriores, nosotros hemos dividido a los criterios en: unarios y binarios. Los unarios son los que simplemente mapean un conjunto a un valor real, y la comparación entre conjuntos se realiza mediante la comparación directa de los valores reales. Los criterios binarios son aquellos que comparan dos conjuntos de soluciones y producen un valor real, el valor de ese número informa sobre el resultado de la comparación.

II.5.1 Criterios unarios

Para todos los criterios siguientes $\mathbf{A}$ es el conjunto soluciones. Todos los criterios se aplican sobre este conjunto.

Criterio $\mathcal{S}$

Una definición de este criterio está en Zitzler (1999). $\mathcal{S}$ calcula el hipervolumen de la región multidimensional encerrada por $\mathbf{A}$ y un 'punto de referencia', por lo tanto calcula el tamaño de la región que $\mathbf{A}$ domina. Este criterio está diseñado para evaluar la distribución del frente no-dominado. Mientras mayor sea el valor que el conjunto obtiene en este criterio, mejor es el conjunto (en cuanto a distribución). Es compatible con O_W siempre y cuando el punto de referencia se escoja de forma que cualquier punto no-dominado válido sea evaluado positivo. Una gran desventaja es la selección del punto de referencia. Diferentes puntos de referencia producen distintos valores y esto puede dar lugar a evaluaciones erróneas.

Razón de Error (Error Ratio - ER)

Está definido en van Veldhuizen (1999) como $\sum_{i=1}^{n} \mathbf{a}_i/n$ donde n es el número de vectores en el conjunto de soluciones $\mathbf{A}$; $\mathbf{a}_i = 0$ si el vector i está en $\mathbf{PF}_{true}$ y 1 de otra forma. Este criterio se enfoca en el aspecto de cercanía al frente Pareto. Un valor de 0 significa que todos los puntos están en el frente Pareto real. Es débilmente compatible con O_C y no es débilmente compatible con O_S u O_C. Este criterio es fácil de entender y de calcular. Aunque se necesita conocimiento de $\mathbf{PF}_{true}$.

Distancia Generacional (Generational Distance - GD)

Se encuentra en van Veldhuizen (1999) y está dado por $GD = \sqrt{\sum_{i=1}^{n} d_i^2}/n$, donde n es el número de vectores en el conjunto de soluciones $\mathbf{A}$, y d_i es la distancia en el espacio de criterios entre el vector i y el miembro *más cercano* de $\mathbf{PF}_{true}$. Esta es otra forma de medir la cercanía al frente Pareto. Si el conjunto tiene todos sus puntos en el frente Pareto obtendrá 0 en este criterio. No es débilmente compatible con O_W, pero es compatible con O_S; sin embargo, viola la monotonía débil. Es fácil de calcular. Al

verificar la cercanía al frente real se requiere conocimiento de $\mathbf{PF}_{true}$. Otro problema con este criterio es que se suman y se multiplican diferentes objetivos a la vez, introduciendo problemas de normalización y escalamiento.

Error Máximo del Frente Pareto (Maximum Pareto Front Error - MPFE)

De acuerdo a van Veldhuizen (1999) el criterio MPFE se define como:

$$MPFE = \max_{j}(\min_{i} \left| f_1^i(\mathbf{x}) - f_1^j(\mathbf{x}^*) \right|^p + \left| f_2^i(\mathbf{x}) - f_2^j(\mathbf{x}^*) \right|^p)^{1/p} \tag{13}$$

donde $\mathbf{x}^* \in \mathbf{P}_{true}$ y además $i = 1, \ldots, n_1$ y $j = 1, \ldots, n_2$ son los índices de los vectores en $\mathbf{A}$ y en $\mathbf{PF}_{true}$, respectivamente. MPFE mide la mayor distancia entre cualquier vector en $\mathbf{A}$ y su vector correspondiente más cercano en $\mathbf{PF}_{true}$. Este criterio también mide la cercanía al frente Pareto. Los conjuntos que obtienen menores valores son mejores, ya que un valor de 0 en este criterio significa que todos los puntos están en el frente Pareto real. No es débilmente compatible con cualquier relación de superioridad y viola la monotonía débil. Una ventaja de este criterio es que es fácil de calcular. Aunque una desventaja es que también requiere conocimiento de $\mathbf{PF}_{true}$.

Generación de Vectores No-dominados (Overall Nondominated Vector Generation - ONVG)

Se define como $ONVG = |\mathbf{A}|$ (van Veldhuizen, 1999). Este criterio cuenta el número de puntos no-dominados en el conjunto $\mathbf{A}$. Esto quiere decir que mientras mayor sea el valor que el conjunto obtiene en este criterio, es mejor. Aunque no es débilmente compatible con ninguna relación de superioridad. Además, no tiene monotonía débil. Este criterio es muy fácil de calcular. Pero, y tal vez debido a su sencillez, hay casos en los que un conjunto $\mathbf{A}$ es mejor que otro conjunto $\mathbf{B}$ en este criterio, aunque en realidad $\mathbf{B}$ sea mejor (en cuanto a relaciones de superioridad) que $\mathbf{A}$. Este criterio no requiere conocimiento de $\mathbf{PF}_{true}$.

Espaciamiento de Schott (Schott's Spacing metric - SS)

Schott (1995) define el siguiente criterio:

$$SS = \sqrt{\frac{1}{n-1}\sum_{i+1} n(\bar{d} - d_i)^2} \tag{14}$$

donde $d_i = \min_j(|a_1^i - a_1^j| + |a_2^i - a_2^j| + \ldots + |a_m^i - a_m^j|)$. $i, j = 1, \ldots, n$. $\bar{d}$ es la media de todas las d_i y $n = |\mathbf{A}|$. SS intenta medir qué tan uniformemente distribuidos están los puntos, por lo que un valor de 0 indica que todos los puntos son equidistantes. El criterio no es débilmente compatible con O_W y tiene un bajo costo computacional.

Cuenta de Pareto (Pareto Count - PC)

Este criterio es parecido a ONVG, pero solamente cuenta el número de soluciones que están realmente en $\mathbf{PF}_{true}$. Se define como $PC = \sum_{i=1}^{n} x_i$, donde $x_i = 1$ si la i-ésima solución está en $\mathbf{PF}_{true}$, 0 de otro modo. Este criterio es igual a $n{*}ER$, pero la incluimos para poder hacer comparaciones con ONVG.

II.5.2 Criterios binarios

El criterio $\mathcal{C}$ (The $\mathcal{C}$ metric)

Está definido en Knowles (2002). Sean $\mathbf{A}, \mathbf{B}$ dos conjuntos de vectores no-dominados. $\mathcal{C}$ mapea el par ordenado $(\mathbf{A}, \mathbf{B})$ en el intervalo [0,1]:

$$\mathcal{C}(\mathbf{A}, \mathbf{B}) = \frac{|\{\mathbf{b} \in \mathbf{B} | \exists \mathbf{a} \in \mathbf{A} : \mathbf{a} \preceq \mathbf{b}\}|}{|\mathbf{B}|} \tag{15}$$

el símbolo $\preceq$ significa 'domina o es igual'.

El valor $\mathcal{C}(\mathbf{A}, \mathbf{B}) = 1$ significa que todos los vectores de $\mathbf{B}$ son dominados por $\mathbf{A}$. Lo opuesto, $\mathcal{C}(\mathbf{A}, \mathbf{B}) = 0$, significa que ninguno de los puntos de $\mathbf{B}$ es dominado por $\mathbf{A}$. Pero $\mathcal{C}(\mathbf{A}, \mathbf{B})$ no es necesariamente $1 - \mathcal{C}(\mathbf{B}, \mathbf{A})$; esto se puede apreciar en la figura

8. Este es un criterio que intenta explicar cuál es la relación entre el conjunto **A** y el conjunto **B**. En general $\mathcal{C}$ no es compatible con O_W. Sin embargo, sí es compatible con O_S y con O_C. Si hay dos conjuntos en los cuales no se cumple que $\mathcal{C}(\mathbf{A},\mathbf{B}) = 1$ y tampoco $\mathcal{C}(\mathbf{B},\mathbf{A}) = 1$ entonces indica que los dos conjuntos son incomparables en términos de la relación de superioridad débil. Obtener conclusiones no es aconsejable en este caso. Este criterio tiene la ventaja de ser independiente de la escala y además no requiere conocer $\mathbf{PF}_{true}$.

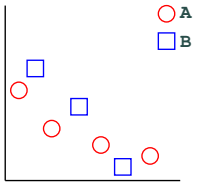

Figura 8: Ejemplo de un caso en que $\mathcal{C}(\mathbf{A},\mathbf{B}) \neq 1 - \mathcal{C}(\mathbf{B},\mathbf{A})$. $\mathcal{C}(\mathbf{A},\mathbf{B}) = 2/3$. $\mathcal{C}(\mathbf{B},\mathbf{A}) = 1/4$

Criterio R1 (R1 metric)

Definido en Hansen y Jaszkiewicz (1998) como:

$$R1(\mathbf{A},\mathbf{B},\mathbf{U},p) = \int_{u\in\mathbf{U}} C(\mathbf{A},\mathbf{B},u)p(u)du$$

$$C(\mathbf{A},\mathbf{B},u) = \begin{cases} 1 & u^*(\mathbf{A}) > u^*(\mathbf{B}) \\ 1/2 & u^*(\mathbf{A}) = u^*(\mathbf{B}) \\ 0 & u^*(\mathbf{A}) < u^*(\mathbf{B}) \end{cases}$$

donde **A** y **B** son dos conjuntos de soluciones, **U** es un conjunto de funciones de utilidad que asignan a cada punto en el espacio de criterios una medida de utilidad, $p(u)$ es la función de densidad de probabilidad de la utilidad $u \in \mathbf{U}$, y $u^*(\mathbf{A}) = \max_{\mathbf{z}\in\mathbf{A}}\{u(\mathbf{z})\}$ y lo mismo para **B**. Este criterio introduce cierto conjunto de funciones de utilidad,

tratando de "adivinar" las preferencias de quien toma las decisiones. Estas preferencias dependen tanto del problema como de la forma de la solución y las estudia el que toma las decisiones (*decision maker* o *DM* es como suele llamársele, dentro del área de Investigación de Operaciones). Lo que **R**1 hace es calcular la probabilidad de que **A** sea mejor que **B** *sobre* este conjunto de funciones de utilidad. Sin embargo, la funcionalidad de este criterio depende de la selección del conjunto de funciones de utilidad, las cuales se pueden escoger sin conocer $\mathbf{PF}_{true}$. En cuanto a la compatibilidad con las relaciones de superioridad si $\mathbf{U}(A > B) = \{u \in \mathbf{U} | u^*(\mathbf{A}) > u^*(\mathbf{B})\}$ y si $p(\mathbf{u})$ es tal que la probabilidad de seleccionar una función de utilidad $u \in \mathbf{U}(\mathbf{A} > \mathbf{B})$ es positiva siempre que $\mathbf{U}(\mathbf{A} > \mathbf{B}) \neq \emptyset$ entonces **R1** es compatible con O_W.

II.6 Criterio Porcentaje de Dominancia – PD

La mayoría de los criterios anteriores están diseñados para evaluar el desempeño de un conjunto de soluciones para un frente Pareto continuo, sin embargo en los problemas multi-objetivo con un frente Pareto discreto estos criterios no son útiles. Esto se debe a que las características de distribución y dispersión dependen por completo de la propia distribución y dispersión del frente Pareto.

Como el problema de calendarización tiene un espacio de criterios discreto (es un problema combinatorio), los criterios anteriores no nos dan una información fidedigna de lo que está sucediendo con los conjuntos de soluciones. Es por eso que nosotros proponemos trabajar con las relaciones de dominancia directamente. Estas relaciones las establecemos entre 2 ó más conjuntos de soluciones de la siguiente manera:

Se toma el conjunto **A** y se compara contra los conjuntos $\mathbf{B}, \mathbf{C}, \ldots$, etc. Entonces se establecen cuatro medidas de dominancia para el conjunto **A**:

- $m^1_{\mathbf{A}}$: El número de veces que las soluciones de **A** son *no comparables* con las

soluciones en los otros conjuntos.

- $m_{\mathbf{A}}^2$: El número de veces que las soluciones de $\mathbf{A}$ son *dominadas* por las soluciones en los otros conjuntos.

- $m_{\mathbf{A}}^3$: El número de veces que las soluciones de $\mathbf{A}$ *dominan* a las soluciones en los otros conjuntos.

- $m_{\mathbf{A}}^4$: El número de veces que las soluciones de $\mathbf{A}$ *son iguales* a las soluciones en los otros conjuntos.

Con la ayuda de estas definiciones se puede establecer si hay algún conjunto que sea mejor que otro (u otros). Si un conjunto obtiene un alto valor en $m_{\mathbf{A}}^3$ y un valor bajo de $m_{\mathbf{A}}^2$ entonces es, relativamente, un buen conjunto. Para establecer qué tan buen (o mal) conjunto es $\mathbf{A}$ tenemos el criterio PD, que se define en la ecuación (16).

$$\mathrm{PD}(\mathbf{A}) = \frac{m_{\mathbf{A}}^3 - m_{\mathbf{A}}^2}{m_{\mathbf{A}}^3 + m_{\mathbf{A}}^2} \tag{16}$$

Este criterio es simétrico cuando se comparan dos conjuntos. Esto significa que si se compara el conjunto $\mathbf{A}$ contra el conjunto $\mathbf{B}$ entonces $m_{\mathbf{A}}^1 = m_{\mathbf{B}}^1$, $m_{\mathbf{A}}^4 = m_{\mathbf{B}}^4$, $m_{\mathbf{A}}^2 = m_{\mathbf{B}}^3$ y $m_{\mathbf{A}}^3 = m_{\mathbf{B}}^2$. Y por lo tanto $\mathrm{PD}(\mathbf{A})$ es igual a $-\mathrm{PD}(\mathbf{B})$.

Este criterio es compatible con O_W y tiene la ventaja de poder comparar más de dos conjuntos a la vez. Además, la comparación está basada en las relaciones de dominancia por lo tanto no se necesita conocer $\mathbf{PF}_{true}$, y no importa si las funciones objetivo son discretas o continuas. Aunque esto puede ser una desventaja cuando se trata de funciones continuas, ya que no aporta información acerca de la dispersión y la distribución de las soluciones.

Ahora tenemos todos los elementos para poder resolver el problema de flujo de tareas multi-objetivo. Sabemos cuál es el problema, conocemos las dificultades que tienen los problemas multi-objetivo (los objetivos en conflicto), sabemos qué es lo que se busca

(el frente Pareto) y sabemos cómo determinar la calidad de las soluciones (usando los criterios). El siguiente paso es desarrollar metodologías que nos ayuden a resolver el problema. Para eso tenemos que hablar de los Algoritmos Genéticos, los cuales son el tema del siguiente capítulo.

Capítulo III

Algoritmos Genéticos

Los Algoritmos Genéticos son una nueva herramienta para tratar con problemas difíciles. La idea es tomar varias soluciones y verlas como si fueran "individuos" dentro de un "ambiente". Cada individuo tiene asignado un valor de aptitud, algo parecido a qué tan "fuerte" es el individuo dentro de ése ambiente en particular (una función objetivo). En cada iteración del algoritmo, el ambiente "pone a prueba" a los individuos y sólo los mejores (los más aptos) sobreviven. Los individuos se generan por medio de dos mecanismos: cruzamiento y mutación. El proceso de seleccionar, cruzar y mutar individuos se repite hasta que se cumple un criterio de paro (número de iteraciones, estancamiento del algoritmo, etc.) y el mejor individuo de todos es la solución buscada. La figura 9 muestra el diagrama de flujo del algoritmo genético y la descripción más completa del algoritmo es la siguiente:

1. Iniciar aleatoriamente la población.
2. Evaluar cada uno de los individuos en la función de aptitud.
3. Seleccionar a los individuos que se cruzarán.
4. Si se cumple el criterio de paro terminar y reportar al mejor individuo encontrado, de lo contrario ir al paso 2.
5. Realizar cruzamiento.
6. Realizar mutación.

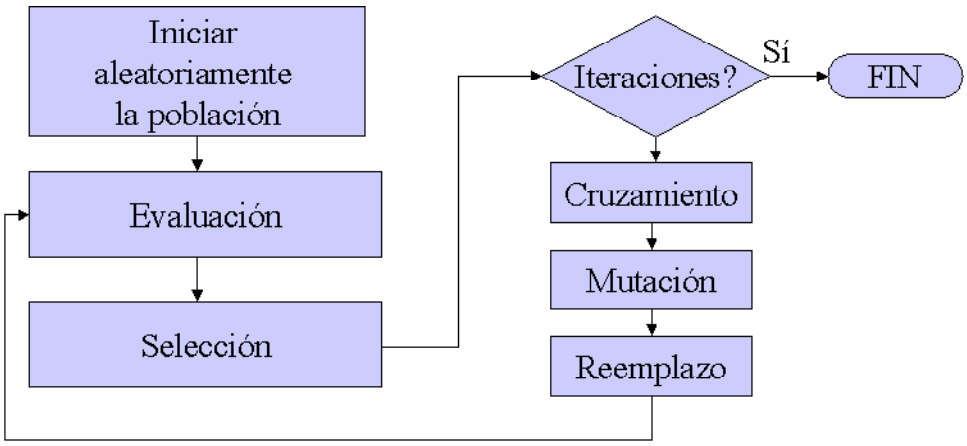

Figura 9: Diagrama de flujo del algoritmo genético estándar

7. Reemplazar a la población.

En el paso 1, a cada individuo se le asigna un valor aleatorio. En el paso 2, se establece el valor de aptitud para cada individuo, esto se hace evaluando cada individuo en la función de aptitud. El paso 3 consiste en seleccionar los individuos que tomarán parte en el cruzamiento. Hay varias maneras de hacer esto: por torneo, por selección proporcional (ruleta), elitismo, etc. En el paso 4 se verifica si se ha cumplido el criterio de paro, entonces se termina el algoritmo y se reporta al mejor individuo encontrado; si no se ha cumplido se repite el proceso a partir del paso 2. El criterio de paro más común es el número de iteraciones, pero también existen otros: el número de veces que el mejor individuo no ha cambiado, la dispersión de la población (cuando se puede medir), etc. En el paso 5 se realiza el cruzamiento entre los individuos seleccionados anteriormente. Hay muchos tipos: de un punto, de dos puntos, basado en orden, basado en precedencia, etc. En el paso 6 a los individuos resultantes del cruzamiento anterior se les aplica una mutación. También existen muchos tipos: simple, inserción, intercambio, etc. Y en el paso 6 se escogen de entre los individuos "viejos" (padres) y los "nuevos" (hijos) los

que formarán la nueva población. Puede ser reemplazo directo, reemplazo elitista, etc. Una descripción detallada de cómo funcionan estos algoritmos se puede encontrar en Holland (1975); Goldberg (1989); Gen y Cheng (1997); Mitchell (1996), entre otros.

Es importante señalar que los algoritmos genéticos trabajan con una representación de la solución, y no con la solución misma. La representación más común es una cadena binaria: cada individuo es en realidad una cadena de unos y ceros. Esta cadena se decodifica de acuerdo a las propiedades del problema. Por ejemplo, si se tiene que encontrar la solución al siguiente problema:

$$\text{Optimizar } f(x,y,z) = \sqrt{\frac{\sum_{i=1}^{N} 2x^2}{zy}}y^2 - \frac{z}{2x} \tag{17}$$

entonces se puede proponer que el valor de x sean los cuatro primeros "bits", los siguientes cuatro para y y los últimos cuatro para z. Claro que con esta representación sólo tendremos números enteros de 0 a 15 para cada variable. Si se necesita un valor decimal se puede dividir el valor binario entre 16 ó entre 32 dependiendo de la precisión que se desea. También se puede optar por una representación real o entera, esto sólo afecta la forma en que se hacen los cruzamientos y las mutaciones, pero el algoritmo sigue siendo el mismo.

Lo anterior es el algoritmo genético estándar (Holland, 1975; Goldberg, 1989; Rudolph, 1994) para un problema mono-objetivo. Como nuestro problema es multi-objetivo se tienen que hacer algunos cambios para producir no una, sino varias soluciones en el frente Pareto. Se pueden obtener distintos algoritmos al hacer diferentes cambios. El cambio más importante es sin duda cómo manejar la diversidad, esto es, cómo agregar algún mecanismo que evite que el algoritmo converja a una solución única. A continuación se describen los algoritmos multi-objetivo usados en este trabajo.

III.1 MOGA

El Algoritmo Genético Multi-Objetivo (Multi-Objective Genetic Algorithm) es el algoritmo estándar para multi-objetivo (Srinivas y Deb, 1994), usado con algunas modificaciones propuestas en Brizuela *et al.* (2001). El algoritmo es el siguiente:

1. Se crea una población inicial de g individuos. Se clasifica a los individuos de acuerdo a los frentes no-dominados.

2. Se asigna un valor de aptitud a cada individuo.

3. Se modifica el valor de aptitud anterior con un valor de aptitud compartido.

4. Se usa selección por ruleta para seleccionar dos individuos para cruzamiento de acuerdo a su valor de aptitud. Se realiza cruzamiento con una probabilidad pc.

5. Se realiza mutación con una probabilidad pm.

6. Se realizan los dos pasos anteriores hasta producir g individuos.

7. Se clasifica en frentes no-dominados la población compuesta por los g padres y los g hijos. Se seleccionan los mejores g individuos.

8. Se regresa al paso 2 hasta completar algún criterio de paro.

La clasificación del paso 2 se realiza por medio de una comparación directa entre todos los individuos. La clasificiación en frentes no-dominados se lleva a cabo de la siguiente manera: los individuos no-dominados están en el primer frente. Si no se consideran a los individuos del primer frente entonces los individuos no-dominados que quedan forman un nuevo frente, el cual es el segundo frente. El tercer frente lo forman los individuos no-dominados al no considerar los individuos del primer ni del segundo frente. Para encontrar a los individuos del cuarto frente se descartan los individuos de

los tres primeros frentes. Se siguen descartando frentes hasta que todos los individuos estén en algún frente no-dominado. El valor de aptitud que se le asigna a cada individuo es la resta del tamaño de la población y el número de individuos clasificados hasta ese momento. Esto con el fin de que los individuos en los frentes más lejanos tengan un valor de aptitud más bajo, como se ve en la figura 10.

