DISEÑO DE UN ALGORITMO MEMÉTICO PARA LA PROGRAMACIÓN REACTIVA DE LA PRODUCCIÓN EN UN AMBIENTE DE SISTEMA DE MANUFACTURA FLEXIBLE: FALLA DE MÁQUINAS

JENNIFER GRICE REYES

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA – ÉNFASIS INGENIERÍA INDUSTRIAL
CARTAGENA, MARZO DE 2016

DISEÑO DE UN ALGORITMO MEMÉTICO PARA LA PROGRAMACIÓN REACTIVA DE LA PRODUCCIÓN EN UN AMBIENTE DE SISTEMA DE MANUFACTURA FLEXIBLE: FALLA DE MÁQUINAS

JENNIFER GRICE REYES

Trabajo de Grado presentado para optar al título de
Magister en Ingeniería con énfasis en Ingeniería Industrial

DIRECTOR

PhD (c) JAIME ACEVEDO CHEDID

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA – ÉNFASIS INGENIERÍA INDUSTRIAL
CARTAGENA, MARZO DE 2016

DISEÑO DE UN ALGORITMO MEMÉTICO PARA LA PROGRAMACIÓN REACTIVA DE LA PRODUCCIÓN EN UN AMBIENTE DE SISTEMA DE MANUFACTURA FLEXIBLE: FALLA DE MÁQUINAS

Nota de Aceptación:

_____
_____
_____
_____
_____
_____

_____
Firma Director Maestría en Ingeniería

_____
Firma del Jurado 1

_____
Firma del Jurado 2

_____
Firma del Jurado 3

Cartagena, Marzo de 2016.

DEDICATORIA

Dedico esta tesis a Dios, que siempre ha estado conmigo durante este largo camino llamado vida, a mis padres quienes me han dado educación, apoyo y consejos, a mi esposo quién ha sido un apoyo incondicional, a mis maestros, compañeros y amigos que con su paciencia y compañía han sido un gran soporte en todos los momentos.

AGRADECIMIENTOS

Este proyecto es el resultado del esfuerzo conjunto de todos los que han estado conmigo durante su desarrollo, es por esto que agradezco a mi director de tesis por brindarme sus conocimientos, a mi padre que con su gran apoyo en todo momento ha logrado hacer de mí quién soy, a mi madre que siempre me brindó un abrazo y sus consejos en los momentos difíciles, a mi hermano, abuela y primos que siempre me acompañaron, a mi esposo que con su apoyo siempre me dio más motivos para continuar y finalizar este proyecto. Gracias a Dios por todo!

CONTENIDO

Página

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ANEXOS

INTRODUCCIÓN

Hoy en día, la creciente competencia obliga a las empresas a satisfacer las necesidades de los clientes de forma ágil con el fin de conservar su puesto en el mercado. Es por esto que existe una gran variedad de productos que pueden ser fabricados dentro de una misma industria con el fin de satisfacer la demanda; esto es posible gracias a los sistemas de manufactura flexibles (FMS, por sus siglas en inglés) que son consecuencia de muchas de las innovaciones tecnológicas generadas a través de los últimos años.

Cuando a un taller llega una serie de trabajos a realizar, se procede a efectuar la programación de dichos trabajos teniendo en cuenta las máquinas, los materiales y los operadores disponibles para realizar los trabajos, es decir, se realiza una asignación de recursos para la fabricación de bienes o prestación de servicios, conociéndose este proceso como Programación de Producción.

La programación es una tarea compleja tanto en la teoría como en la práctica. Teóricamente, los problemas de programación, que consisten en buscar un programa óptimo sujeto a un limitado número de restricciones, sufren de complejidad excesiva y son en su mayoría NP-Hard, categoría en la que se ubican aquellos problemas que para su resolución no son suficientes tiempos computacionales polinomiales, son no determinísticos y que no se pueden resolver mediante el uso de algoritmos clásicos, se necesita la utilización de métodos heurísticos (Garey, M. and Johnson, D., 1976).

Después de obtener un programa óptimo se puede iniciar la producción. Sin embargo, el piso de la planta es un ambiente dinámico que cambia con cada nueva actividad, nuevo cliente o incluso nuevos materiales. Entre esos factores dinámicos se deben considerar las posibles fallas de las máquinas, el ausentismo de los trabajadores, la llegada de nuevas órdenes, entre otros. Responder inmediatamente a los factores dinámicos del piso de la planta cuando ellos ocurren, se llama programación reactiva de la producción o reprogramación en tiempo real (Acevedo J. y Mejía G., 2005).

Muchas han sido las formas de abordar el problema de fallas de máquina en los FMS, en lo que tiene que ver con la producción de un algoritmo eficiente en cuanto a tiempo y costos computacionales. En los últimos años se ha insistido en los algoritmos genéticos combinados con técnicas de búsqueda local entre las cuales destacan las de Branch and Bound, pero no son las únicas.