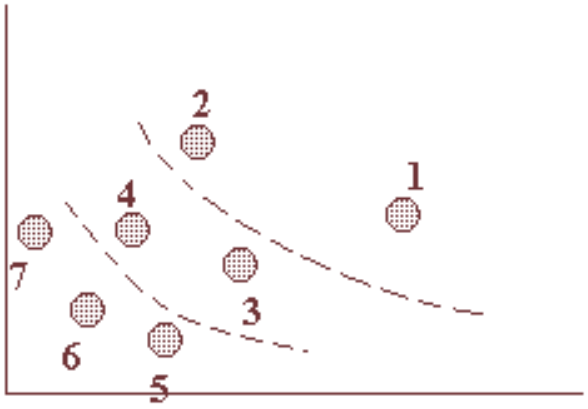

Figura 10: Ejemplo de los valores de aptitud asignados a los individuos. Los individuos en los frentes no-dominados más lejanos tienen un valor de aptitud más bajo

La modificación del paso 3 tiene como fin evitar que los individuos se "aglomeren" y se pierda diversidad. Por eso se crean nichos de radio definido (Horn *et al.*, 1994), cuyos radios se controlan por medio de parámetros externos. Para encontrar el valor de aptitud compartido de un individuo i se calcula la distancia (en el espacio de criterios) entre el individuo i y cada uno de los otros individuos j en la población. Si esta distancia es menor que el primer radio, la cuenta de nichos se incrementa en 1; si está entre el primer radio y el segundo la cuenta de nichos se incrementa en

$$\frac{1-(d(i,j))^2}{d^2_{share}}$$

donde $d(i, j)$ es la distancia (en el espacio de criterios) entre el individuo i y el individuo j. Después se divide el valor de aptitud asignado en el paso 2 entre el valor de la cuenta de nichos. El resultado de este procedimiento es que los individuos que se encuentran en un frente no-dominado bajo pero con muy poca distancia entre sus vecinos (amontonados), tienen el mismo valor de aptitud que los individuos en un frente no-dominado mayor pero con una mayor distancia hacia sus vecinos (espaciados).

En el paso 4 se seleccionan a los individuos que tomarán parte en el cruzamiento de acuerdo al valor de aptitud compartido asignado en el paso 3. Esto se hace por medio de la clásica ruleta (Bäck, 1996). Después se realiza el cruzamiento entre estos individuos. En el paso 5 se realiza la mutación a los individuos recién cruzados. En el paso 6 se continúa con el cruzamiento y la mutación hasta generar g nuevos individuos. Los g individuos padres y los g individuos hijos se combinan en una superpoblación de tamaño $2g$. Esta superpoblación se clasifica en frentes no-dominados en el paso 7, los mejores g individuos (aquellos que están en los frentes más bajos) reemplazan a la población original (padres). En el paso 8 se repite el proceso hasta que se cumpla algún criterio de paro.

Este algoritmo tiene la ventaja de que conserva los mejores individuos sin perder diversidad. La pérdida de diversidad puede hacer que el algoritmo evolucione hacia un solo individuo en la población final. La diferencia fundamental entre el algoritmo utilizado aquí y el propuesto en Syswerda (1991) es el elitismo. Este mecanismo ayuda a mejorar la calidad de las soluciones en cada iteración. Una desventaja del algoritmo es que tiene que ordenar a la población en frentes de no-dominancia en cada iteración, lo cual tiene un alto costo computacional.

III.2 NSGA-II

Este algoritmo, Algoritmo Genético de Ordenamiento No-dominado (Non-dominated Sorting Genetic Algorithm), fue propuesto por Deb *et al.* (2000). El algoritmo es el siguiente:

1. Se crea una población inicial al azar. Se ordena la población de acuerdo a sus rangos de dominancia. Se hace $i = 0$.

2. Se realiza torneo binario para seleccionar a los individuos que tomarán parte en el cruzamiento.

3. Se realiza cruzamiento y mutación con los individuos seleccionados anteriormente.

4. Se crea una superpoblación que combina a los padres y a los hijos. Se ordena esta población de acuerdo a los frentes de nodominancia.

5. Se toman los frentes de esa población para formar la nueva población. Se toman tantos como sea el tamaño de la población original, cuando menos. A los individuos escogidos se les calcula su distancia de amontonamiento local.

6. Estos individuos se ordenan de la siguiente forma: los individuos con menor rango primero, en caso de empate se escoge el que tenga la mayor distancia de amontonamiento local.

7. Se escogen solamente los primeros N individuos para la nueva población.

8. Se incrementa i. Si i=i_{max} entonces PARAR, de lo contrario ir al paso 2.

Para calcular el rango de dominancia de un individuo i se calculan primero dos variables: n_i que es el número de soluciones que dominan a la solución i y $\mathbf{S_i}$ que es el conjunto de soluciones dominadas por la solución i. Las soluciones que tienen $n_i = 0$

forman el primer frente (ya que nadie las domina). Para encontrar las soluciones del segundo frente se decrementa en uno n_j, donde j es el índice de cada solución dominada por las soluciones del primer frente. Todas las soluciones j que ahora tengan $n_j = 0$ son las soluciones del segundo frente. Para encontrar los frentes subsiguientes se realiza el mismo procedimiento.

En el paso 2 se seleccionan dos individuos al azar y se realiza torneo binario entre ellos. El ganador se coloca en la población que se cruzará y mutará. El torneo binario tiene como criterio un operador ($\geq_n$) llamado 'operador de comparación de amontonamiento'. Al comparar el individuo i con el individuo j por medio de este operador se tendrá que $i \geq_n j$ si i tiene menor rango de dominancia que j, o si i y j están en el mismo rango de dominancia, entonces $i \geq_n j$ si i tiene una 'distancia de amontonamiento local' mayor que j. La distancia de amontonamiento local es la distancia (en el espacio de criterios) del individuo i a su vecino más cercano (Deb *et al.*, 2000).

En la forma clásica de clasificar a la población en frentes no-dominados se necesita comparar cada solución de la población con cada una de las restantes soluciones. El número de comparaciones requeridas es $O(mN)$, donde m es el número de funciones objetivo y N es el número de soluciones en la población. Para encontrar a todos los miembros del primer frente no-dominado la complejidad de este algoritmo es $O(mN^2)$. Para encontrar el siguiente frente no-dominado se descartan temporalmente las soluciones del primer frente y se repite el procedimiento anterior, lo cual tiene otra vez una complejidad de $O(mN^2)$. Esto se repite hasta encontrar todos los frentes no-dominados. El peor caso ocurre cuando sólo hay una solución en cada frente lo cual tiene una complejidad de $O(mN^3)$. Sin embargo, en NSGA-II se calculan las variables n_i y $\mathbf{S_i}$ las cuales requieren $O(mN^2)$ comparaciones. Y para clasificar a la población se sigue el procedimiento descrito arriba, el cual requiere $O(N)$ comparaciones para cada frente. Como a lo más existen N frentes, la peor complejidad de esta parte del algoritmo es $O(N^2)$. La

complejidad total del algoritmo es $O(N^2) + O(mN^2)$, o simplemente $O(mN^2)$. Como se ve, NSGA-II reduce la complejidad de $O(mN^3)$ a $O(mN^2)$. La explicación detallada de este procedimiento y de cómo se calcula la distancia de amontonamiento local están en Deb *et al.* (2000).

III.3 SPEA2

El Algoritmo Evolutivo de Reforzamiento de Pareto (Strength Pareto Evolutionary Algorithm) fue propuesto por Zitzler *et al.* (2002a). Utiliza un "archivo externo" para ir almacenando las mejores soluciones encontradas. Este archivo externo se actualiza en cada iteración. Además, el cruzamiento y la mutación se realizan únicamente entre individuos dentro de este archivo, esto es, entre los mejores individuos. El algoritmo es el siguiente:

1. *Inicialización.* Se crea una población aleatoria $\mathbf{P}_0$ y se crea un archivo externo (conjunto vacío) $\overline{\mathbf{P}}_0 = \emptyset$. Sea $t = 0$.

2. *Asignación de Fitness.* Se calcula el valor de aptitud para los individuos en $\mathbf{P}_t$ y en $\overline{\mathbf{P}}_t$.

3. *Selección Ambiental.* Se copian todos los individuos no-dominados de $\mathbf{P}_t$ y $\overline{\mathbf{P}}_t$ a $\overline{\mathbf{P}}_{t+1}$. Si el tamaño de $\overline{\mathbf{P}}_{t+1}$ es mayor que $\overline{N}$ entonces se reduce $\overline{\mathbf{P}}_{t+1}$ por medio de un operador de truncamiento. Si el tamaño de $\overline{\mathbf{P}}_{t+1}$ es menor que $\overline{N}$ entonces se llena $\overline{\mathbf{P}}_{t+1}$ con individuos dominados en $\mathbf{P}_t$ y $\overline{\mathbf{P}}_t$.

4. *Finalización.* Si $t \geq T$ o si se cumple cualquier otro criterio de paro, hacer $\mathbf{A}$ igual al conjunto de individuos no-dominados de $\overline{\mathbf{P}}_{t+1}$. Parar.

5. *Selección para Reproducción.* Se realiza torneo binario con reemplazo en $\overline{\mathbf{P}}_{t+1}$ para seleccionar los individuos que tomarán parte en el cruzamiento y en la mutación.

6. *Variación.* Se aplica cruzamiento y mutación a los individuos seleccionados en el paso anterior y se hace $\mathbf{P}_{t+1}$ la población resultante. Se incrementa el contador de generaciones ($t = t + 1$). Ir al paso 2.

Aquí N es el tamaño de la población, $\overline{N}$ es el tamaño del archivo externo, T es el número de iteraciones y $\mathbf{A}$ es el conjunto de soluciones no-dominadas resultante.

El archivo externo sirve para almacenar a los mejores individuos encontrados hasta el momento. Es de tamaño fijo y lo novedoso de este algoritmo es la forma en que se actualiza.

El valor de aptitud (*fitness*) del individuo i se calcula en dos pasos. Primero se calcula un valor S_i que es el número de individuos que i domina y después se calcula un valor R_i que es la suma del valor S_j de todos los individuos que dominan a i. El valor R_i incorpora la información de cuántos individuos domina i y además cuántos individuos dominan a i; esto se debe a que si un individuo i es dominado por muchos individuos tendrá un valor R_i muy alto. Los individuos con $R_i = 0$ son los individuos no-dominados. Para evitar que las soluciones converjan a un único punto se incorpora también información de qué tan aislada está la solución i (D_i). El valor de *fitness* total para el individuo i es la suma de $F(i) = R_i + D_i$. Ahora los individuos con $F(i) < 1$ son los individuos no-dominados.

En el paso 3 se lleva a cabo la actualización del archivo. Esto se logra copiando todos los individuos no-dominados al archivo, lo que da lugar a tres casos distintos:

i) El número de individuos no-dominados es igual al tamaño del archivo. No hay más que hacer.

ii) El número de individuos no-dominados es menor al tamaño del archivo. Se ordenan los individuos de acuerdo al *fitness* y se copian los primeros individuos dominados al archivo, tantos como sean necesarios para llenarlo.

iii) El número de individuos no-dominados es mayor al tamaño del archivo. Se realiza un proceso de truncamiento para eliminar individuos. Se remueven los individuos más cercanos entre sí (en el espacio de criterios).

En el paso 4 se finaliza el algoritmo si se cumple algún criterio de paro y se entregan a los individuos del archivo como los mejores individuos encontrados.

En el paso 5 se selecciona por medio de torneo binario los individuos que se cruzarán y mutarán. El torneo binario se realiza comparando los valores de *fitness* de dos individuos del archivo tomados al azar. En el paso 6 se realizan el cruzamiento y la mutación.

El funcionamiento básico de los algoritmos MOGA y NSGA-II es el mismo. Los dos trabajan ordenando a la población en frentes no-dominados. La diferencia entre los dos algoritmos es la manera de encontrar esos frentes no-dominados. Mientras que MOGA lo hace comparando un individuo contra el resto, NSGA-II lo hace calculando el número de individuos que dominan a un individuo en particular. Por lo tanto NSGA-II es más rápido que MOGA. Además MOGA requiere un parámetro externo para controlar qué tan lejana debe estar una solución de otra y NSGA-II calcula este valor automáticamente.

Por otro lado SPEA2 utiliza un archivo externo en donde va almacenando a los mejores individuos encontrados. Este archivo se va actualizando en cada iteración con un procedimiento bastante costoso. El gran costo se deriva de que en cada iteración se determinan los individuos no-dominados de la población actual y del archivo externo. El mecanismo para evitar la pérdida de diversidad se basa en calcular la distancia euclidiana hasta el k-ésimo vecino en el espacio de los criterios, lo cual también es costoso.

Estos algoritmos serán utilizados para resolver el problema de flujo de tareas. Sin embargo, para que un algoritmo genético funcione adecuadamente se debe escoger una

buena representación de las soluciones. A continuación se describe la representación utilizada para nuestro problema.

III.4 Representación

Los algoritmos genéticos por lo general utilizan una representación binaria. En dicha representación cada individuo es una cadena de unos y ceros de longitud fija (aunque existen algunos de longitud variable). El significado de cada número depende del problema (ver página 32 para un ejemplo). Sin embargo, en el problema de flujo de tareas esta representación no es la adecuada debido a que la solución del problema es una permutación de números enteros. Asignar un mapeo entre una cadena binaria y una permutación es más complicado que trabajar con la permutación misma.

Por lo tanto, en todos los algoritmos nuestra representación es entera, es decir, cada individuo es una permutación de los primeros N números naturales; donde N es el número de tareas. De acuerdo a esta representación tenemos distintos tipos de cruzamiento y de mutación.

III.4.1 Cruzamiento

Usamos cuatro tipos de cruzamiento:

- **OBX** (Order Based Crossover). Este es el cruzamiento basado en orden (ver Gen y Cheng, 2000, página 239), propuesto por Syswerda para el TSP. La posición de algunos genes correspondientes al padre 1 son mantenidos en el hijo y los elementos que aún no han sido copiados del padre 2 son copiados al hijo, en el orden en que aparecen en el padre. Se genera una máscara como la que se muestra en la figura 11 (izquierda) para este cruzamiento.

- **PPX** (Precedence Preservative Crossover). (Bierwirth *et al.*, 1996). Un subconjunto de relaciones de precedencia de los genes de los padres son preservados en el hijo. La figura 11 (derecha) muestra cómo trabaja este operador. Los 1's en la máscara indican que los genes del padre 1 serán copiados y los 0's indican que los genes serán copiados del padre 2, en el orden en que éstos aparecen, de izquierda a derecha.

- **OSX** Cruzamiento de un segmento. Este es similar al cruzamiento de dos puntos (Gen y Cheng, 2000, página 408) donde también se seleccionan dos puntos al azar. En la figura 12 (izquierda) se muestra cómo trabaja este operador, aquí los genes del padre 1 de los *loci* 1 al $s1$ son copiados en los *loci* 1 a $s1$ en el hijo, y de los *loci* $s1$ al $s2$ (en el hijo) se copia del padre 2 empezando del *locus* 1 y copiando todos aquellos genes que no han sido copiados del padre 1. Desde el *locus* $s2+1$ hasta n, los genes se copian del padre 1 considerando solamente los genes que todavía no han sido copiados.

- **Twopoint**. Se seleccionan dos puntos al azar en ambos padres, se copia del padre 1 desde el *loci* 1 al $s1$ y del $s2$ al n en el hijo. Los *loci* faltantes ($s1$ a $s2$) se copian del padre 2 considerando los genes que no han sido copiados. Esto se muestra en la figura 12 (derecha).

III.4.2 Mutación

Los operadores de mutación que se usaron aquí también son estándar, todos ellos son explicados en Gen y Cheng (1997). Se usaron los siguientes operadores de mutación.

Insert. Se seleccionan dos *loci* (s_1, s_2) al azar, si $s_1 < s_2$ entonces el gen correspondiente a s_1 se coloca en s_2 y todos los genes desde $s_1 + 1$ hasta s_2 se recorren una

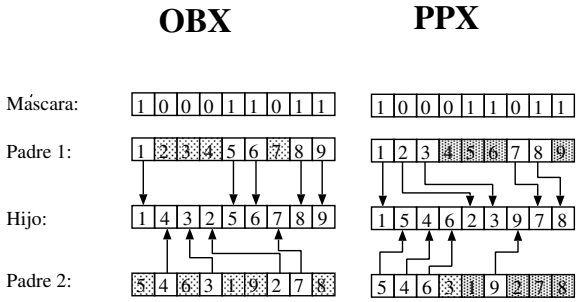

Figura 11: Ejemplo de Obx y Ppx para nueve tareas

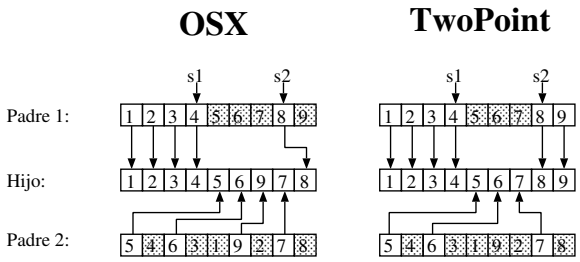

Figura 12: Ejemplo de Osx y TwoPoint para nueve tareas

posición hacia s_1. Si $s_1 > s_2$ entonces la operación de corrimiento se realiza hacia s_2 como se muestra en la figura 13 (izquierda).

Swap. Se seleccionan dos *loci* al azar y sus genes se intercambian. En la figura 13 (centro) se muestra un ejemplo.

Switch. Se realiza un intercambio simple entre dos genes adyacentes. Se selecciona al azar el *locus* para intercambio (s_1). Una vez que se selecciona este *locus* el gen correspondiente se intercambia con su sucesor inmediato a la derecha. Si se selecciona el último gen (*locus* n) entonces éste se intercambia con el primero (*locus* 1). En la figura 13 (derecha) se muestra un ejemplo.

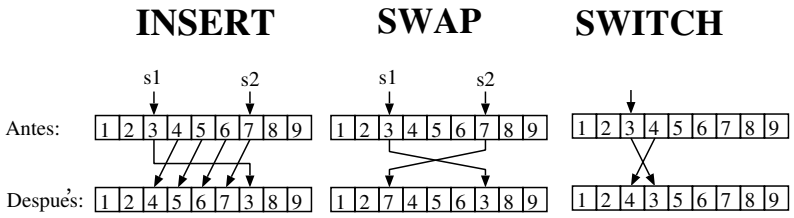

Figura 13: Operadores de mutación INSERT, SWAP y SWITCH para un problema de 9 tareas

III.5 Algoritmos Genéticos en Calendarización

Los algoritmos genéticos se han convertido en una alternativa eficaz para resolver problemas de optimización combinatoria. El problema del viajante (Traveling Salesman Problem – TSP) es uno de los más famosos en el cual se han aplicado con éxito un gran número de estos algoritmos.

Se han hecho algunos intentos para solucionar el problema de calendarización monoobjetivo por medio de los algoritmos genéticos (Jain y Bagchi, 2000; Cartwright y Tuson, 1994; Murata *et al.*, 1996) entre otros. También existen trabajos que contemplan el caso multi-objetivo del problema de flujo de tareas (Tagami y Kawabe, 1998, 1999; Talbi *et al.*, 2001; Ishibuchi y Murata, 1998; Brizuela *et al.*, 2001; Ishibuchi *et al.*, 2003).

Jain y Bagchi (2000) proponen un algoritmo genético híbrido y lo aplican para minimizar el *makespan*; lo que hacen es agregarle al algoritmo genético una heurística tomada del área de investigación de operaciones propuesta por Ho y Chang (1991). Cartwright y Tuson (1994) proponen un algoritmo genético para minimizar el *makespan* en un problema de calendarización de procesos químicos (Ku *et al.*, 1993) dentro de un ambiente dinámico. En el trabajo de Murata *et al.* (1996) se comparan diferentes operadores genéticos de cruzamiento y de mutación. También proponen dos algoritmos híbridos: en el primero combinan algoritmos genéticos y búsqueda local, y en el segundo

combinan algoritmos genéticos con búsqueda tabú.

En cuanto a los trabajos de calendarización multi-objetivo Tagami y Kawabe (1998, 1999) usan un algoritmo genético que crea particiones del frente Pareto para resolver el problema de flujo de tareas, minimizan dos objetivos: *makespan* y retraso. Verifican su algoritmo con 20, 40 y 50 trabajos asignados a 10 máquinas. Su algoritmo usa una representación entera, un cruzamiento llamado PMX (Bierwirth *et al.*, 1996), mutación inversa y selección por ruleta. Comparan sus resultados contra MOGA, un algoritmo genético simple y el método iterativo mejorado (*Iterative Improvement Method (IIM)*). Sus comparaciones son sólo gráficas e indican que el método de partición de Pareto produce soluciones mejor distribuidas que las generadas con MOGA.

Talbi *et al.* (2001) realizan un estudio comparativo de algoritmos usados para resolver el problema de flujo de tareas; consideran dos objetivos a minimizar: *makespan* y retraso total; comparan los algoritmos NSGA, MOGA, VEGA y una versión elitista del NSGA. También experimentan con diferentes tipos de aptitud compartida: fenotipo, genotipo y una combinación de ambas. Lo que se compara es el número de elementos en el conjunto óptimo de Pareto y la calidad de las soluciones generadas. No logran establecer un ganador, aunque el elitismo y la búsqueda local fueron muy útiles para mejorar el rendimiento.

Ishibuchi y Murata (1998) usan un algoritmo genético con una combinación lineal de los objetivos a minimizar junto con una estrategia de búsqueda local, minimizan tres objetivos: *makespan*, retraso máximo y tiempo total de flujo. Su algoritmo consiste en un algoritmo genético con una función de agregación en la cual los pesos son generados al azar mientras se realiza la recombinación. Estos pesos son usados como direcciones de búsqueda para el algoritmo de búsqueda local que se aplica a los hijos, después del cruzamiento y la mutación. La búsqueda local está restringida para evitar un costo computacional excesivo. Se utiliza cruzamiento de dos puntos, mutación *shift*, selección

de ruleta con escalamiento lineal y elitismo. Los resultados se comparan con VEGA y un algoritmo genético mono-objetivo. El algoritmo propuesto mejora a ambos en términos de la calidad y de la dispersión de las soluciones generadas.

Brizuela *et al.* (2001) usan un NSGA para resolver el problema de flujo de tareas. Minimizan tres objetivos: *makespan*, tiempo promedio de flujo y tiempo promedio de retraso. Usan diferentes tipos de cruzamiento y de mutación junto con selección de ruleta y elitismo. La meta de este trabajo es estudiar la influencia de los operadores en la generación de vectores no-dominados. El resultado de este análisis fue usado para diseñar un algoritmo evolutivo multi-objetivo que mejoró (en términos de la cercanía a $\mathbf{PF}_{true}$) al ENSGA propuesto por Bagchi (1999).

En el trabajo de Ishibuchi *et al.* (2003), el cual es el seguimiento de Ishibuchi *et al.* (2002), utilizan un algoritmo que combina búsqueda local y algoritmos genéticos, a este algoritmo le llaman MOGLS (Multi-Objective Genetic Local Search). Muestran que se puede combinar su MOGLS con otros algoritmos como NSGA-II y SPEA2; y que al hacerlo dichos algoritmos mejoran su desempeño. También muestran que la hibridización propuesta reduce el tiempo computacional empleado. Sin embargo, su principal contribución es que concluyen que debe existir un balance entre la intensidad y la profundidad de la búsqueda local empleada en los algoritmos genéticos para el problema de flujo de tareas.

Capítulo IV

Experimentos y Resultados

Para poder obtener alguna conclusión respecto a cuál es la mejor combinación de operadores genéticos para el problema del flujo de tareas multi-objetivo, realizamos una serie de experimentos sobre dos conjuntos de casos: un conjuno de casos de tamaño "pequeño" (de 9 tareas y 5 máquinas) y otro conjunto de casos de tamaño grande (de 75 tareas y 20 máquinas).

IV.1 Casos de 9x5

El tamaño de este caso hace posible encontrar el conjunto óptimo de Pareto mediante enumeración completa. Este tamaño se escogió para poder evaluar qué tan cerca están las soluciones producidas por los algoritmos del conjunto óptimo ($\mathbf{PF}_{true}$).

IV.1.1 Datos de Entrada

Este problema fue generado de manera aleatoria tanto para los tiempos de procesamiento como para las fechas límite, la tabla I muestra los datos de entrada para un caso particular; el elemento (i, j) identifica el tiempo procesamiento de la j-ésima operación de la tarea i. La última columna es el tiempo límite de cada tarea. Así, la intersección del renglón tres (3) con la columna cuatro (4) es el tiempo que tarda la cuarta operación de la tarea tres (3), que es 92 minutos.

Además, y como se mencionó anteriormente, el conjunto óptimo se puede encontrar por enumeración. A pesar de tener tan sólo 9 tareas, el número de soluciones posibles es

Tabla I: Datos de entrada del caso de 9x5 (en minutos). Los renglones son las tareas y las columnas son las máquinas. La intersección es el tiempo que tarda el proceso de cada tarea. La sexta columna es el tiempo límite para cada tarea

	P1	P2	P3	P4	P5	Tiempo límite
Tarea 1	52	0	24	44	72	1424
Tarea 2	40	68	58	28	57	152
Tarea 3	46	41	81	92	36	1992
Tarea 4	38	97	28	27	91	1056
Tarea 5	19	6	86	13	50	1912
Tarea 6	61	78	37	96	55	876
Tarea 7	55	48	7	79	93	1228
Tarea 8	79	20	13	89	0	804
Tarea 9	70	35	93	3	27	2396

$9! = 362,880$; por lo que el número de comparaciones requeridas para encontrar el frente Pareto real es $9! * 9! = 131.6$ miles de millones (por eso las comillas en "pequeño"). Este número, si bien es enorme, todavía es computacionalmente alcanzable. Se desarrolló un algoritmo que genera las 9! permutaciones de los números $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$[1], calcula el valor de f_1, f_2 y f_3 para cada una de ellas y compara dichos valores entre sí. Al final se obtiene el conjunto $\mathbf{PF}_{true}$. Este proceso se tarda un total aproximado de 20 minutos[2]. Sin embargo, agregar una tarea más incrementa el número de comparaciones a tal grado que encontrar el $\mathbf{PF}_{true}$ tarda un total aproximado de 2 días, además de que requiere considerablemente más espacio de almacenamiento. Esto es un orden mayor que con la versión mono-objetivo, en la que para encontrar la solución óptima se requieren $n!$ comparaciones.

[1]El tamaño de este archivo es de 3.11 Mbytes en formato binario

[2]En una máquina AMD Athlon 4 a 900 Mhertz

IV.1.2 Generación de Resultados

El primer experimento se realizó con MOGA, con los parámetros que se muestran en la tabla II. Para probar el desempeño de los operadores genéticos de cruzamiento y mutación descritos en §III.4.1 y §III.4.2 se crearon 12 combinaciones distintas. Son 12 combinaciones porque se toma cada operador de cruzamiento con cada uno de los operadores de mutación, siendo cuatro (4) operadores de cruzamiento y tres (3) de mutación tenemos 12 combinaciones en total. En la tabla III se muestran los 12 nombres de las distintas combinaciones entre operadores y sus correspondientes significados.

Tabla II: Parámetros de MOGA para el conjunto de casos de 9x5

Parámetro	Valor
Tamaño de la población	30
Probabilidad de cruzamiento	0.9
Probabilidad de mutación	0.01
d_{Share}	3.00
Número de iteraciones	2000

Tabla III: Nombres de los 12 algoritmos generados así como su correspondiente significado

Nombre	Significado
OsxInsert	Cruzamiento OSX con mutación INSERT
OsxSwap	Cruzamiento OSX con mutación SWAP
OsxSwitch	Cruzamiento OSX con mutación SWITCH
ObxInsert	Cruzamiento OBX con mutación INSERT
ObxSwap	Cruzamiento OBX con mutación SWAP
ObxSwitch	Cruzamiento OBX con mutación SWITCH
PpxInsert	Cruzamiento PPX con mutación INSERT
PpxSwap	Cruzamiento PPX con mutación SWAP
PpxSwitch	Cruzamiento PPX con mutación SWITCH
TwopointInsert	Cruzamiento TWOPOINT con mutación INSERT
TwopointSwap	Cruzamiento TWOPOINT con mutación SWAP
TwopointSwitch	Cruzamiento TWOPOINT con mutación SWITCH

Asimismo, utilizamos los algoritmos NSGA-II y SPEA2 con los parámetros mostrados en las tablas IV y V, respectivamente. En total, los resultados para este caso son 36: 12 generados por MOGA, 12 generados por NSGA-II y 12 más generados por SPEA2. Probamos con 10 casos distintos, por lo tanto tenemos una colección de resultados como se ve en la figura 14.

Tabla IV: Parámetros de NSGA-II para el conjunto de casos 9x5

Parámetro	Valor
Tamaño de la población	30
Probabilidad de cruzamiento	0.9
Probabilidad de mutación	0.01
Número de iteraciones	2000

Tabla V: Parámetros de SPEA2 para el conjunto de casos 9x5

Parámetro	Valor
Tamaño de la población	30
Tamaño del archivo	30
Probabilidad de cruzamiento	0.9
Probabilidad de mutación	0.01
Número de iteraciones	2000

Para tener una mayor confianza en los resultados generados, cada una de las combinaciones se ejecutó 50 veces. Por tanto cada resultado está compuesto de 50 colecciones de conjuntos de soluciones no-dominadas (recordemos, de §II.2, que en multi-objetivo queremos encontrar un *conjunto* de soluciones, no una solución única). El número de soluciones en cada corrida es variable, pero no puede ser mayor que el tamaño de la población.

En la tabla VI se muestra un fragmento de uno de los elementos de la colección vista en la figura 14. En dicha tabla se ven dos bloques: el primero con 8 renglones y el

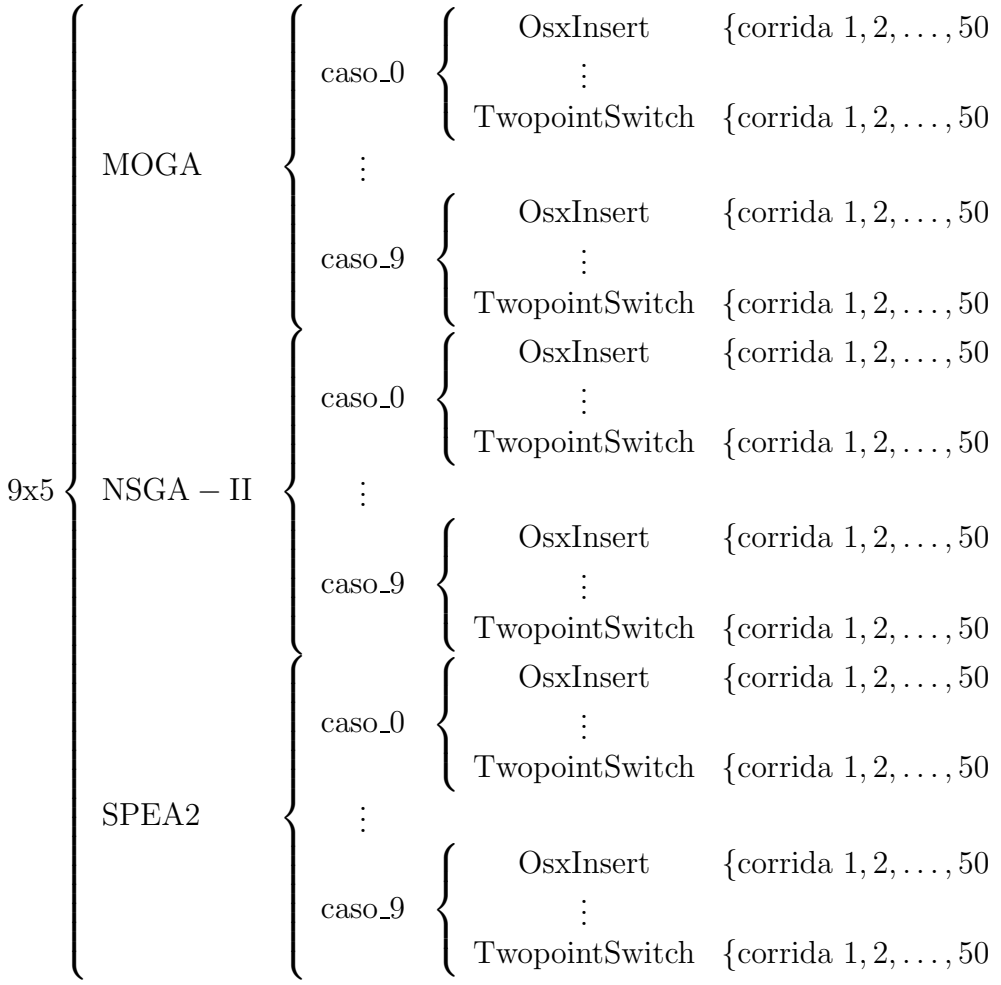

Figura 14: Representación gráfica de los resultados generados para un caso de 9x5

segundo con 9, que corresponden a 2 de las 50 corridas de una de las combinaciones de un caso específico. Los puntos suspensivos representan, que igual a esos dos bloques, hay 48 más, todos de longitud variable. La primera columna corresponde al valor de *makespan*, la segunda corresponde al tiempo promedio de flujo y la tercera al tiempo promedio de retraso. Recordando de §III.4 que cada solución es una permutación de las tareas, entonces ¿por qué estas salidas contienen los valores de las funciones objetivo y no las soluciones en sí? La respuesta es simple: estamos buscando la mejor aproximación al frente Pareto, y dicho frente se encuentra en el espacio de criterios (§II.4) (Coello

Tabla VI: Fragmento de una salida de un caso específico de 9x5

⋮	⋮	⋮
710	13.1111	475.111
710	26.1111	463.889
713	13.1111	464.889
713	26.1111	460.667
736	11	484.111
720	11	487.222
734	26.1111	456.556
736	13.1111	463
662	13.1111	464.333
675	11	500.667
712	31.8889	459
714	11	493.111
724	11	490.111
726	13.1111	459
747	31.8889	453.778
734	11	485.556
726	18.8889	457.556
⋮	⋮	⋮

et al., 2003).

IV.1.3 Análisis de Resultados

Para evaluar la calidad de soluciones generadas por los operadores genéticos, utilizamos 6 criterios: ONVG, SS, GD, ER, MPFE y PC (ver §II.5.1).

Se hizo el cálculo de cada criterio en cada una de las 50 respuestas y después se promedió. En la figura 15 se muestran los resultados del criterio ONVG para los tres algoritmos y las 12 combinaciones. Las barras de error están dadas por el promedio más/menos la desviación estándar. El mejor valor que un conjunto de soluciones puede obtener es 4 (para el caso en estudio), que es el número de respuestas en $\mathbf{PF}_{true}$. De

esta gráfica se ve que ObxSwitch con el algoritmo NSGA-II tiene el mayor promedio.

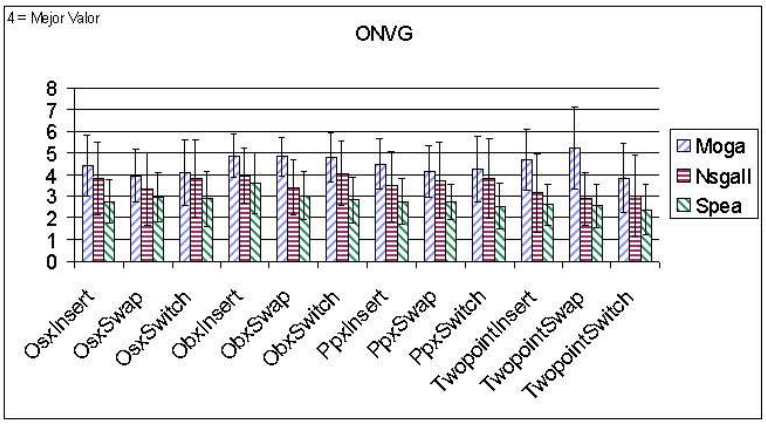

Figura 15: Criterio ONVG para un caso de 9x5

La figura 16 muestra los resultados del criterio SS. Como es un frente Pareto discreto el mejor valor para el criterio SS es el valor de SS que tenga el frente Pareto, que para este caso es SS=22.0989; la combinación que en promedio se acerca más a este valor es PpxSwap con el algoritmo MOGA. En esta figura la desviación estándar de SPEA2 es muy grande, debido a que este algoritmo algunas veces encuentra 0, 1 ó 2 soluciones (SS=0) y en otras corridas encuentra muchas soluciones con un SS muy grande. Lo que se puede concluir aquí es que este criterio no nos ayuda para saber qué tan buena es la solución, por tres razones: *i*) sólo es aplicable a problemas donde se conoce $\mathbf{PF}_{true}$; *ii*) puede llevar a conclusiones erróneas porque, en principio, una buena solución tendría un valor de 0, sin embargo, al tener un frente discreto la única conjetura que podemos hacer es que mientras más se acerque al valor SS de $\mathbf{PF}_{true}$ mejor es la solución; *iii*) tiene el mismo valor cuando el algoritmo encuentra 0,1 ó 2 soluciones.

Las figuras 17 y 18 muestran los resultados de los criterios GD y MPFE, respectivamente. En ambos criterios el mejor valor que un conjunto de respuestas puede obtener es 0; para ambas el que tiene el menor valor es TwopointSwap con NSGA-II. El criterio

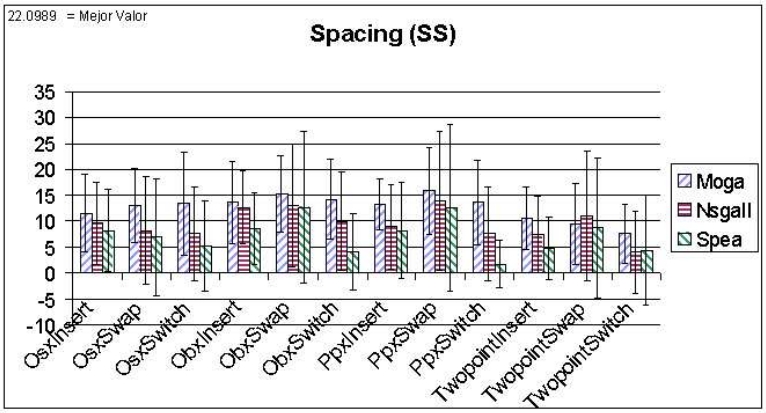

Figura 16: Criterio SS para un caso de 9x5

MPFE refleja la peor solución encontrada por el algoritmo, sin embargo nos interesan los algoritmos que produzcan las mejores soluciones, por lo tanto no nos ayuda en la búsqueda de $\mathbf{PF}_{true}$. GD refleja el promedio de cercanía a $\mathbf{PF}_{true}$, lo cual favorece a los algoritmos que encuentren muchas soluciones cerca de un único punto de $\mathbf{PF}_{true}$ y descalifica a los algoritmos que encuentren soluciones más alejadas de varios puntos distintos de $\mathbf{PF}_{true}$.

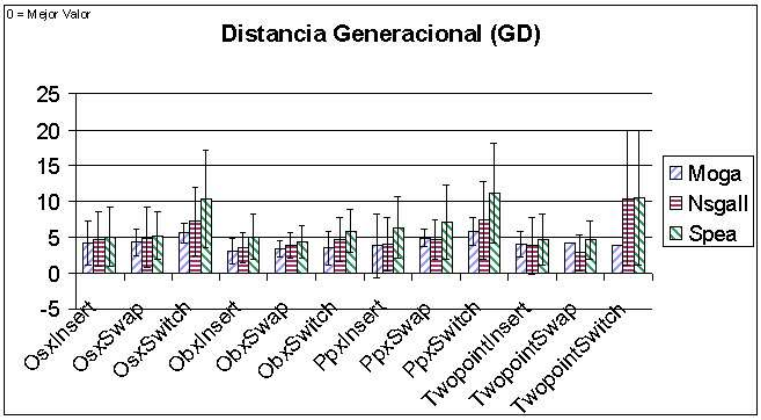

Figura 17: Criterio GD para un caso de 9x5

Las figuras 19 y 20 muestran los resulados de los criterios PC y ER. Para PC el

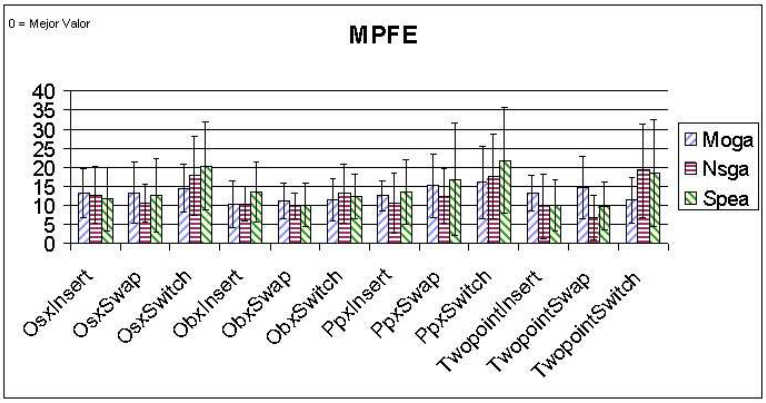

Figura 18: Criterio MPFE para un caso de 9x5

mejor valor que se puede obtener, en este caso, es 4; siendo ObxInsert con MOGA el que más se acerca. Y para ER el mejor valor es 1, siendo TwopointSwap con NSGA-II el que presenta el valor más cercano. A pesar de que PC y ER son en esencia lo mismo hay ocasiones en que no concuerdan en qué algoritmo es el mejor (como en este caso). Mientras PC sólo reporta qué algoritmo obtuvo más respuestas en $\mathbf{PF}_{true}$, ER dice quién obtuvo más en relación con los vectores no-dominados encontrados. Ambos criterios son muy útiles pero tienen la gran desventaja de que sólo pueden ser utilizadas cuando se conoce $\mathbf{PF}_{true}$.

Por último, la figura 21 (Izquierda) muestra los resultados para las relaciones $m^i_{\mathbf{A}}$ (ver §II.6) aplicadas a MOGA. Lo que se puede ver en esta gráfica es que cerca del 60% de las respuestas son *no comparables*, esto significa que todas las combinaciones tienen un desempeño similar. Lo mismo se puede decir de NSGA-II (figura 21 – derecha) y también para SPEA2 (figura 22). La figura 23 (izquierda) muestra los resultados del criterio PD. Esta figura confirma que todas las combinaciones tienen un desempeño similar porque todas están muy cerca unas de otras. Lo mismo se ve para NSGA-II (figura 23 – derecha) y para SPEA2 (figura 24).

Para comprobar que efectivamente los algoritmos se desempeñan de manera similar,

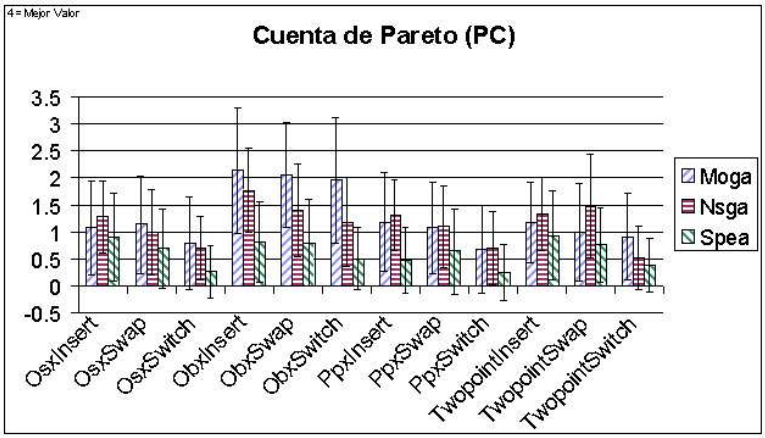

Figura 19: Criterio PC para un caso de 9x5

incluimos la figura 25 donde se ve el resultado del criterio SS aplicado a la población inicial de MOGA contra su población final. Las figuras 26 y 27 muestran la misma comparación para los algoritmos NSGA-II y SPEA2, respectivamente.

También incluimos las comparaciones entre las poblaciones inicial y final de cada algoritmo para el criterio GD. Esto se muestra en las figuras 28, 29 y 30 para los algoritmos MOGA, NSGA-II y SPEA2, respectivamente. La superioridad de las poblaciones finales es evidente.

Por último, la figura 31 muestra el criterio PD aplicada a la comparación entre la población inicial y la población final de MOGA. Solamente comparamos la combinación ObxInsert por ser la que en promedio es mejor. Como se esperaba, la población final domina totalmente a la población inicial, de hecho, el promedio de las 50 corridas es 1 y la desviación estándar 0. Esta misma situación ocurre para NSGA-II y también para SPEA2, figuras 32 y 33.

Todas estas gráficas son para uno de los 10 casos con los que experimentamos. En el apéndice B se reporta otra serie de gráficas para los otros 9 casos. En la figura 34 se muestran los algoritmos que en promedio fueron mejores en cada caso, bajo cada

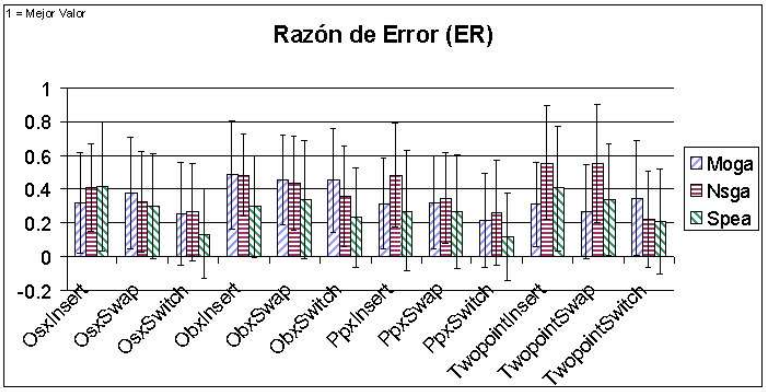

Figura 20: Criterio ER para el caso de 9x5

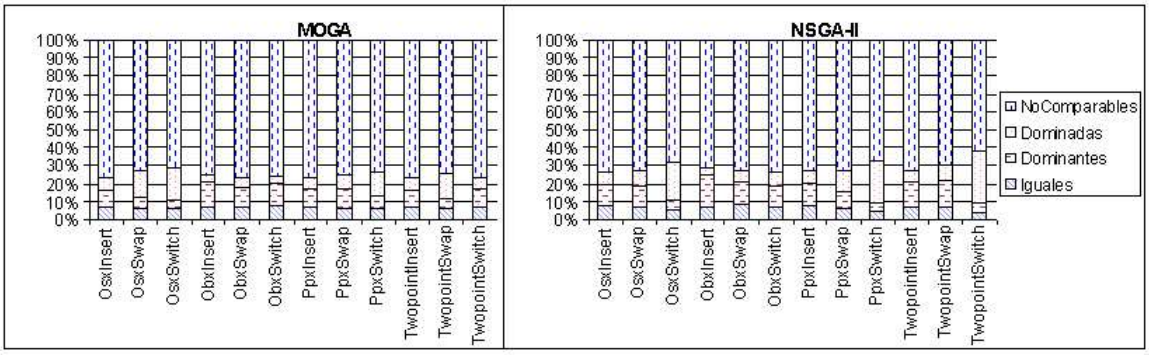

Figura 21: Relaciones $m^i_{\mathbf{A}}$ para MOGA (Izquierda) y NSGA-II (Derecha)

criterio. Ninguna combinación tiene una línea horizontal, eso significa que no fue mejor en todos los casos bajo el mismo criterio.

El criterio PD muestra lo mismo que ONVG, SS, GD, ER, MPFE y PC juntas: que ninguna combinación es mejor que otra. Esto se debe a que el problema es "sencillo" y todos los algoritmos se comportan de manera bastante similar. Sin embargo, esto no ocurre en un problema más grande, como podremos apreciar en la siguiente sección.

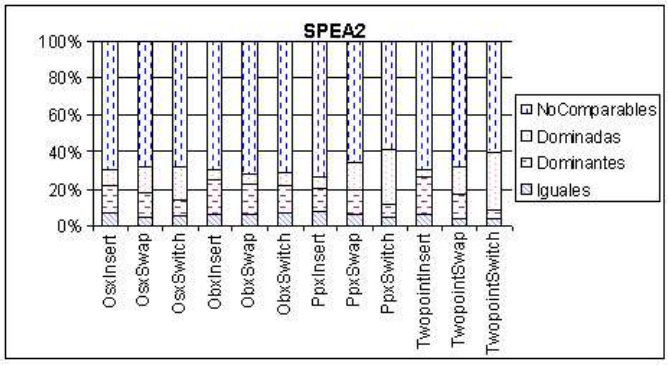

Figura 22: Relaciones $m^i_{\mathbf{A}}$ para SPEA2

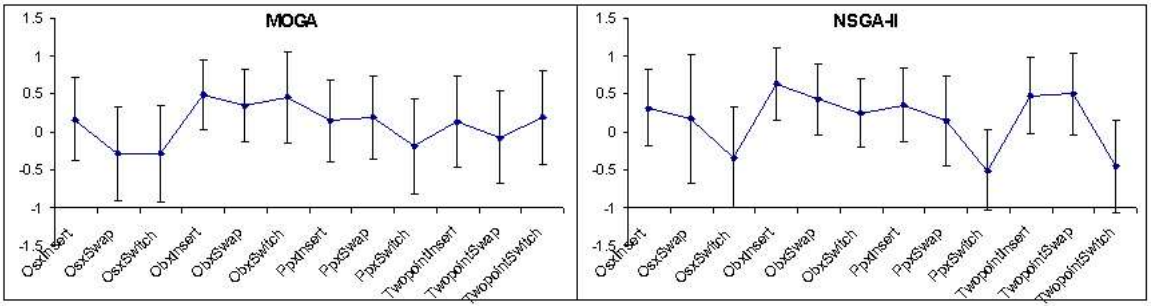

Figura 23: Criterio PD para MOGA (Izquierda) y para NSGA-II (Derecha)

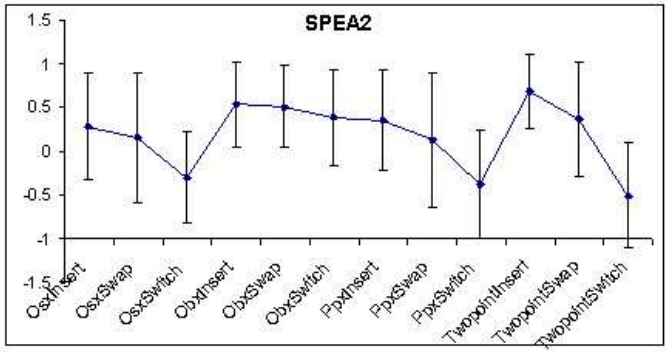

Figura 24: Criterio PD para SPEA2

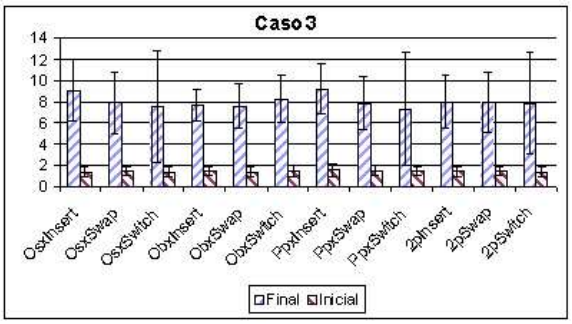

Figura 25: Comparación del criterio SS: población inicial y población final. Algoritmo MOGA

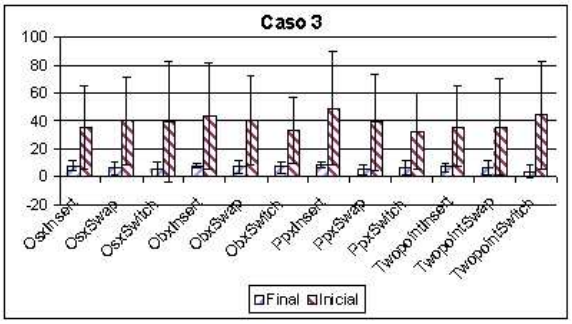

Figura 26: Comparación del criterio SS: población inicial y población final. Algoritmo NSGA-II

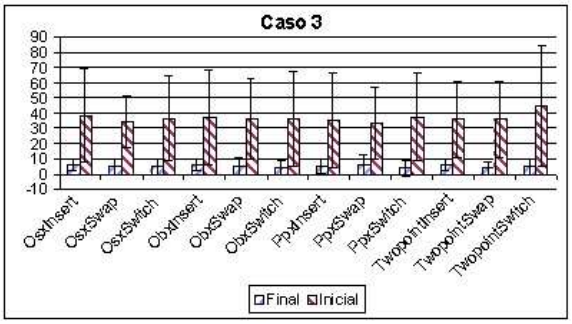

Figura 27: Comparación del criterio SS: población inicial y población final. Algoritmo SPEA2

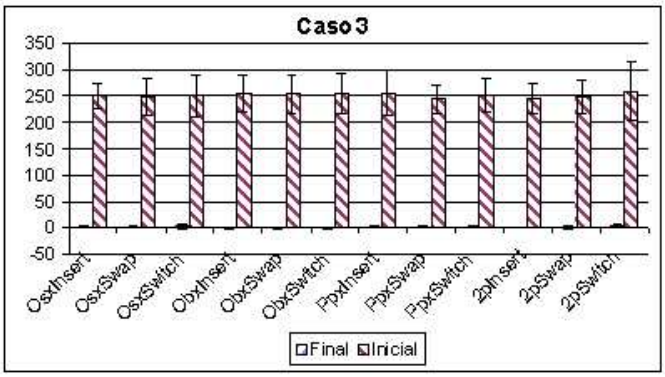

Figura 28: Comparación del criterio GD: población inicial y población final. Algoritmo MOGA

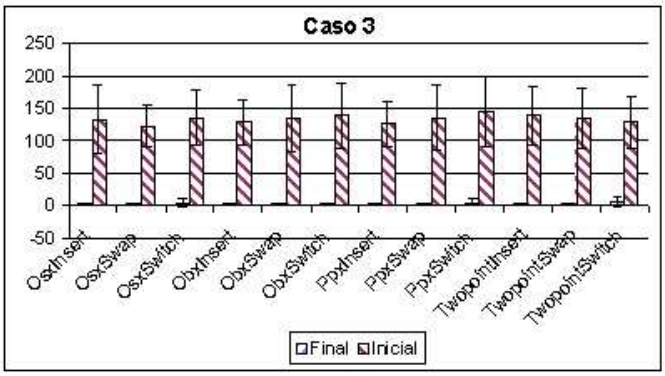

Figura 29: Comparación del criterio GD: población inicial y población final. Algoritmo NSGA-II

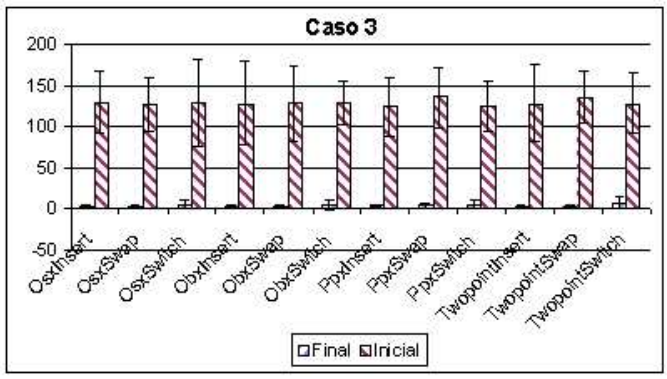

Figura 30: Comparación del criterio GD: población inicial y población final. Algoritmo SPEA2

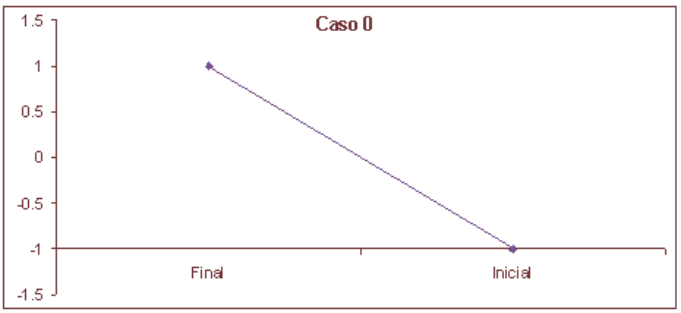

Figura 31: Criterio PD al comparar la población inicial contra la población final de MOGA para la combinación ObxInsert. Caso 0

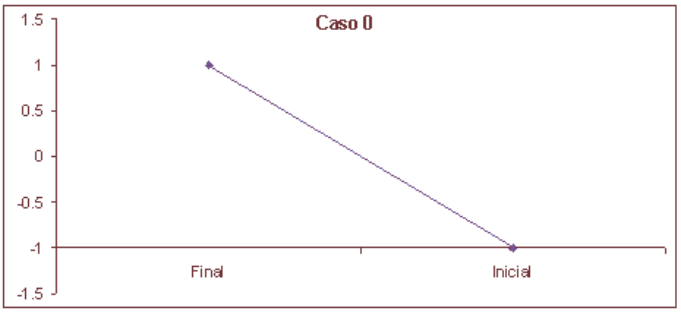

Figura 32: Criterio PD al comparar la población inicial contra la población final de NSGA-II para la combinación ObxInsert. Caso 0

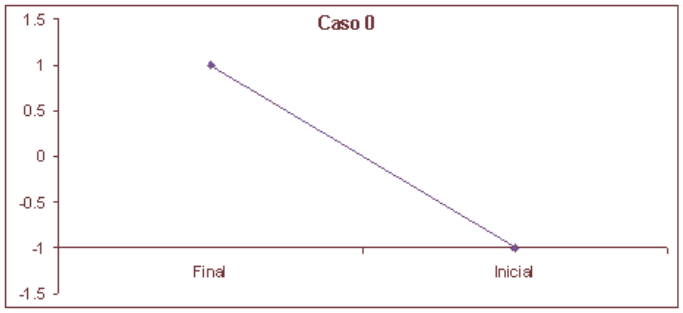

Figura 33: Criterio PD al comparar la población inicial contra la población final de SPEA2 para la combinación ObxInsert. Caso 0

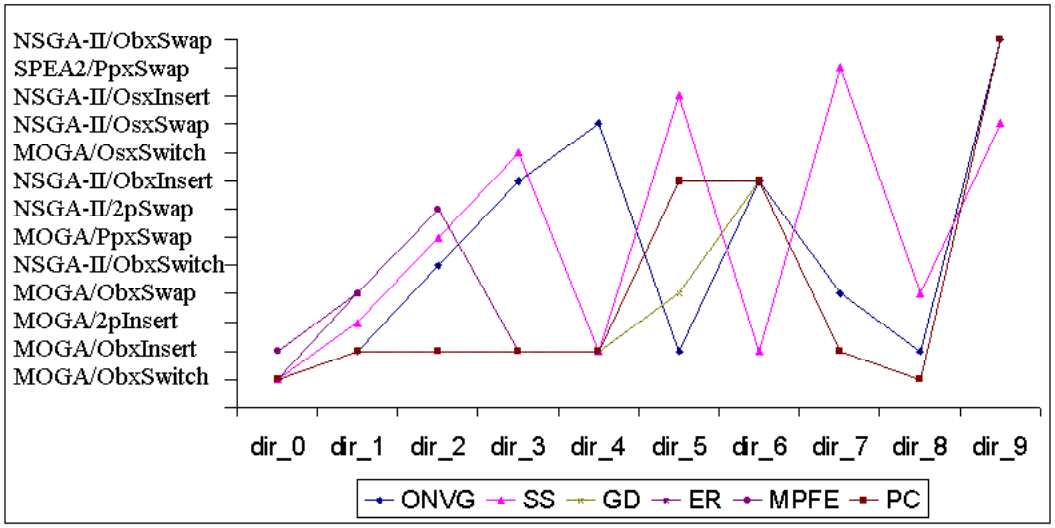

Figura 34: Combinaciones ganadoras para cada uno de los 10 casos de 9x5. Una linea horizontal indicaría que fue mejor en todos los casos bajo el mismo criterio

IV.2 Caso 75x20

Los experimentos que se realizaron con este tamaño del problema tienen como objetivo encontrar experimentalmente una combinación de operadores genéticos que tengan el mejor desempeño. El frente Pareto de este problema es desconocido, ya que aún con las computadoras actuales es imposible enumerar todas las soluciones (son $75! \approx 2.5*10^{109}$) en un tiempo razonable.

IV.2.1 Datos de Entrada

El caso que abordamos se tomó de la biblioteca mantenida por Beasly[3] de 75 tareas y 20 máquinas. Se generaron fechas límite para cada tarea (los casos mantenidos por Beasly no contemplan fechas límite) siguiendo el procedimiento propuesto en Armentano y Ronconi (1999), el cual se explica a continuación.

1. Se calcula el tiempo de cada tarea sumando sus tiempos de procesamiento uno después de otro. Se encuentra el menor de estos tiempos (que llamaremos a).

2. Para cada máquina i se calculan dos tiempos: el menor de los tiempos de procesamiento de las tareas considerando las primeras $1, 2, \ldots, i-1$ máquinas y el menor de los tiempos de procesamiento considerando las $i+1, ..., m$ máquinas. Estos dos tiempos se suman y al máximo de estos tiempos se le llama b. Se hace $p = max\{a, b\}$. Este tiempo es una estimación burda del *makespan*.

3. Se calcula el tiempo límite para cada tarea i con la siguiente fórmula

$$d_i = p - p * t - \text{rand}(p * r) + \frac{p * r}{2} \tag{18}$$

   donde t y r son dos parámetros externos y rand(x) es una función que devuelve un número aleatorio uniformemente distribuido en el intervalo [0,x).

[3]http://mscmga.ms.ic.ac.uk/info.html; disponible al día 12 de octubre de 2003

Usando el procedimiento anterior se generaron 10 casos de 75 tareas y 20 máquinas (75x20), con $t = 0.4$ y $r = 1.2$. Estos casos fueron los datos de entrada para todos los algoritmos.

IV.2.2 Experimentos

Debido a que los algoritmos genéticos tienen un componente estocástico, los resultados generados por el mismo algoritmo, en las mismas condiciones, para dos corridas distintas, son generalmente diferentes. Por esta razón tenemos que correr el algoritmo un número suficientemente grande de veces y después calcular el promedio de los resultados (o cualquier otra medida estadística). Nosotros escogimos correr el algoritmo 50 veces, este número es más bien arbitrario; podría ser 30 ó 100, pero creemos que es un justo medio entre tiempo de procesamiento (una mayor cantidad de veces requiere un mayor tiempo de procesamiento) y significancia estadística confiable (entre menos muestras menos confianza estadística). De hecho, en la comunidad de Computación Evolutiva no existe un consenso acerca del número de corridas necesarias, basado en un análisis profundo. Por ejemplo, Bierwirth *et al.* (1996) utilizan 10 corridas y Starkweather *et al.* (1991) utilizan 30.

Al igual que en los casos de 9x5, primero probamos con MOGA; cuyos parámetros están en la tabla VII. Estos parámetros fueron sugeridos en Brizuela *et al.* (2001). Todos las combinaciones de cruzamiento y de mutación tienen los mismos parámetros y los mismos datos de entrada.

Cada salida contiene los resultados de cada una de las 50 corridas del algoritmo. Cada respuesta es el valor de las funciones objetivo para cada uno de los mejores individuos encontrados. Todo esto es igual al caso de 9x5.

Entonces para el algoritmo MOGA tenemos 10 casos de tamaño 75x20 y para cada caso tenemos 12 salidas (que corresponden a las 12 combinaciones de operadores de

Tabla VII: Parámetros utilizados por MOGA para el caso 75x20

Parámetro	Valor
Tamaño de la población	100
Probabilidad de cruzamiento	0.9
Probabilidad de mutación	0.01
d_{Share}	3.00
Número de iteraciones	2000

Tabla VIII: Parámetros utilizados por NSGA-II para el caso 75x20

Parámetro	Valor
Tamaño de la población	100
Probabilidad de cruzamiento	0.9
Probabilidad de mutación	0.01
Número de iteraciones	2000

cruzamiento y mutación) que contienen las soluciones de 50 corridas encontradas por el algoritmo. Una representación gráfica de esto se encuentra en la figura 35.

Todos estos datos de salida son para el algoritmo MOGA. Como también hicimos pruebas con el algoritmo NSGA-II y con el algortimo SPEA2 (los parámetros de NSGA-II y de SPEA2 están en la tabla VIII y tabla IX, respectivamente), tenemos otra serie de datos iguales para cada uno de los tres algoritmos. En total, tenemos 3 algoritmos, cada algoritmo tiene 10 casos de 75x20, cada caso tiene 12 salidas y cada salida tiene las soluciones de 50 corridas. Esto es similar a lo ilustrado en la figura 14.

$$
\text{MOGA}\begin{cases}
\text{caso\_0}\begin{cases}\text{OsxInsert} & \{\text{corrida } 1,2,\ldots,50\\ \vdots & \\ \text{TwopointSwitch} & \{\text{corrida } 1,2,\ldots,50\end{cases}\\
\vdots\\
\text{caso\_9}\begin{cases}\text{OsxInsert} & \{\text{corrida } 1,2,\ldots,50\\ \vdots & \\ \text{TwopointSwitch} & \{\text{corrida } 1,2,\ldots,50\end{cases}
\end{cases}
$$

Figura 35: Estructura de los datos de salida de MOGA

Tabla IX: Parámetros utilizados por SPEA2 para el caso 75x20

Parámetro	Valor
Tamaño de la población	100
Tamaño del archivo	100
Probabilidad de cruzamiento	0.9
Probabilidad de mutación	0.01
Número de iteraciones	2000

Como se mencionó en §II.6 y se confirmó en §IV.1.3, los criterios propuestos en la literatura para evaluar el desempeño de los algoritmos para soluciones a problemas multi-objetivo no son de utilidad en el caso de problemas de optimización combinatoria. Es por esto que hacemos la evaluación de los distintos algoritmos usando las relaciones $m^i_{\mathbf{A}}$.

De todos los datos que tenemos, vamos a comparar entre sí las 12 combinaciones para cada uno de los 10 casos. Esto lo hacemos de la siguiente manera:

1. Hacemos $i = 1$.
2. Escogemos a la i-ésima combinación como entrada.
3. Se toma al azar un frente no-dominado de la combinación i, éste es el conjunto $\mathbf{A}_i$.
4. Se toma un frente al azar de cada una de las 11 combinaciones restantes.
5. Se compara $\mathbf{A}_i$ con los otros conjuntos para calcular $m^1_{\mathbf{A}_i}, m^2_{\mathbf{A}_i}, m^3_{\mathbf{A}_i}$ y $m^4_{\mathbf{A}_i}$.
6. Se repiten los pasos 3 y 4 para cada uno de los 50 conjuntos sumando los respectivos $m^1_{\mathbf{A}_i}, m^2_{\mathbf{A}_i}, m^3_{\mathbf{A}_i}$ y $m^4_{\mathbf{A}_i}$.
7. Si i es menor que 12 se hace $i = i + 1$ y se repite el proceso desde el paso 2, de lo contrario, continuar al paso 8.

8. Se saca el porcentaje de $m^1_{\mathbf{A}_i}, m^2_{\mathbf{A}_i}, m^3_{\mathbf{A}_i}$ y $m^4_{\mathbf{A}_i}$.

Este procedimiento genera una gráfica como la mostrada en la figura 36. Para que sea más fácil de leer, hemos sustituido $m^1_{\mathbf{A}}$ por NoComparables, $m^2_{\mathbf{A}}$ por Dominadas, $m^3_{\mathbf{A}}$ por Dominantes y $m^4_{\mathbf{A}}$ por Iguales. Además, después de cada figura donde se muestran las relaciones de dominancia, están las correspondientes figuras donde se muestra el criterio PD (ecuación 16, página 28), como en la figura 37.

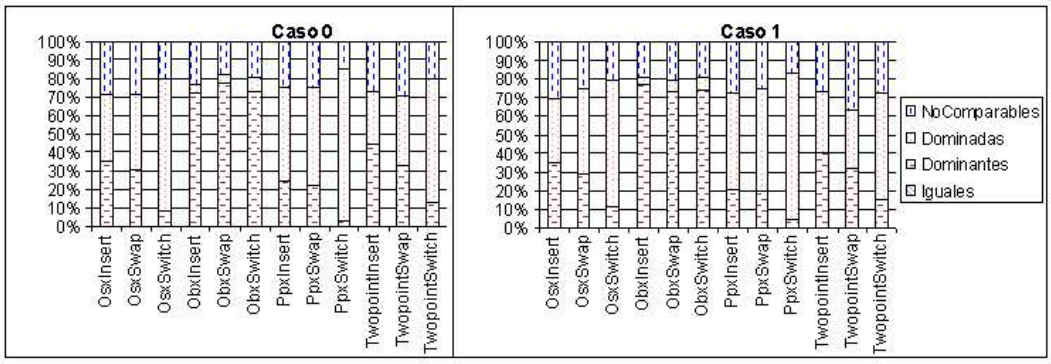

Figura 36: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo MOGA casos 0 y 1

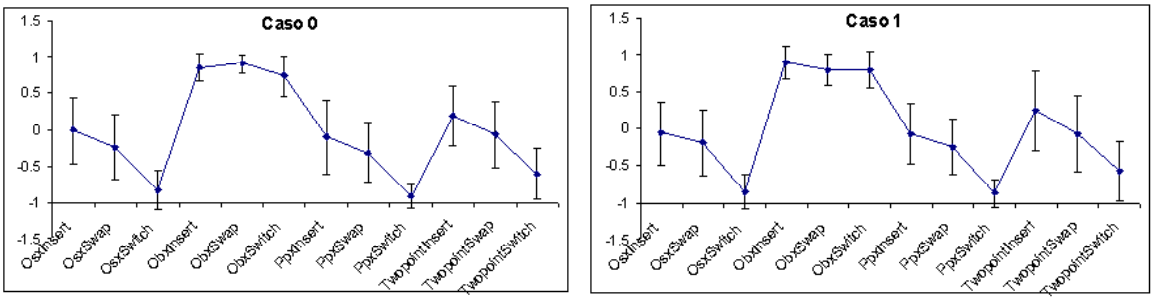

Figura 37: Criterios PD para el caso 75x20. Algoritmo MOGA casos 0 y 1

Todo el proceso anterior se repite para cada uno de los 10 casos de 75x20. Las figuras 36 a 44 muestran los resultados para cada uno de los 10 casos.

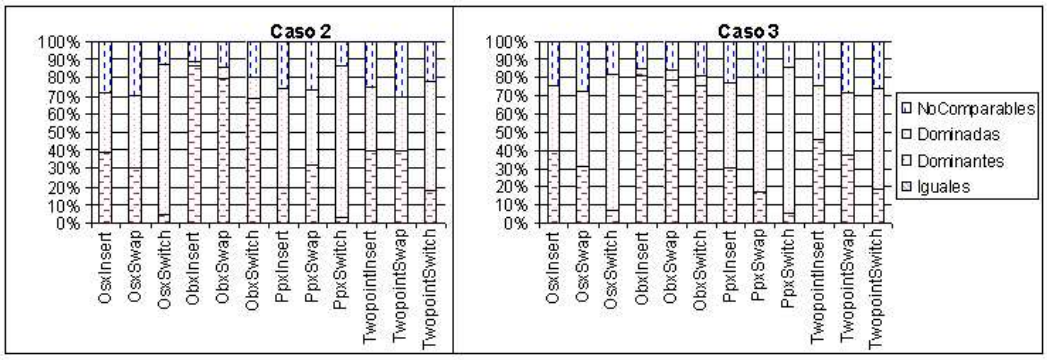

Figura 38: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo MOGA casos 2 y 3

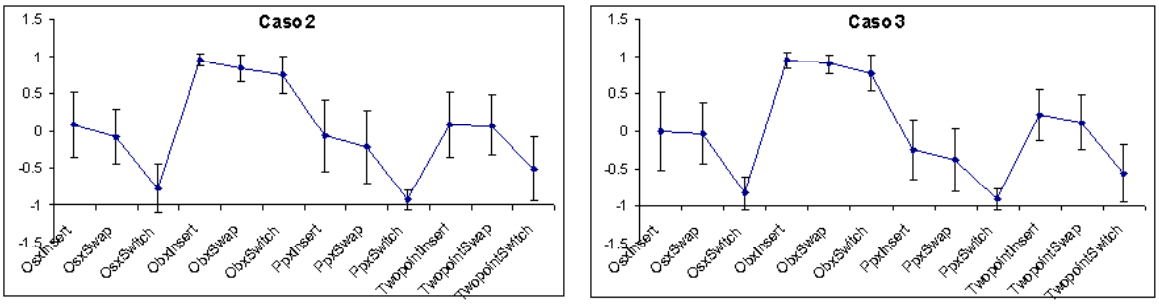

Figura 39: Criterios PD para el caso 75x20. Algoritmo MOGA casos 2 y 3

A su vez todo este procedimiento se repite para el algoritmo NSGA-II, las figuras 48 a 56 muestran estos resultados. Y también para SPEA2, como se ve en las figuras 58 a 66.

Puntualizando, las figuras 36 a 66 muestran el comportamiento de 10 casos del problema de 75x20 (cada caso contiene 50 corridas) resueltos utilizando MOGA, NSGA-II y SPEA2.

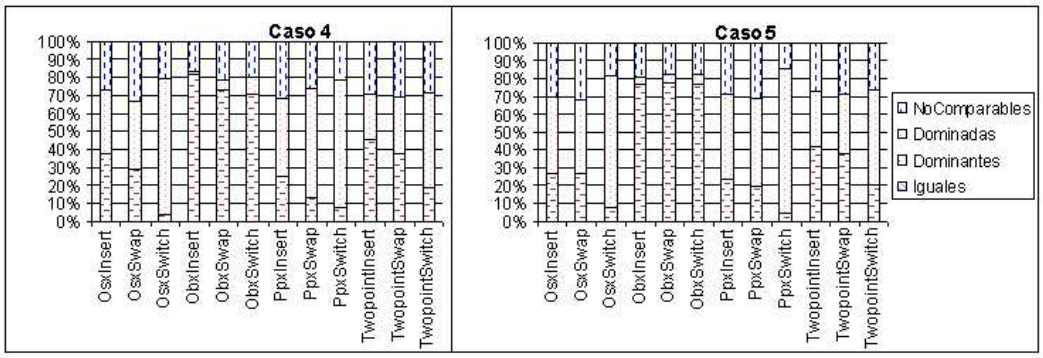

Figura 40: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo MOGA casos 4 y 5

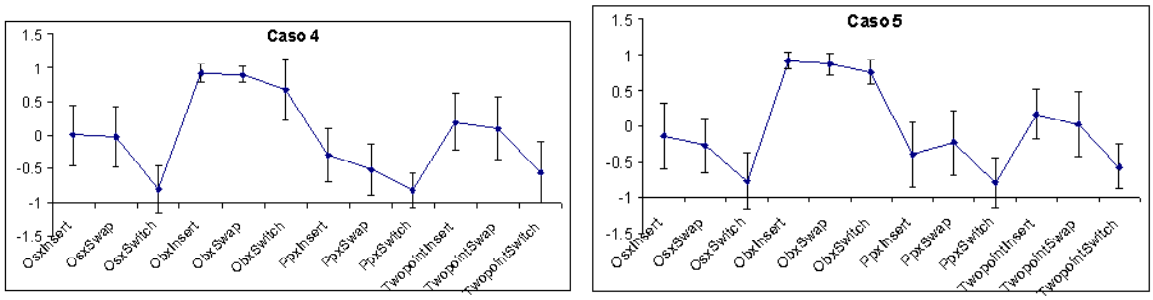

Figura 41: Criterios PD para el caso 75x20. Algoritmo MOGA casos 4 y 5

IV.2.3 Análisis de Resultados

El objetivo principal en un problema multi-objetivo es encontrar el verdadero frente Pareto. Como no lo conocemos, entonces nos interesa obtener la mayor cantidad de soluciones no-dominadas que más se acerquen al frente desconocido. O lo que es lo mismo, el conjunto que domine en un mayor porcentaje a los conjuntos contra los cuales se compara. En todas las figuras el operador que tiene el mayor porcentaje de soluciones no-dominadas es OBX, por tanto concluimos que es el operador de cruzamiento más adecuado para este conjunto de casos. En cuanto a los operadores de mutación no hay una gran diferencia entre Insert y Swap. En la mayoría de los casos es Insert el que tiene

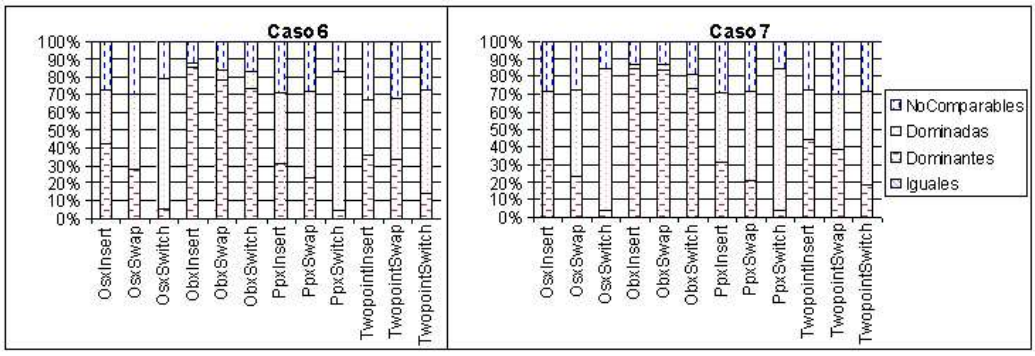

Figura 42: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo MOGA casos 6 y 7

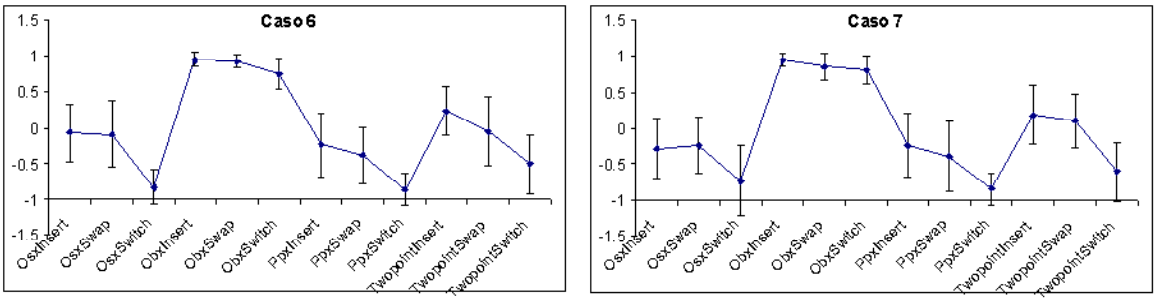

Figura 43: Criterios PD para el caso 75x20. Algoritmo MOGA casos 6 y 7

el mayor porcentaje de soluciones no-dominadas, pero existen algunos en que Swap es mejor. Lo que parece ser definitivo es que la mutación Switch no es particularmente útil para estos casos; ya que sin importar con qué operador de cruzamiento se utilice, ni con qué algoritmo, es la que genera el menor porcentaje de soluciones no-dominadas (por tanto el mayor de dominadas).

Además, los criterios que aplicamos en el caso de 9x5 no se pueden utilizar en este caso porque no conocemos $\mathbf{PF}_{true}$. Los únicos criterios que se pueden utilizar son ONVG y SS. Sin embargo, ONVG sólo nos dice cuántas respuestas no-dominadas generó el algoritmo, pero no dice nada de la calidad de éstas; y el valor 'ideal' para SS

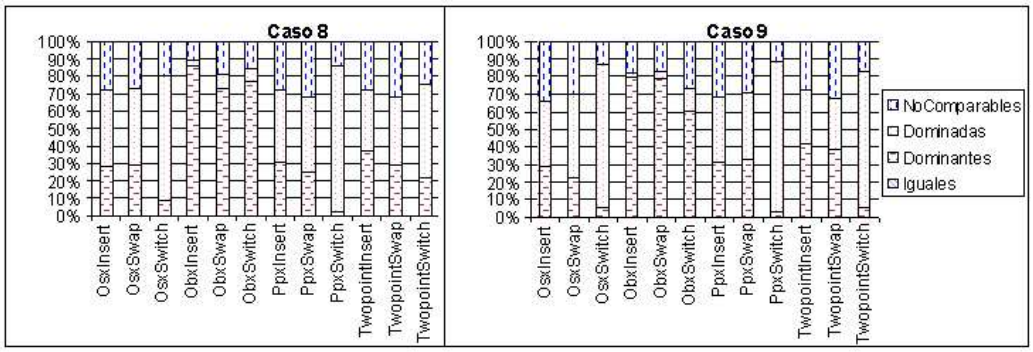

Figura 44: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo MOGA casos 8 y 9

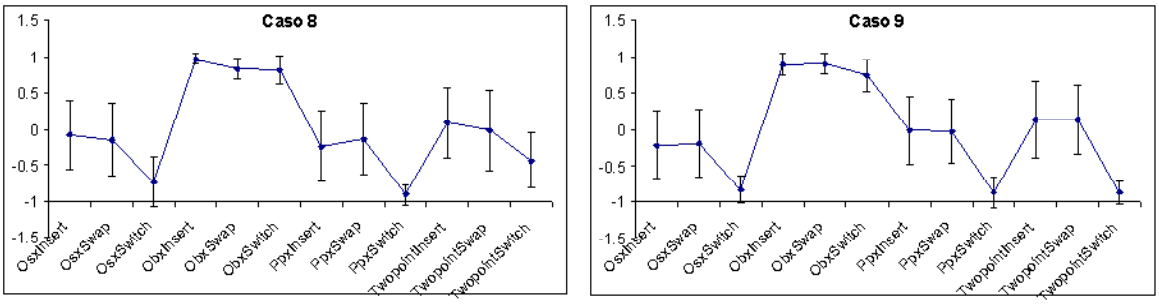

Figura 45: Criterios PD para el caso 75x20. Algoritmo MOGA casos 8 y 9

es 0 solamente cuando se trata de funciones continuas (cuando es importante que las soluciones se encuentren equidistantes unas de otras), en nuestro caso SS no aporta ninguna información porque no conocemos SS de $\mathbf{PF}_{true}$. Para mostrar que estos criterios no nos ayudan está la figura 46, donde se observa que para el caso 9 de MOGA el mejor algoritmo es TwopointInsert lo cual, según la figura 45 es falso. La figura 47 muestra el criterio SS para el mismo caso, en esta figura se ve que el mejor (si consideramos que el mejor es el que se acerca más a 0) es OsxSwitch, sin embargo, por la misma figura 45 es falso. Lo mismo ocurre con NSGA-II y SPEA2.

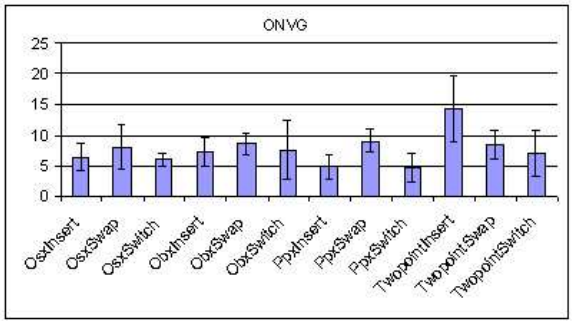

Figura 46: Criterio ONVG para el caso 9 de MOGA

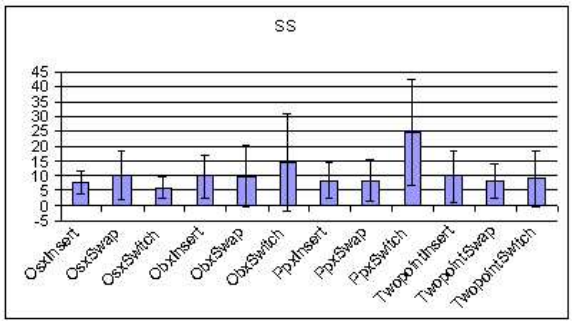

Figura 47: Criterio SS para el caso 9 de MOGA

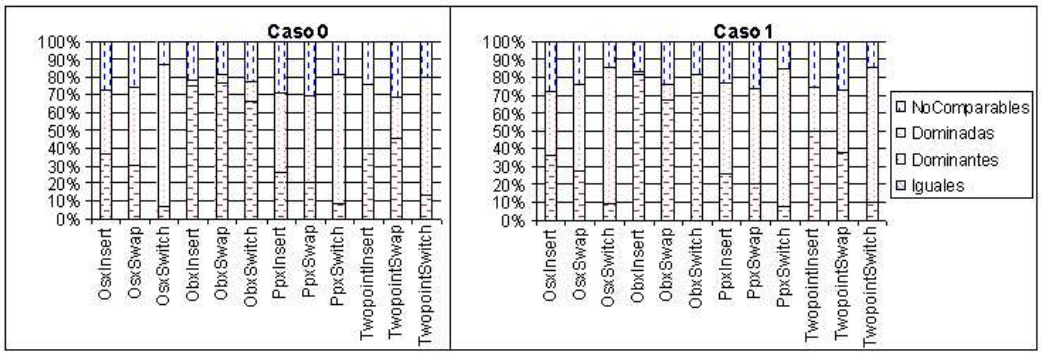

Figura 48: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo NSGA-II casos 0 y 1

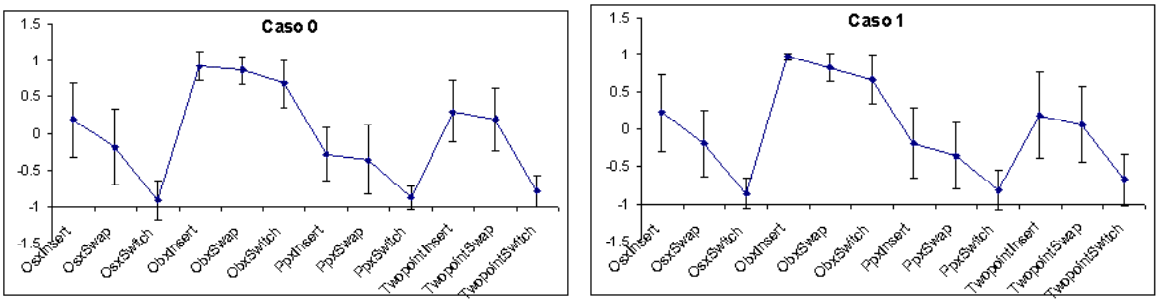

Figura 49: Criterios PD para el caso 75x20. Algoritmo NSGA-II casos 0 y 1

El operador OBX funciona como se explicó en §III.4.1. Lo que hace es mantener pequeñas porciones de la permutación actual (bloques), mientras crea bloques nuevos a lo largo de todo el cromosoma. Este comportamiento parece beneficiar los bloques útiles mientras que desecha aquellos bloques que no contribuyen a mejorar el desempeño. Por el contrario, el operador PPX es el que más tendencia tiene a destruir los bloques que conforman el cromosoma y esa podría ser la razón de su pobre desempeño. El operador OSX conserva el bloque que está al inicio del cromosoma mientras que TWOPOINT conserva los bloques del inicio y del final del cromosoma; su desempeño es similar al de OSX ya que el bloque inicial (loci 1 a s_1) es el mismo para ambos casos.

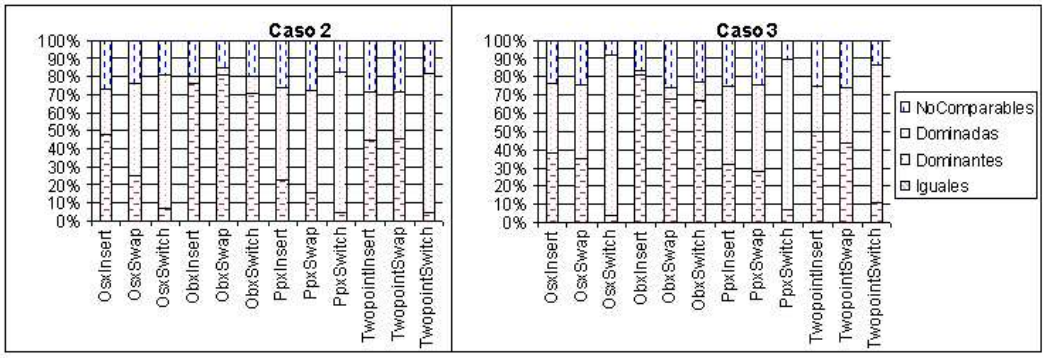

Figura 50: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo NSGA-II casos 2 y 3

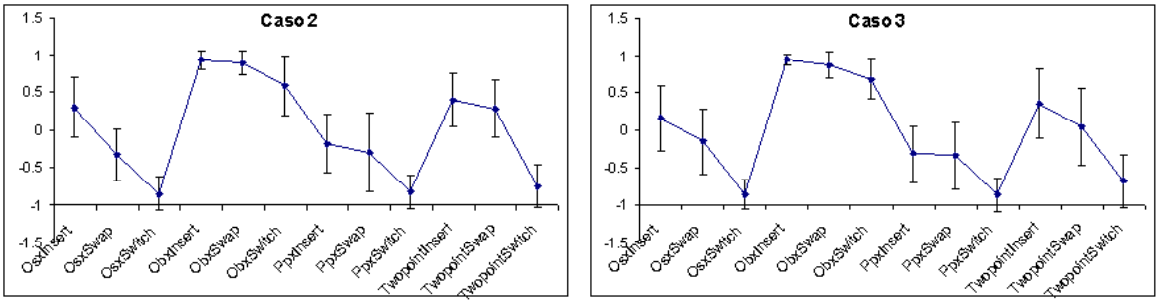

Figura 51: Criterios PD para el caso 75x20. Algoritmo NSGA-II casos 2 y 3

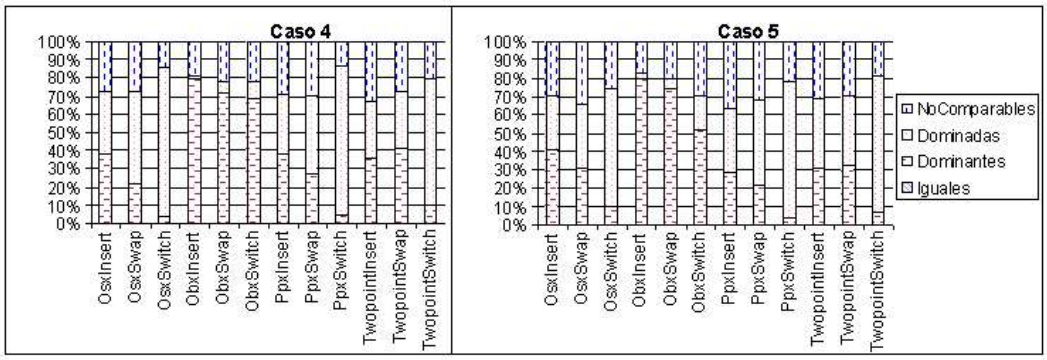

Figura 52: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo NSGA-II casos 4 y 5

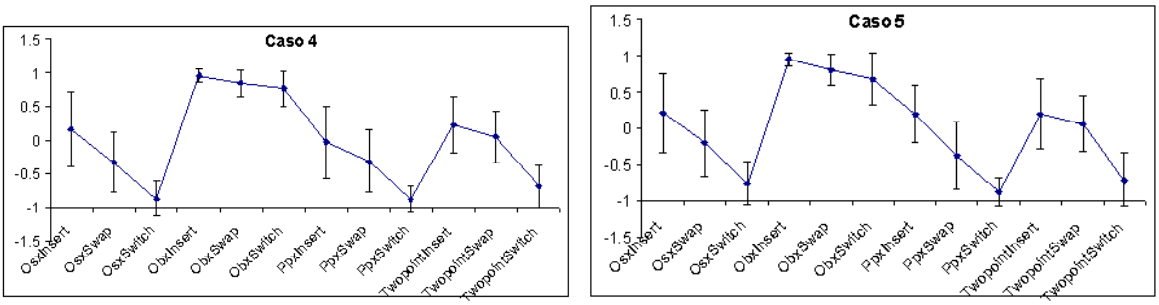

Figura 53: Criterios PD para el caso 75x20. Algoritmo NSGA-II casos 4 y 5

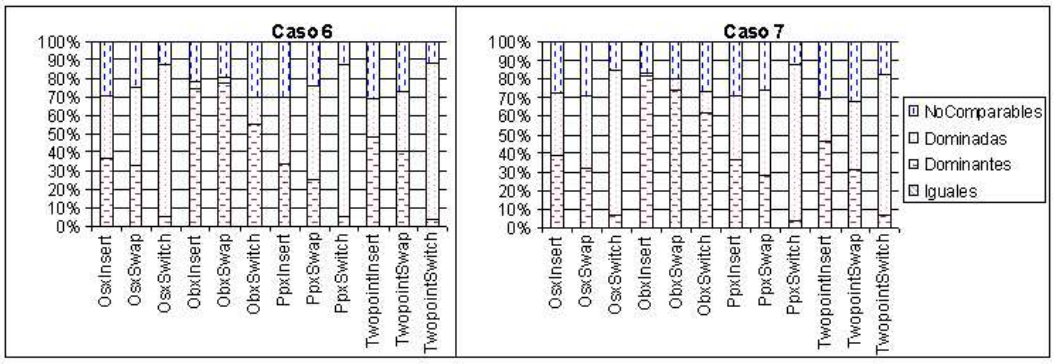

Figura 54: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo NSGA-II casos 6 y 7

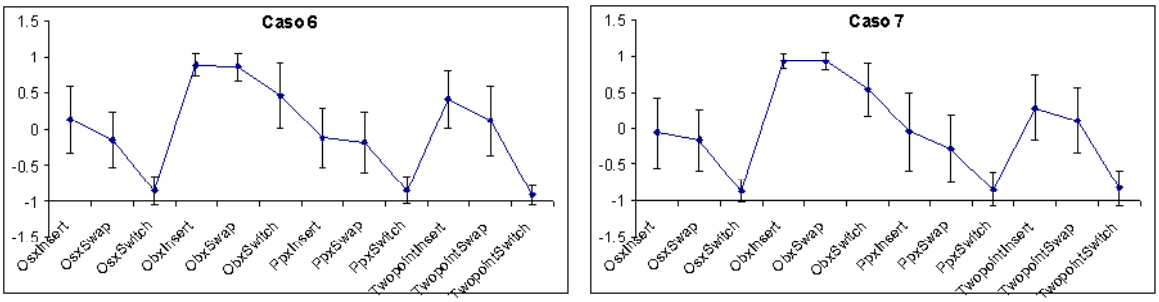

Figura 55: Criterios PD para el caso 75x20. Algoritmo NSGA-II casos 6 y 7

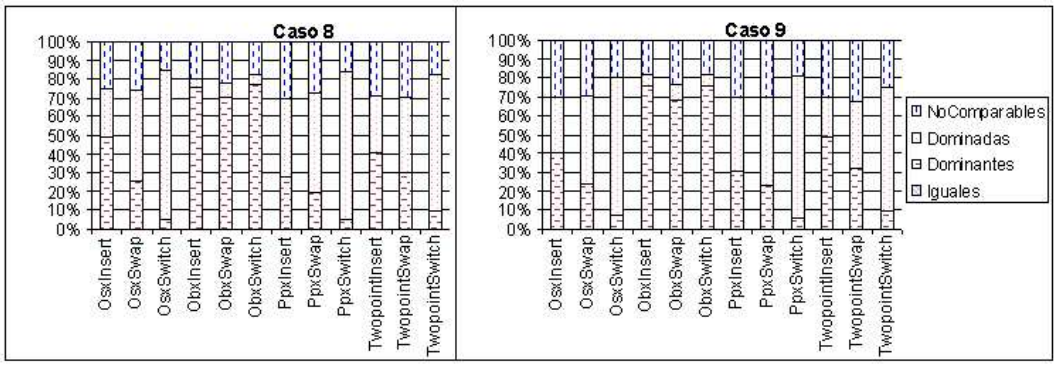

Figura 56: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo NSGA-II casos 8 y 9

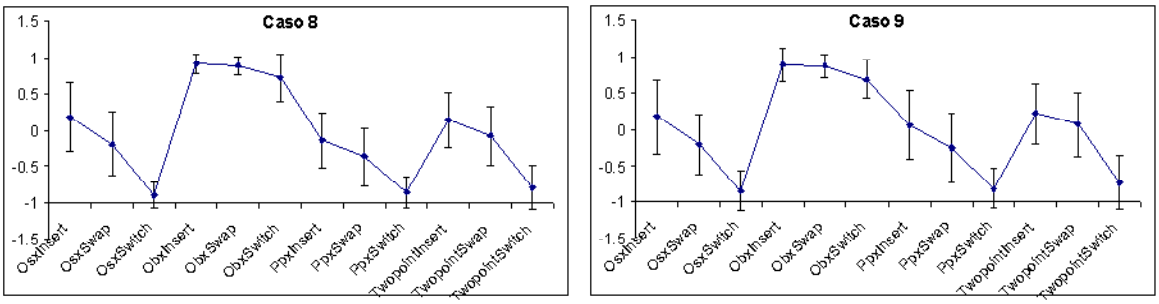

Figura 57: Criterios PD para el caso 75x20. Algoritmo NSGA-II casos 8 y 9

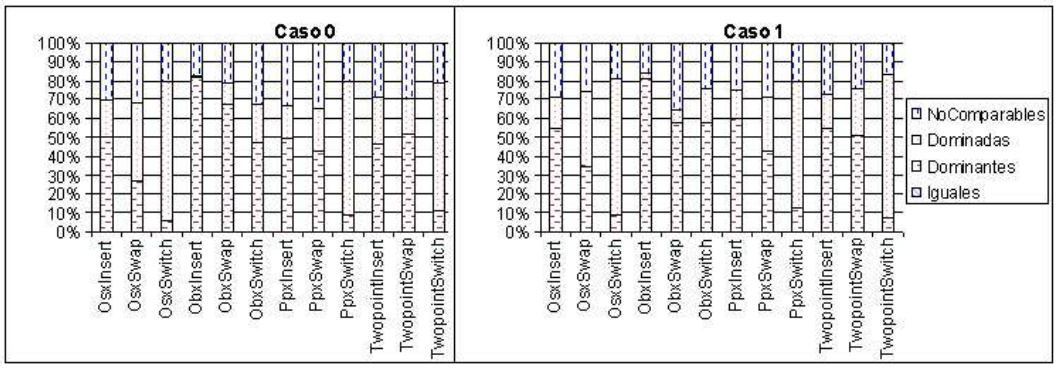

Figura 58: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo SPEA2 casos 0 y 1

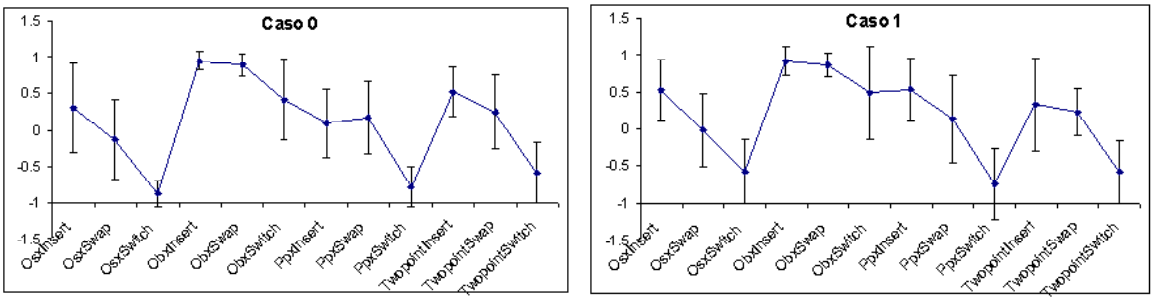

Figura 59: Criterios PD para el caso 75x20. Algoritmo SPEA2 casos 0 y 1

Los datos con los que estamos trabajando provienen de un algoritmo genético multiobjetivo. En van Veldhuizen (1999) se demuestra que estos datos no siguen una distribución normal. Por lo tanto, para verificar que las muestras sean estadísticamente diferentes se aplicó la prueba de Kruskal-Wallis (Devore, 1998; Langley, 1971). Esta prueba no requiere que las muestras tengan una distribución normal, simplemente que tengan la misma distribución continua; lo cual se cumple para nuestro caso como se indica en van Veldhuizen y Lamont (2000). La prueba clasifica las n observaciones desde la 1 a la n. Cuando la hipótesis nula $H_0 : \mu_1 = \mu_2 = \cdots = \mu_n$ es verdadera, las n muestras son estadísticamente similares. Si H_0 es falsa las muestras son estadísticamente

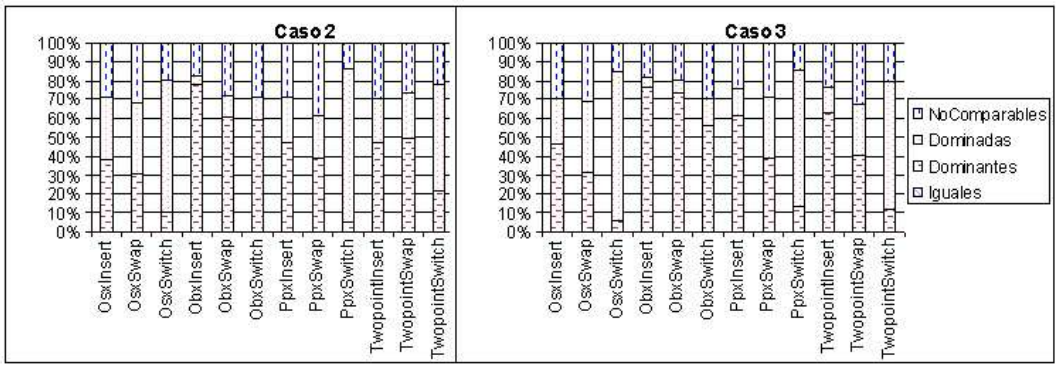

Figura 60: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo SPEA2 casos 2 y 3

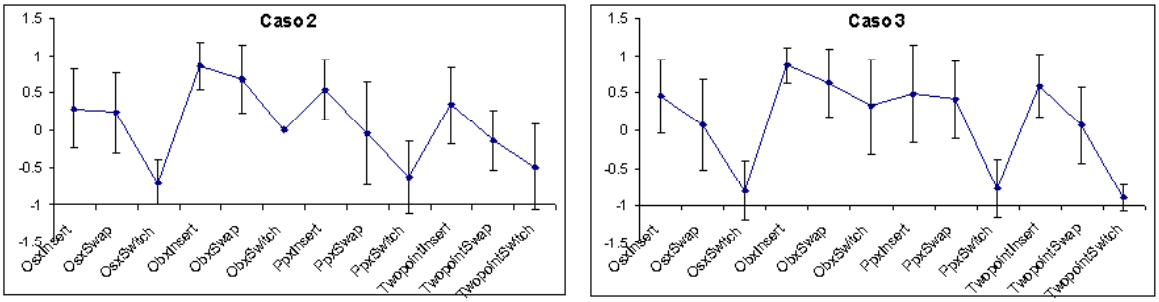

Figura 61: Criterios PD para el caso 75x20. Algoritmo SPEA2 casos 2 y 3

diferentes. La prueba se basa en aplicar el estadístico de prueba:

$$K = \frac{12}{n(n+1)} \sum_{i=1}^{I} \frac{R_i^2}{J_i} - 3(n+1) \tag{19}$$

donde I es el número de muestras, J_i es el número de observaciones en la muestra i y R_i^2 es la suma de los rangos de la muestra i al cuadrado.

H_0 es rechazado con un nivel α si $K \geq \chi^2_{\alpha,I-1}$.

En nuestro caso $I = 12$, $J = 50$ (12 algoritmos y 50 observaciones por algoritmo). La tabla X muestra los valores del estadístico de prueba K obtenidos por cada algoritmo y por cada caso. Para el nivel de confianza 0.01, $\chi^2_{.01,11} = 24.73$; esto significa que todas las muestras rechazan H_0. Por lo tanto, existe evidencia de que para cada caso y cada

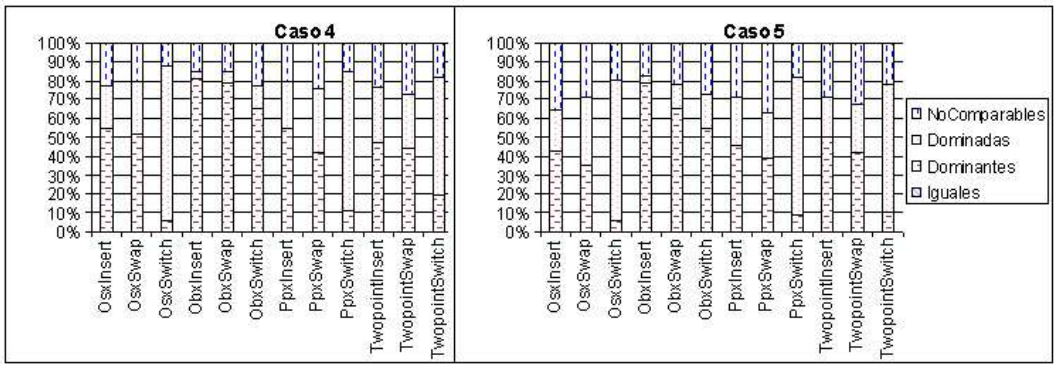

Figura 62: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo SPEA2 casos 4 y 5

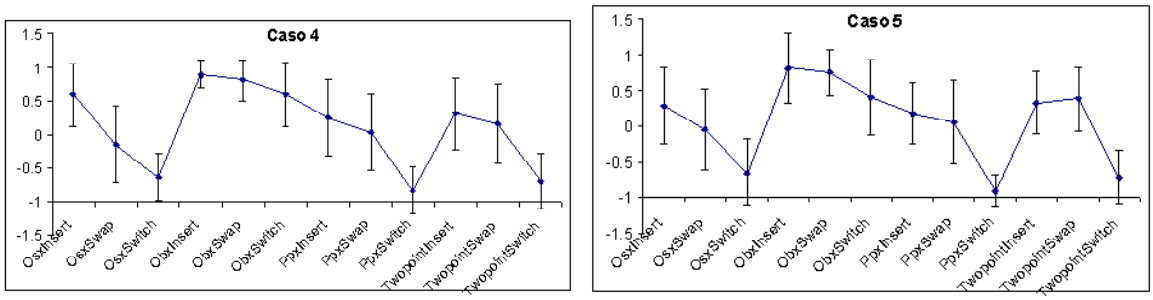

Figura 63: Criterios PD para el caso 75x20. Algoritmo SPEA2 casos 4 y 5

algoritmo las muestras son estadísticamente diferentes. Esto quiere decir que el criterio PD depende del tipo de operadores de cruzamiento y mutación que se utilicen.

Si la prueba de Kruskal-Wallis nos dice que los datos son estadísticamente diferentes, esto significa que para algún par de muestras $\mu_i \neq \mu_j$; pero no nos dice cuál (o cuáles) par (o pares) en particular. Para conocer esa información aplicamos la prueba de suma de rangos de Wilcoxon (Langley, 1971).

La prueba de suma de rangos de Wilcoxon compara dos muestras para saber si son estadísticamente similares. Verifica la hipótesis nula $H_0 : \mu_1 = \mu_2$ contra la prueba alternativa $H_a : \mu_1 \neq \mu_2$. Cuando el número de observaciones de ambas muestras es

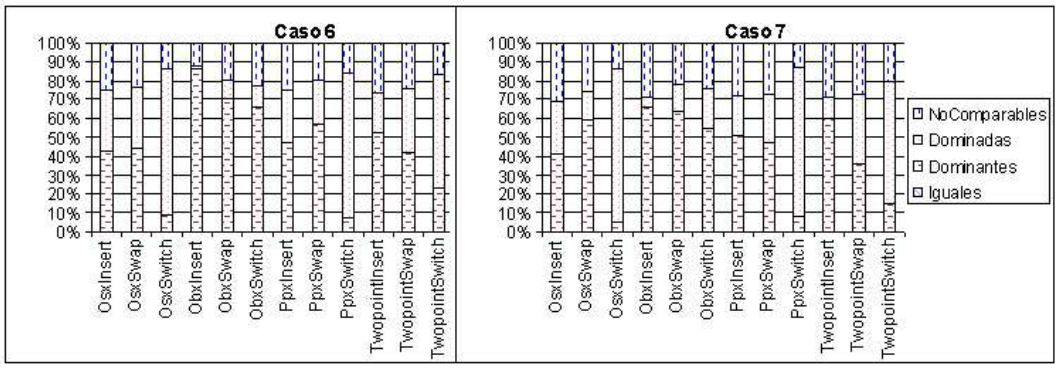

Figura 64: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo SPEA2 casos 6 y 7

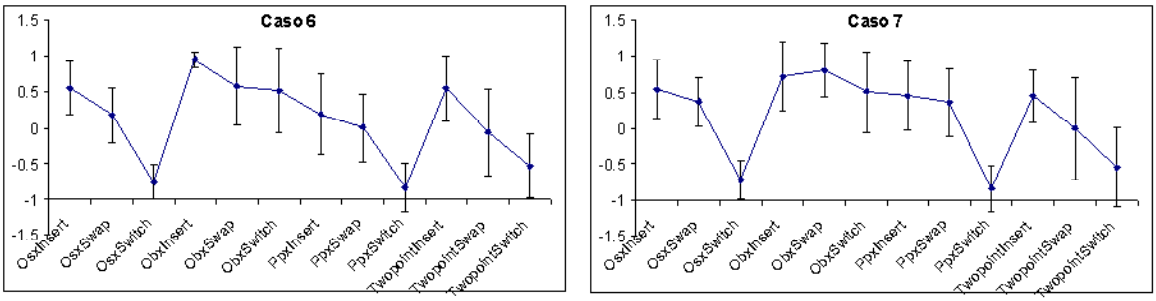

Figura 65: Criterios PD para el caso 75x20. Algoritmo SPEA2 casos 6 y 7

mayor que 8, la prueba de suma de rangos de Wilcoxon tiene una aproximación normal dada por:

$$Z = \frac{W - m(m+n+1)/2}{\sqrt{mn(m+n+1)/12}} \tag{20}$$

donde m es el número de observaciones en la muestra 1, n es el número de observaciones en la muestra 2 y W es la suma de rangos de la muestra 1.

Para un nivel de confianza α, H_0 se rechaza si $Z \geq |z_{\alpha/2}|$. Donde $z_{\alpha/2}$ es el valor de z para el cual el área bajo la curva de una distribución normal con media cero y desviación estándar 1 es igual a $\alpha/2$.

La figura 68 muestra los resultados de aplicar esta prueba con un nivel de confianza

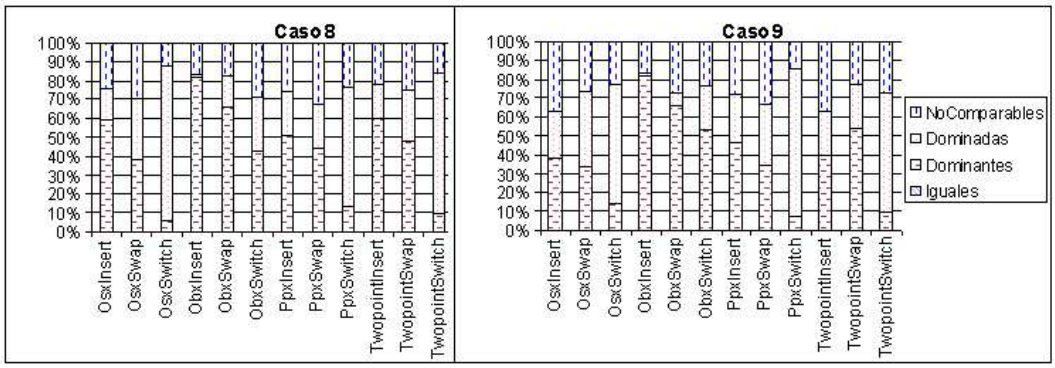

Figura 66: Relaciones $m^j_{\mathbf{A}_i}$, $i \in \{1, \cdots, 12\}, j \in \{1, \cdots, 4\}$ para el problema de flowshop de 75 tareas y 20 máquinas. Algoritmo SPEA2 casos 8 y 9

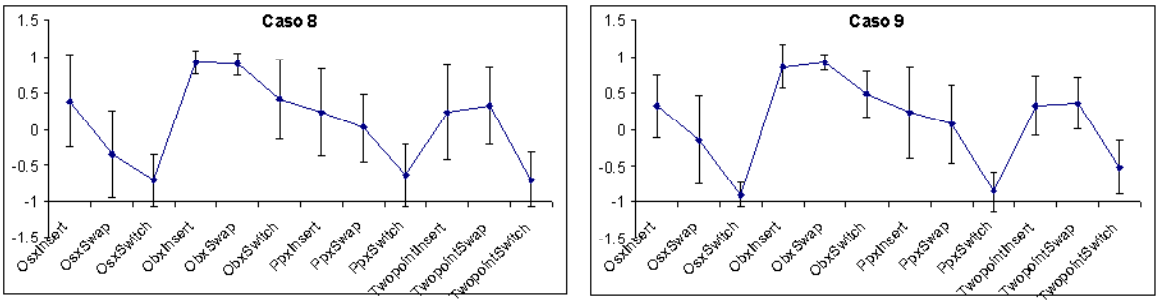

Figura 67: Criterios PD para el caso 75x20. Algoritmo SPEA2 casos 8 y 9

0.01 a los datos generados por el caso 5 con el algoritmo MOGA. Cada intersección representa la comparación entre el algoritmo de la fila y el algoritmo del renglón correspondientes. Un símbolo '=' significa que los algoritmos son estadísticamente iguales y un símbolo 'X' signifca que son diferentes. Se puede ver que la tabla es una matriz simétrica, lo cual tiene sentido porque si ObxInsert contra ObxSwap son diferentes también lo serán ObxSwap contra ObxInsert; es decir, no importa el orden en que se comparen las muestras.

Así, por ejemplo, en la intersección de ObxInsert con ObxSwap aparece un símbolo 'X', por lo tanto son estadísticamente distintas. De hecho, ObxInsert es diferente al resto

Tabla X: Valores del estadístico de prueba K obtenidos por los diferentes casos y por los diferentes algoritmos

caso	MOGA	NSGA-II	SPEA2
caso 0	457.1	455.375	368.897
caso 1	449.85	432.536	374.365
caso 2	445.151	474.717	350.709
caso 3	461.678	462.442	356.091
caso 4	428.599	442.27	338.372
caso 5	445.953	435.326	378.605
caso 6	457.546	441.606	366.842
caso 7	466.023	445.209	376.923
caso 8	421.879	469.647	376.424
caso 9	460.77	431.041	352.643

de las combinaciones porque en su correspondiente fila sólo aparecen 'X'. Sin embargo, ObxSwap es estadísticamente similar a ObxSwitch (hay un '=' en la intesección). El mismo comportamiento se puede observar con los datos generados por NSGA-II para el caso 5 en la figura 69 y también para los datos generados por SPEA2 para el mismo caso 5 en la figura 70.

Estas pruebas estadísticas nos aportan evidencia de que el cruzamiento Obx junto con la mutación Insert es la combinación que genera el mayor porcentaje relativo de soluciones no-dominadas. Por esa razón y como no conocemos $\mathbf{PF}_{true}$, decimos que ObxInsert es la mejor elección para este conjunto de casos.

	OsxInsert	OsxSwap	OsxSwitch	ObxInsert	ObxSwap	ObxSwitch	PpxInsert	PpxSwap	PpxSwitch	TwopointInsert	TwopointSwap	TwopointSwitch
OsxInsert		=	X	X	X	X	=	=	X	X	=	X
OsxSwap	=		X	X	X	X	=	=	X	X	=	X
OsxSwitch	X	X		X	X	X	X	X	=	X	X	X
ObxInsert	X	X	X		X	X	X	X	X	X	X	X
ObxSwap	X	X	X	X		=	X	X	X	X	X	X
ObxSwitch	X	X	X	X	=		X	X	X	X	X	X
PpxInsert	=	=	X	X	X	X		=	X	X	=	X
PpxSwap	=	=	X	X	X	X	=		X	X	=	X
PpxSwitch	X	X	=	X	X	X	X	X		X	X	X
TwopointInsert	X	X	X	X	X	X	X	X	X		X	X
TwopointSwap	=	=	X	X	X	X	=	=	X	X		X
TwopointSwitch	X	X	X	X	X	X	X	X	X	X	X	

Figura 68: Prueba de Wilcoxon para los resultados generados con MOGA. Caso 5

	OsxInsert	OsxSwap	OsxSwitch	ObxInsert	ObxSwap	ObxSwitch	PpxInsert	PpxSwap	PpxSwitch	TwopointInsert	TwopointSwap	TwopointSwitch
OsxInsert		=	X	X	X	X	X	X	X	=	=	X
OsxSwap	=		X	X	X	X	=	X	X	=	=	X
OsxSwitch	X	X		X	X	X	X	X	=	X	X	=
ObxInsert	X	X	X		X	X	X	X	X	X	X	X
ObxSwap	X	X	X	X		=	X	X	X	X	X	X
ObxSwitch	X	X	X	X	=		X	X	X	X	X	X
PpxInsert	X	=	X	X	X	X		X	X	X	=	X
PpxSwap	X	X	X	X	X	X	X		X	X	X	=
PpxSwitch	X	X	=	X	X	X	X	X		X	X	X
TwopointInsert	=	=	X	X	X	X	X	X	X		=	X
TwopointSwap	=	=	X	X	X	X	=	X	X	=		X
TwopointSwitch	X	X	=	X	X	X	X	=	X	X	X	

Figura 69: Prueba de Wilcoxon para los resultados generados con NSGA-II. Caso 5

	OsxInsert	OsxSwap	OsxSwitch	ObxInsert	ObxSwap	ObxSwitch	PpxInsert	PpxSwap	PpxSwitch	TwopointInsert	TwopointSwap	TwopointSwitch
OsxInsert		=	X	X	X	X	=	=	X	X	=	X
OsxSwap	=		X	X	X	X	X	=	X	X	=	X
OsxSwitch	X	X		X	X	X	X	X	=	X	X	=
ObxInsert	X	X	X		X	X	X	X	X	X	X	X
ObxSwap	X	X	X	X		=	X	X	X	X	X	X
ObxSwitch	X	X	X	X	=		=	X	X	=	X	X
PpxInsert	=	X	X	X	X	=		X	X	=	X	X
PpxSwap	=	=	X	X	X	X	X		X	X	=	X
PpxSwitch	X	X	=	X	X	X	X	X		X	X	=
TwopointInsert	X	X	X	X	X	=	=	X	X		X	X
TwopointSwap	=	=	X	X	X	X	X	=	X	X		X
TwopointSwitch	X	X	=	X	X	X	X	X	=	X	X	

Figura 70: Prueba de Wilcoxon para los resultados generados con SPEA2. Caso 5

Capítulo V

Conclusiones y Trabajo Futuro

V.1 Resumen

En este trabajo se presentó un análisis experimental de los operadores genéticos de cruzamiento: OSX, OBX, PPX y Twopoint; junto con los operadores de mutación: Insert, Swap y Switch. Se combinó cada uno de los operadores de cruzamiento con cada uno de los operadores de mutación. Se probó cada combinación con tres algoritmos evolutivos multi-objetivo: MOGA, NSGA-II y SPEA2, en dos conjuntos de casos del problema de flujo de tareas multi-objetivo. En el primer conjunto el tamaño de cada caso fue de nueve (9) tareas y cinco (5) máquinas (9x5) y en el segundo conjunto de 75 tareas y 20 máquinas (75x20). Para cada elemento de cada conjunto del problema de flujo de tareas multi-objetivo se hicieron 50 corridas. Cada conjunto de casos contenía 10 elementos. Se buscó la combinación que generara las mejores soluciones en términos de relaciones de dominancia.

Para cada caso del conjunto de 9x5 se encontró $\mathbf{PF}_{true}$ por medio de enumeración completa de las soluciones. Se estudiaron los criterios: ONVG, GD, SS, MPFE, ER, PC y PD para saber qué algoritmo tiene soluciones de mejor calidad.

Para cada caso de 75x20 sólo se aplicó el criterio PD y el criterio ONVG.

Se utilizaron las pruebas estadísticas no paramétricas Kruskal-Wallis y suma de rangos de Wilcoxon para validar los resultados.

V.2 Conclusiones

- El criterio ONVG no muestra información relevante en cuanto a calidad de las soluciones.

- Los promedios de los criterios GD, MPFE, ER y PC no muestran consistencia. Por ejemplo, para el caso 2 es mejor TwopointSwap según el criterio MPFE pero según el criterio PC es mejor ObxInsert. Esto se muestra en §IV.1.3.

- Aunque el valor promedio de SS sea cercano al valor de SS de $\mathbf{PF}_{true}$, esto no siempre es un indicativo de que este criterio sea útil. Y tampoco muestra consistencia a través de los diferentes casos de tamaño 9x5 estudiados (§IV.1.3).

- El promedio del criterio PD mostró que ObxInsert es la combinación que genera el mayor porcentaje relativo de soluciones no-dominadas en la mayoría de los casos. Esto se muestra en §IV.1.3 para los casos de 9x5 y en §IV.2.3 para los casos de 75x20.

- Las combinaciones que en promedio muestran peor desempeño en todos los casos son las que incluyen la mutación Switch.

- Se encontró que con el algoritmo MOGA, la combinación ObxInsert es superior, en términos de las relaciones de dominancia, al resto de las combinaciones (Brizuela y Aceves, 2003).

- Se verificó que la combinación ObxInsert sigue siendo superior a pesar de cambiar el algoritmo y utilizar NSGA-II (Aceves y Brizuela, 2003).

- Al aplicar la prueba de Kruskal-Wallis se muestra que existe una diferencia estadística (con un nivel de confianza de 0.01) entre las 12 combinaciones.

- Según la prueba de suma de rangos de Wilcoxon, al nivel de confianza 0.01, para los casos de 75x20 la combinación ObxInsert es diferente al resto de las combinaciones en todos los casos y sólo es similar a ObxSwap el 40% de los casos. Pero ObxSwap es estadísticamente similar a otras combinaciones.

- La mutación Switch con las combinaciones Osx, Ppx y Twopoint es la que muestra peor desempeño en todos los casos, tanto de 9x5 como de 75x20. Además, dichas combinaciones resultaron estadísticamente similares en todos los casos.

Después de analizar entre un conjunto de operadores genéticos encontramos que existe una combinación, ObxInsert, que genera mejores soluciones para casos específicos del problema de flujo de tareas, por lo tanto el objetivo principal de esta tesis ha sido cubierto.

V.3 Aportaciones

- Se demostró experimentalmente que para un conjunto de casos específicos del problema de flujo de tareas (versión permutación) el operador de cruzamiento Obx y el operador de mutación Insert genera el mayor porcentaje relativo de soluciones no-dominadas.

- Se presentaron dos trabajos en congreso:

  - Brizuela y Aceves (2003) en "Second International conference on Evolutionary Multi-criterion Optimization" (EMO'03) celebrado en Faro, Portugal.

  - Aceves y Brizuela (2003) en el primer "Congreso Mexicano de Computación Evolutiva" (COMCEV'2003) realizado en Guanajuato, México.

- Se desarrolló una infraestructura de programas donde se pueden realizar experimentos con diferentes conjuntos de soluciones multi-objetivo. Dichos programas permiten calcular los criterios ONVG, GD, SS, PC, MPFE, ER, y PD sobre los conjuntos de soluciones. También aplican la prueba de Kruskal-Wallis y la prueba de suma de rangos de Wilcoxon sobre los datos generados para calcular el criterio PD. Esto ayudaría a realizar estudios similares para otros problemas de calendarización.

V.4 Trabajo Futuro

- Investigar por qué ObxInsert es la combinación de operadores genéticos que genera la mayor cantidad relativa de soluciones no-dominadas.
  - Definir las vecindades que crea el operador Insert.
  - Encontrar alguna relación entre el cruzamiento Obx y la generación de buenas soluciones.

Bibliografía

Aceves, R. y C. A. Brizuela 2003. "Análisis experimental de operadores genéticos en NSGA-II para un problema de calendarización multi-objetivo". En: Botello, S., A. Hernández, y C. Coello, editores, "Memorias del Primer Congreso Mexicano de Computación Evolutiva, COMCEV'03", Guanajuato, México. CIMAT. 55–66 p.

Armentano, V. A. y D. P. Ronconi 1999. "Tabu search for total tardiness minimization in flowshop scheduling problems". Computers & Operations Research, 26:219–235 p.

Bagchi, T. P. 1999. "Multiobjective scheduling by genetic algorithms". Kluwer Academic Publishers, Boston, Massachusetts. 376 pp.

Bäck, T. 1996. "Evolutionary algorithms in theory and practice". Oxford University Press, New York. 328 pp.

Bierwirth, C., D. C. Mattfeld, y H. Kopfer 1996. "On permutation representations for scheduling problems". En: Voigt, H.-M., W. Ebeling, I. Rechenberg, y H.-P. Schwefel, editores, "Parallel Problem Solving from Nature – PPSN IV", Berlin, Germany, volume 1141 of LNCS. Springer. 310–318 p.

Brizuela, C. A. y R. Aceves 2003. "Experimental genetic operators analysis for the multi-objective permutation flowshop". En: Fonseca, C. M., P. J. Fleming, E. Zitzler, K. Deb, y L. Thiele, editores, "Evolutionary Multi-Criterion Optimization, Second International Conference, EMO'03", Faro, Portugal, volume 2632 of LNCS. Berlin:Springer-Verlag. 578–592 p.

Brizuela, C. A., Y. Zhao, y N. Sannomiya 2001. "Multi-objective flowshop: Preliminary results". En: Zitzler, E., K. Deb, L. Thiele, C. A. Coello, y D. Corne, editores, "Evolutionary Multi-Criterion Optimization, First International Conference, EMO'01", Zurich, Switzerland, volume 1993 of LNCS. Berlin:Springer-Verlag. 443–457 p.

Bibliografía (Continuación)

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

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

Deb, K., S. Agrawal, A. Pratap, y T. Meyarivan 2000. "A fast elitist non-dominated sorting algorithm for multi-objective optimization: NSGA-II". En: Schoenauer, M., K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, y H.-P. Schwefel, editores, "Parallel Problem Solving from Nature – PPSN VI", Paris, France, volume 1917 of LNCS. Springer-Verlag. 849–858 p.

Devore, J. L. 1998. "Probabilidad y estadística para ingeniería y ciencias". International Thompson Editores, México. 712 pp.

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

Gamelin, T. W. y R. E. Greene 1983. "Introduction to topology". Saunders College Publishing, Mexico City. 196 pp.

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

Gen, M. y R. Cheng 1997. "Genetic algorithms & engineering design". John Wiley & Sons, New York. 432 pp.

Gen, M. y R. Cheng 2000. "Genetic algorithms & engineering optimization". John Wiley & Sons, New York. 512 pp.

Bibliografía (Continuación)

Goicoechea, A., D. R. Hansen, y L. Duckstein 1982. "Multiobjective analysis with engineering and business applications". John Wiley and Sons, New York. 519 pp.

Goldberg, D. 1989. "Genetic algorithms in search, optimization and machine learning". Addison-Wesley, Reading, MA. 432 pp.

Hansen, M. P. y A. Jaszkiewicz 1998. "Evaluating the quality aproximations to the nondominated set". Reporte Técnico IMM-REP-1998-7, Technical University of Denmark. 31 pp.

Ho, J. C. y Y. L. Chang 1991. "A new heuristic for the n-job m-machine flowshop problem". European Journal of Operational Research, 52:194–202 p.

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

Horn, J., N. Nafpliotis, y D. E. Goldberg 1994. "A niched pareto genetic algorithm for multiobjective optimization". En: Michalewicz, Z., editor, "Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, (ICEC'94)", Orlando, USA, volume 1. IEEE Service Center. 82–87 p.

Ishibuchi, H. y T. Murata 1998. "Multi-objective genetic local search algorithm and its application to flowshop scheduling". IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews, 28(3):392–403 p.

Bibliografía (Continuación)

Ishibuchi, H., T. Yoshida, y T. Murata 2002. "Balance between genetic search and local search in hybrid evolutionary multi-criterion optimization algorithms". En: Langdon, W. B., E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, y N. Jonoska, editores, "GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference", New York, USA. Morgan Kaufman Publishers. 1301–1308 p.

Ishibuchi, H., T. Yoshida, y T. Murata 2003. "Balance between genetich search and local search in memetic algorithms for multiobjective permutation flowshop scheduling". IEEE Transactions on Evolutionary Computation, 7(2):204–223 p.

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

Knowles, J. 2002. "Local-search and hybrid evolutionary algorithms for pareto optimization". Tesis de Doctorado, University of Reading, Department of Computer Science, Reading, U.K. 120 pp.

Knowles, J. y D. Corne 2002. "On metrics for comparing nondominated sets". En: Fogel, D. B., M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, y M. Shackleton, editores, "Proceedings of the 2002 Congress on Evolutionary Computation", Hawaii, USA. IEEE Neural Network Council (NNC), Institution of Electrical Engineers (IEE), Evolutionary Programming Society (EPS), IEEE Presss. 711–716 p.

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

Langley, R. 1971. "Practical statistics, simply explained". Dover Publications, Inc., New York. 399 pp.

Bibliografía (Continuación)

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

Murata, T., H. Ishibuchi, y H. Tanaka 1996. "Genetic algorithms for flowshop scheduling problems". Computer and Industrial Engineering, 30(40):1061–1071 p.

Osyczka, A. 1984. "Multicriterion optimization in engineering with FORTRAN programs". Ellis Horwood Limited, Chichester, UK. 178 pp.

Pinedo, M. 1995. "Scheduling - theory, algorithms, and systems". Prentice Hall, Englewood Cliffs. 586 pp.

Rudolph, G. 1994. "Convergence analysis of canonical genetic algorithms". IEEE Transactions on Neural Networks, 5(1):96–101 p.

Schott, J. R. 1995. "Fault tolerant design using single and multicriteria genetic algorithm optimization". Tesis de Maestría, Department of Aeronautics and Astronautics. Massachusetts Institute of Technology, Cambridge, Massachusetts. 130 pp.

Srinivas, N. y K. Deb 1994. "Multiobjective optimization using nondominated sorting in genetic algorithms". Evolutionary Computation, 2(3):221–248 p.

Starkweather, T., S. McDaniel, K. Mathias, y D. Whitley 1991. "A comparison of genetic sequencing operators". En: Belew, R. y L. Booker, editores, "Proceedings of the Fourth International Conference on Genetic Algorithms", California, USA. San Mateo: Morgan Kaufmann. 69–76 p.

Steuer, R. E. 1986. "Multiple criteria optimization: Theory, computation, and application". Krieger Publishing Company, Melbourne, Florida. 586 pp.

Bibliografía (Continuación)

Syswerda, G. 1991. "Handbook of genetic algorithms". International Thomson Publishing, capítulo 21: Schedule optimization using genetic algorithms, 332–249 p.

Tagami, T. y T. Kawabe 1998. "Genetic algorithm with a pareto partitioning method for multi-objective flowshop scheduling". En: "Proceedings of the 1998 International Symposium of Nonlinear Theory and its Applications(NOLTA'98)", Crans-Montana, Switzerland. Presses Polytechniques et Universitaires Romandes. 1069–1072 p.

Tagami, T. y T. Kawabe 1999. "Genetic algorithm based on a pareto neighborhood search for multiobjective optimization". En: Yamamura, K., editor, "Proceedings of the 1999 International Symposium of Nonlinear Theory and its Applications (NOLTA'99)", Hawaii, USA. 331–334 p.

Talbi, E. G., M. Rahoual, M. H. Mabed, y C. Dhaenens 2001. "A hybrid evolutionary approach for multicriteria optimization problems: Application to the flow shop". En: Zitzler, E., K. Deb, L. Thiele, C. A. Coello, y D. Corne, editores, "Evolutionary Multi-Criterion Optimization, First International Conference, EMO'01", Zurich, Switzerland, volume 1993 of LNCS. Berlin:Springer-Verlag. 416–428 p.

Ulder, N. L., E. H. Aarts, H.-J. Bandelt, P. J. M. van Laarhoven, y E. Pesch 1991. "Genetic local search algorithm for the traveling salesman problem". En: Schwefel, H.-P. y R. Männer, editores, "Parallel Problem Solving from Nature", Dortmund, Germany, volume 496 of LNCS. Berlin:Springer-Verlag. 109–116 p.

van Veldhuizen, D. A. 1999. "Multiobjective evolutionary algorithms: Classifications, analyses, and new innovations". Tesis de Doctorado, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio. 270 pp.

Bibliografía (Continuación)

van Veldhuizen, D. A. y G. B. Lamont 2000. "On measuring multiobjective evolutionary algorithm performance". En: "Proceedings of the 2000 Congress on Evolutionary Computation. CEC 2000", California, USA. IEEE Service Center. 204–211 p.

Zadeh, L. A. 1963. "Optimality and nonscalar-valued performance criteria". IEEE Transactions on Automatic Control, AC-8(1):59–60 p.

Zitzler, E. 1999. "Evolutionary algorithms for multiobjective optimization: Methods and applications". Tesis de Doctorado, Swiss Federal Institute of Technology (ETH), Zurich, Switzlerand. 132 pp.

Zitzler, E., M. Laumanns, y L. Thiele 2002a. "SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization". En: Giannakoglou, K., D. Tsahalis, J. Periaux, K. Papailiou, y T. Fogarty, editores, "Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems", Athens, Greece. International Center for Numerical Methods in Engineering (Cmine). Proceedings of the EUROGEN2001 Conference, 95–100 p.

Zitzler, E., M. Laumanns, L. Thiele, C. M. Fonseca, y V. Grunert da Fonseca 2002b. "Why quality assesment of multiobjective optimizers is difficult". En: Langdon, W. B., E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, y N. Jonoska, editores, "GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference", New York, USA. Morgan Kaufman Publishers. 666–673 p.

Apéndice A

Ejemplo de un Problema de Flujo de Tareas

El problema de flujo de tareas no es exclusivo de las líneas de producción en serie. Si se reemplaza *artículos* por *tareas* y *máquinas* por *procesos* se abarca cualquier tipo de escenario en el que las tareas deban ser realizadas siguiendo una secuencia predeterminada. Para ilustrar mejor este problema, se puede pensar en una empresa que fabrica *software* que tiene 3 programas en su agenda: un programa para controlar nóminas (nómina), un programa para hacer gráficas en 3D (gráfica) y un programa para simular el clima de una región (simulación). Supongamos que cada programa está dividido en seis procesos, como se ve a continuación,

- **Nómina**

  P1. Entrevistar al cliente (R1).

  P2. Diseñar la base de datos (R2).

  P3. Diseñar la interfaz gráfica (R3).

  P4. Programar y depurar (R4).

  P5. Instalar el *software* en la empresa del cliente (R5).

  P6. Capacitar al personal (R6).

- **Gráfica**

  P1. Analizar el nicho de mercado (R1).

  P2. Estudiar las funciones que serán incluidas (R2).

  P3. Diseñar la interfaz (R3).

P4. Programar y depurar (R4).

P5. Hacer pruebas de corrida y validación de datos (R5).

P6. Escribir manual de usuario y manual técnico (R6).

- **Simulación**

P1. Obtener información histórica del clima de la región (R1).

P2. Analizar la estructura de los datos (R2).

P3. Diseñar el algoritmo de simulación (R3).

P4. Programar y depurar (R4).

P5. Hacer pruebas estadísticas de los resultados de la simulación (R5).

P6. Reportar resultado (R6).

donde cada Ri indica el recurso utilizado para realizar el proceso Pi.

En este tipo de problemas cada proceso de una tarea debe ser hecho uno a la vez, sin embargo, procesos de tareas distintas se pueden hacer al mismo tiempo (siempre y cuando ambos procesos no se ejecuten en la misma máquina y estén disponibles). Esto se puede entender mejor si se imagina que cada proceso anterior es realizado por una persona distinta. Esto es, se tienen seis personas, y cada una hace *un y sólo un proceso* a la vez; pero dos o más personas pueden estar ocupadas *al mismo tiempo*. Por ejemplo, la persona 1 puede ir a entrevistar al cliente para la nómina pero no puede al mismo tiempo analizar el nicho de mercado (para el programa de graficación) ni obtener información (para el programa de simulación). De hecho, se puede ver que la persona 4 (R4) realiza los tres programas, pero no los puede hacer al mismo tiempo. Esto quiere decir que las tareas deben utilizar los mismos recursos, y de aquí es donde surge el problema.

Como se ve, se deben producir tres artículos (tres tareas) en seis etapas (seis procesos u operaciones cada tarea). Además, se tienen que cumplir ciertos tiempos de finalización específicos para cada producto. Después de varias pruebas se logró cronometrar el tiempo de cada proceso y esto se muestra en la Tabla XI, así como su tiempo de

Tabla XI: Tiempos de cada proceso para el problema de Nómina, Gráfica y Simulación (minutos)

Tarea	P1	P2	P3	P4	P5	P6	Fecha Límite (d_i)
Nómina	4	1	9	18	7	15	93
Gráfica	4	4	18	17	3	10	64
Simulación	4	4	8	14	21	10	86

finalización. La empresa quiere terminar cada pedido lo más pronto posible, por lo que surge la pregunta ¿Qué se hace primero: Nómina, Gráfica o Simulación? Si se considera un caso más real, se puede pensar que la empresa quiere minimizar tres cosas: el tiempo de finalización total, el tiempo promedio en que se hacen los productos o el tiempo promedio de retraso. Para tener una idea visual del proceso, podemos graficar cada tarea en el tiempo, y así obtenemos lo que se conoce como diagrama de Gantt. Este problema consta de 3 tareas y 6 procesos, lo que nos da $3! = 6$ formas distintas de realizar la producción. Las figuras 71 a 76 muestran las seis diferentes soluciones.

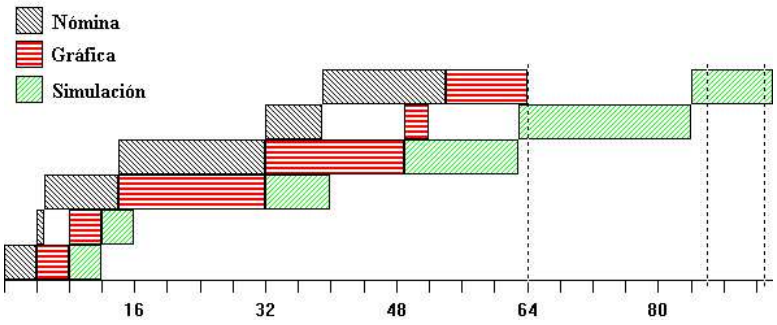

Figura 71: Diagrama de Gantt que muestra los resultados al producir Nómina-Gráfica-Simulación

La mejor forma de producción depende por completo de lo que queremos mejorar. La Tabla XII muestra los resultados de las seis combinaciones, donde el 1 corresponde a Nómina, el 2 corresponde a Gráfica y el 3 corresponde a Simulación. Si lo que nos interesa es terminar en el menor tiempo posible, trataremos de minimizar el tiempo de finalización de la última tarea. En el orden (3,1,2) este tiempo es 86 minutos, siendo el menor de todos. Pero si lo que nos interesa es que las tareas se terminen rápido

Tabla XII: Resultados de las seis diferentes formas de producción (en minutos). C_{max}=Tiempo de finalización de la última tarea. $\overline{F}$=Tiempo promedio de permanencia en el taller. $\overline{T}$=Tiempo promedio de retraso en las tareas

Orden	C_{max}	$\overline{F}$	$\overline{T}$
(1,2,3)	94	212	8
(1,3,2)	87	218	23
(2,1,3)	106	245	20
(2,3,1)	103	247	12
(3,1,2)	86	223	22
(3,2,1)	91	223	7

una a una, es decir, que estén el menor tiempo posible en el taller –en nuestro caso la empresa de *software*– entonces vemos que el orden (1,2,3) es el que tiene el menor tiempo promedio (212 minutos). Más aún, si quisiéramos minimizar el tiempo promedio de retraso, escogeríamos el orden (3,2,1) que tiene un tiempo promedio de 7 minutos. Ésto se puede ver en las figuras 71 a 76.

Como sólo estamos tomando un criterio a la vez, sólo escogemos la combinación que más nos convenga. En el primer caso sería la (3,1,2), en el segundo la (1,2,3) y en el tercero la (3,2,1). Pero, ¿qué pasa si queremos minimizar el tiempo máximo de finalización, el tiempo promedio de finalización *y además* el tiempo promedio de retraso en las tareas? Entonces estamos ante el problema de flujo de tareas *multi-objetivo*.

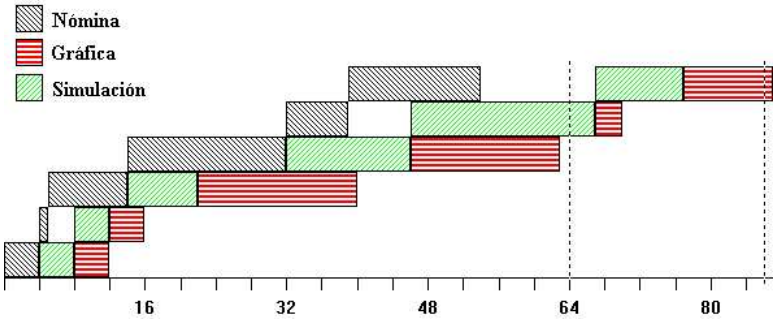

Figura 72: Diagrama de Gantt que muestra los resultados al producir Nómina-Simulación-Gráfica

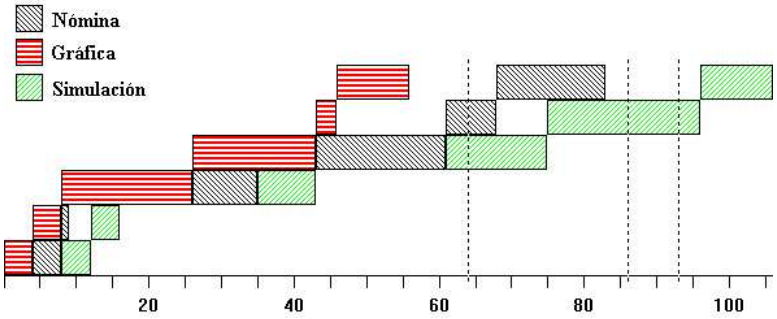

Figura 73: Diagrama de Gantt que muestra los resultados al producir Gráfica-Nómina-Simulación

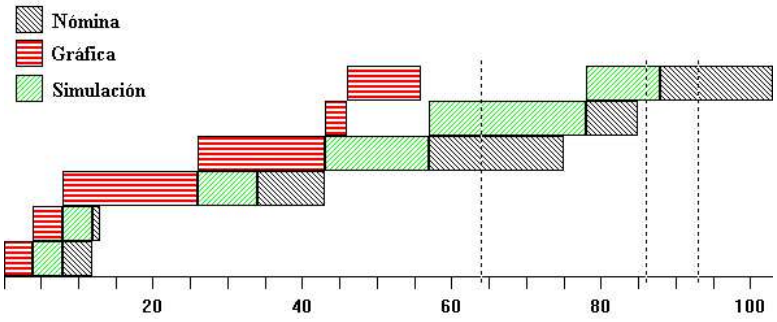

Figura 74: Diagrama de Gantt que muestra los resultados al producir Gráfica-Simulación-Nómina

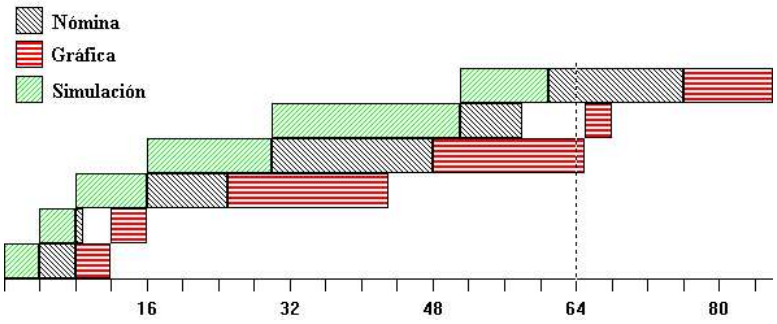

Figura 75: Diagrama de Gantt que muestra los resultados al producir Simulación-Nómina-Gráfica

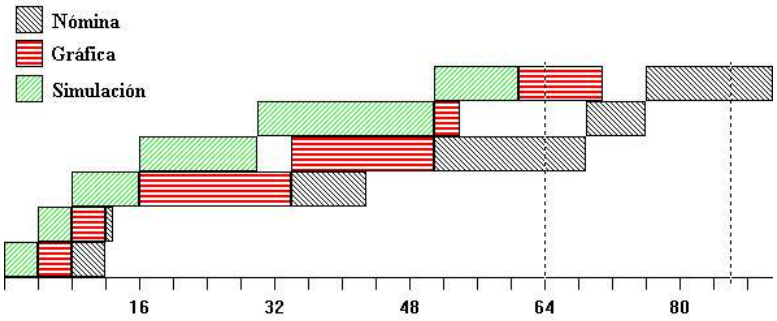

Figura 76: Diagrama de Gantt que muestra los resultados al producir Simulación-Gráfica-Nómina

Apéndice B

Resultados de los Criterios de Comparación

En este apéndice se presentan los resultados de aplicar el criterio PD a cada caso de tamaño 9x5. Estos resulados son dados para las soluciones generadas por los algoritmos MOGA, NSGA-II y SPEA2.

B.1 MOGA

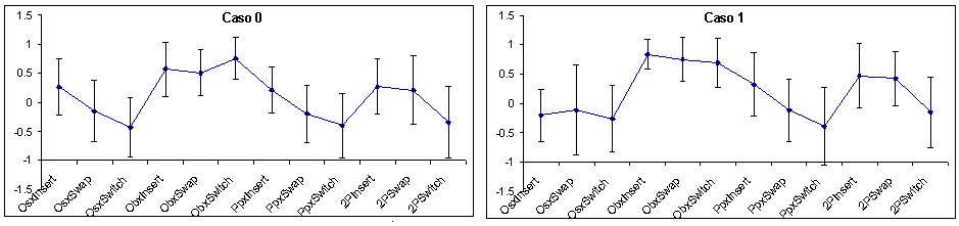

Figura 77: Criterio PD para el algoritmo MOGA. Casos 0 y 1 de 9x5

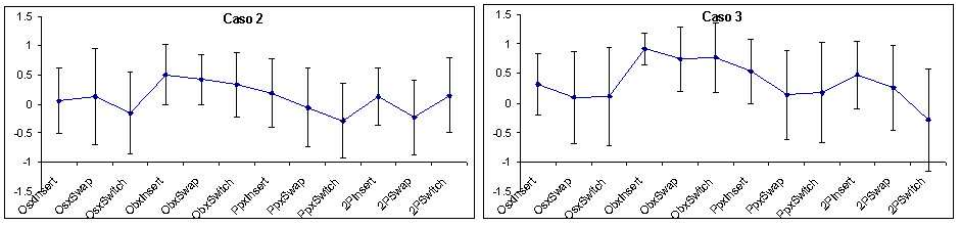

Figura 78: Criterio PD para el algoritmo MOGA. Casos 2 y 3 de 9x5

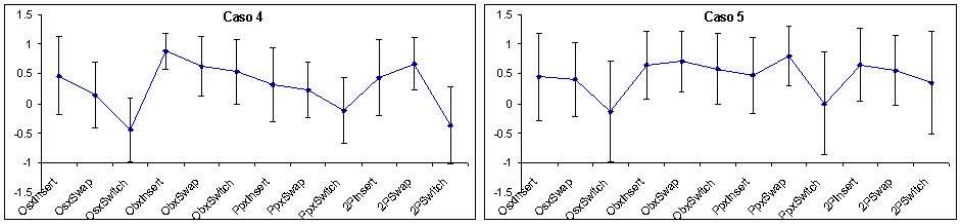

Figura 79: Criterio PD para el algoritmo MOGA. Casos 4 y 5 de 9x5

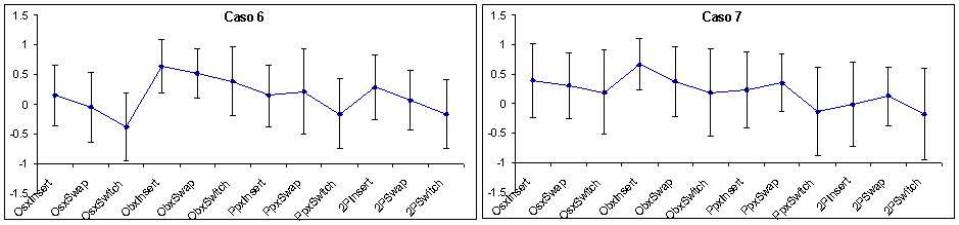

Figura 80: Criterio PD para el algoritmo MOGA. Casos 6 y 7 de 9x5

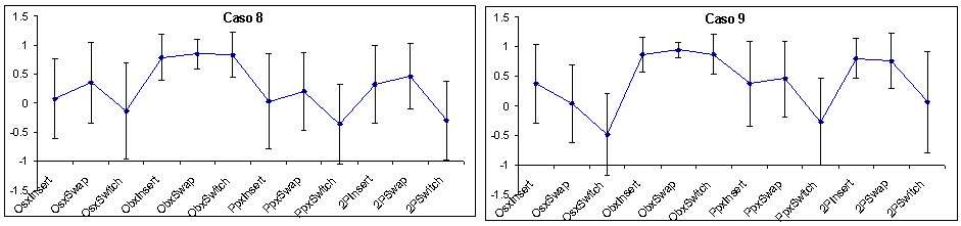

Figura 81: Criterio PD para el algoritmo MOGA. Casos 8 y 9 de 9x5

B.2 NSGA-II

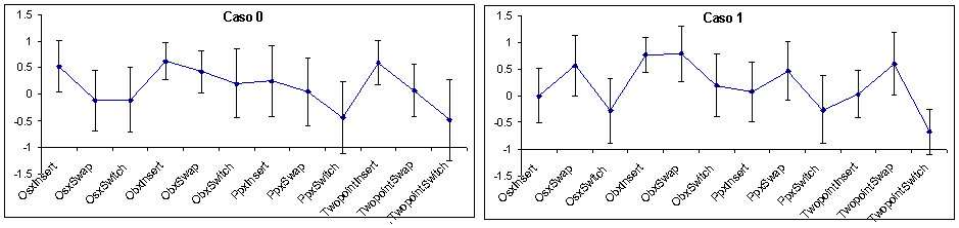

Figura 82: Criterio PD para el algoritmo NSGA-II. Casos 0 y 1 de 9x5

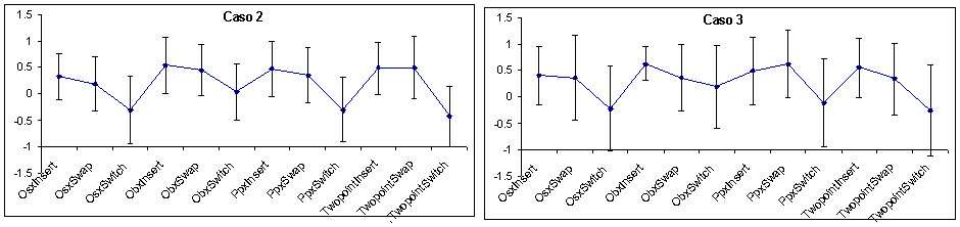

Figura 83: Criterio PD para el algoritmo NSGA-II. Casos 2 y 3 de 9x5

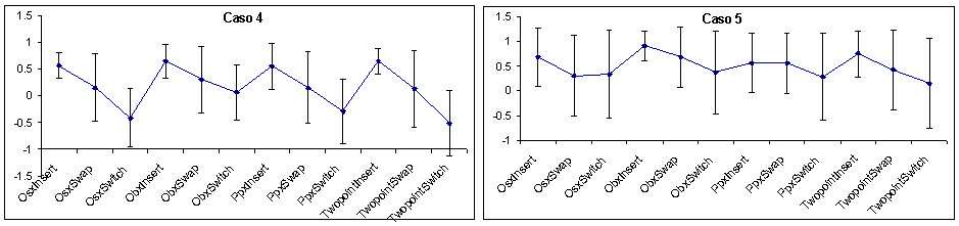

Figura 84: Criterio PD para el algoritmo NSGA-II. Casos 4 y 5 de 9x5

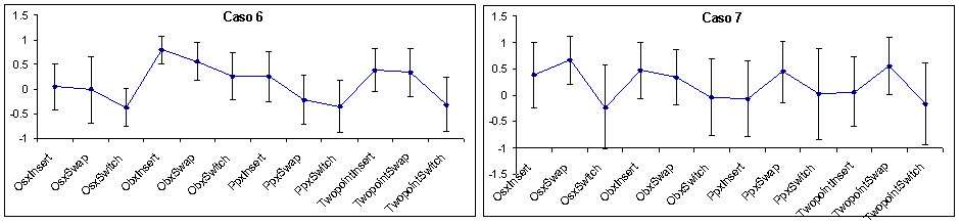

Figura 85: Criterio PD para el algoritmo NSGA-II. Casos 6 y 7 de 9x5

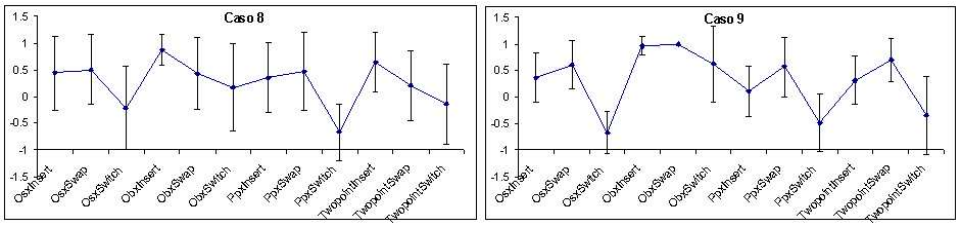

Figura 86: Criterio PD para el algoritmo NSGA-II. Casos 8 y 9 de 9x5

B.3 SPEA2

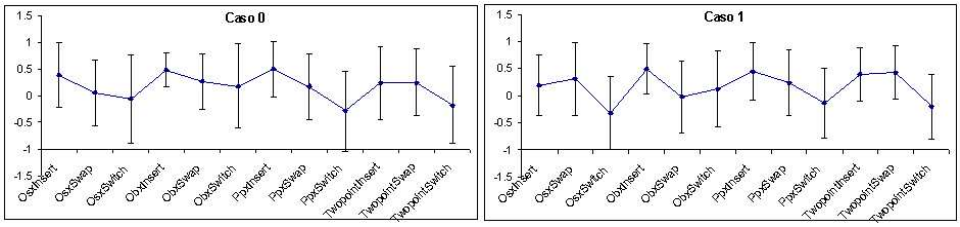

Figura 87: Criterio PD para el algoritmo SPEA2. Casos 0 y 1 de 9x5

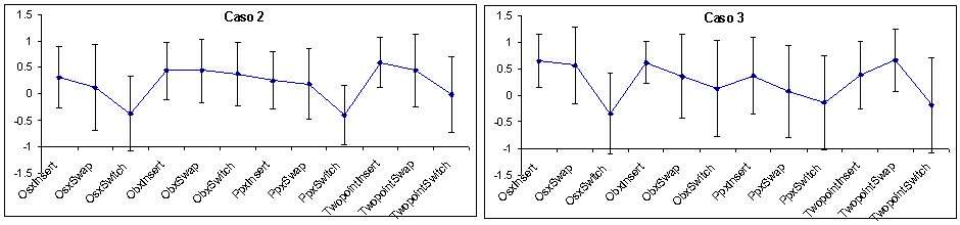

Figura 88: Criterio PD para el algoritmo SPEA2. Casos 2 y 3 de 9x5

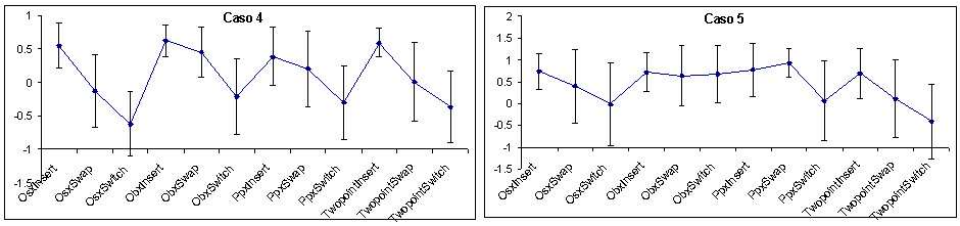

Figura 89: Criterio PD para el algoritmo SPEA2. Casos 4 y 5 de 9x5

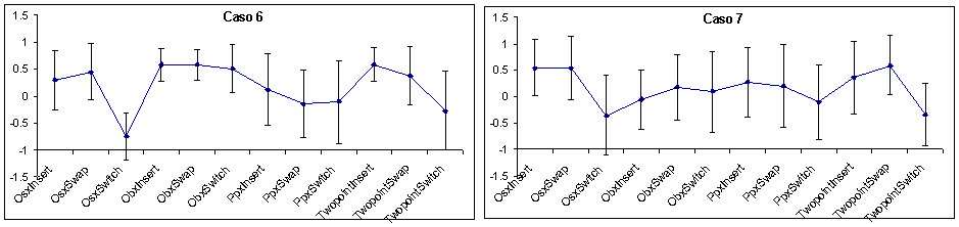

Figura 90: Criterio PD para el algoritmo SPEA2. Casos 6 y 7 de 9x5

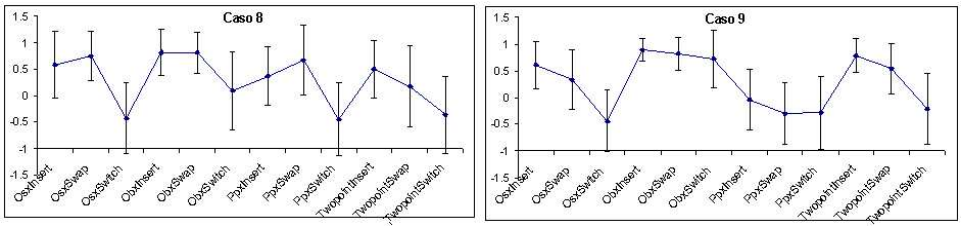

Figura 91: Criterio PD para el algoritmo SPEA2. Casos 8 y 9 de 9x5

Índice