En esta investigación se estudió la perturbación en la programación inicial de los sistemas debido a la falla de las máquinas, que generan cambios en los objetivos evaluados en los FMS. Se abordó el

problema mediante un algoritmo híbrido entre las Redes de Petri, la simulación de eventos discretos y los algoritmos meméticos. El algoritmo híbrido fue llamado “PetNMA” y sus medidas de desempeño son el Makespan y la tardanza total ponderada.

Para generar un programa reactivo de producción se contemplan las operaciones que aún no han sido procesadas al momento de la falla y aquella operación que estaba siendo procesada por la máquina que falló en ese instante.

El algoritmo PetNMA, se encuentra conformado por tres (3) subalgoritmos: el primero para la programación inicial de los trabajos; el segundo para simular la ejecución del programa inicial establecido hasta que se produzca la falla de la máquina, por lo que se requiere una programación reactiva; y el tercero para construir la programación reactiva a través de la aplicación de un Algoritmo Memético (MA, por sus siglas en inglés), que ha sido conformado por un algoritmo genético y un algoritmo de recocido simulado (SA, pos sus siglas en inglés). Para probar el algoritmo propuesto, se utilizaron problemas que representan un FMS, obteniéndose buenos resultados en los objetivos propuestos. Los resultados del subalgoritmo de programación inicial fueron comparados con el cuello de botella y con las reglas de despacho: SPT (shortest Processing Time), LPT (Largest Processing Time), MRWT (Most Remaining Work Time), EDD (Erliest Due Date), CR (Critical Ratio), ATC (Apparent Tardiness Cost) y M. SLACK (Minimum Slack), mientras los resultados del subalgoritmo de programación reactiva fueron al comparados con las reglas de despacho: SPT (shortest Processing Time), LPT (Largest Processing Time), MRWT (Most Remaining Work Time), EDD (Erliest Due Date), CR (Critical Ratio), ATC (Apparent Tardiness Cost) y M. SLACK (Minimum Slack).

El presente documento está organizado de la siguiente manera: en la sección 2, se presenta el marco teórico a través de un resumen sobre FMS, Programación Reactiva de la Producción, Redes de Petri y Algoritmos Meméticos; en la sección 3, se realiza una descripción de la manera como fue abordado el problema de programación reactiva de la producción y los planteamientos considerados en la construcción del algoritmo “PetNMA”; en la sección 4, se presentan los resultados obtenidos con el algoritmo “PetNMA” en las corridas de los problemas de programación reactiva de la producción y las conclusiones de la investigación. Al final del documento se presentan los anexos que soportan la investigación y las referencias que se utilizaron.

1. MARCO TEÓRICO

Con el fin de satisfacer a los clientes y cumplir con las metas propuestas dentro de cada empresa u organización como son reducir al mínimo las horas extras, terminar los trabajos en las fechas pactadas, obtener una alta utilización de los recursos, entre otras, se deben realizar la planeación y programación de las actividades a largo, mediano y corto plazo sin importar si la empresa es de manufactura o de servicios. La planeación y programación interactúan con otras funciones que usualmente dependen del sistema y pueden diferir sustancialmente de un sistema a otro (Pinedo, 2009). Las compañías utilizan y/o elaboran sistemas de planificación de recursos empresariales (ERP, por sus siglas en inglés, Enterprise resource planning) para controlar y coordinar la información y las actividades en cada una de sus áreas. La implementación de un sistema ERP no es fácil ya que requiere de un desarrollo particular para cada empresa y puede llegar a ser muy costosa, sin embargo, el ERP puede llegar a ser un sistema de enlace para la toma de decisiones a largo, mediano y corto plazo.

Teniendo en cuenta las actividades a realizar que tienen repercusión a largo plazo, se debe realizar un Planeación de procesos y una Planeación estratégica de la capacidad (Actividades relacionadas con las decisiones de carácter estratégico.), de las cuales se desprenden la Planeación Agregada y un Plan maestro.

En la Planeación de procesos se determinan las tecnologías y procedimientos necesarios para realizar la producción de los bienes o la prestación de los servicios. Por su lado, la Planeación de la capacidad sirve para determinar las capacidades a largo plazo de los sistemas de producción. El plan agregado de las operaciones establece las tasas de producción por grupo de productos o por otras categorías amplias, para el mediano plazo (6 a 18 meses). Por último, el Plan Maestro genera los volúmenes y las fechas de entrega de las órdenes.

Teniendo un Plan Maestro de Producción, es necesario adquirir los materiales necesario con el fin de dar marcha a la producción de las ordenes, este requerimiento se realiza mediante una Planeación de requerimiento de materiales (MRP, por sus siglas en inglés, Material requirements planning).

Posteriormente, se realiza la programación de la producción que es la función de atribuir eficientemente los recursos de un sistema de producción sobre la dimensión del tiempo, para la fabricación eficaz de los productos (Rodammer, F. A. & White Jr, K. P., 1988; Baudin, M., 1990). En la figura 1, se pueden observar las diferentes actividades de tomas de decisiones que pueden impactar sobre la programación de la producción.

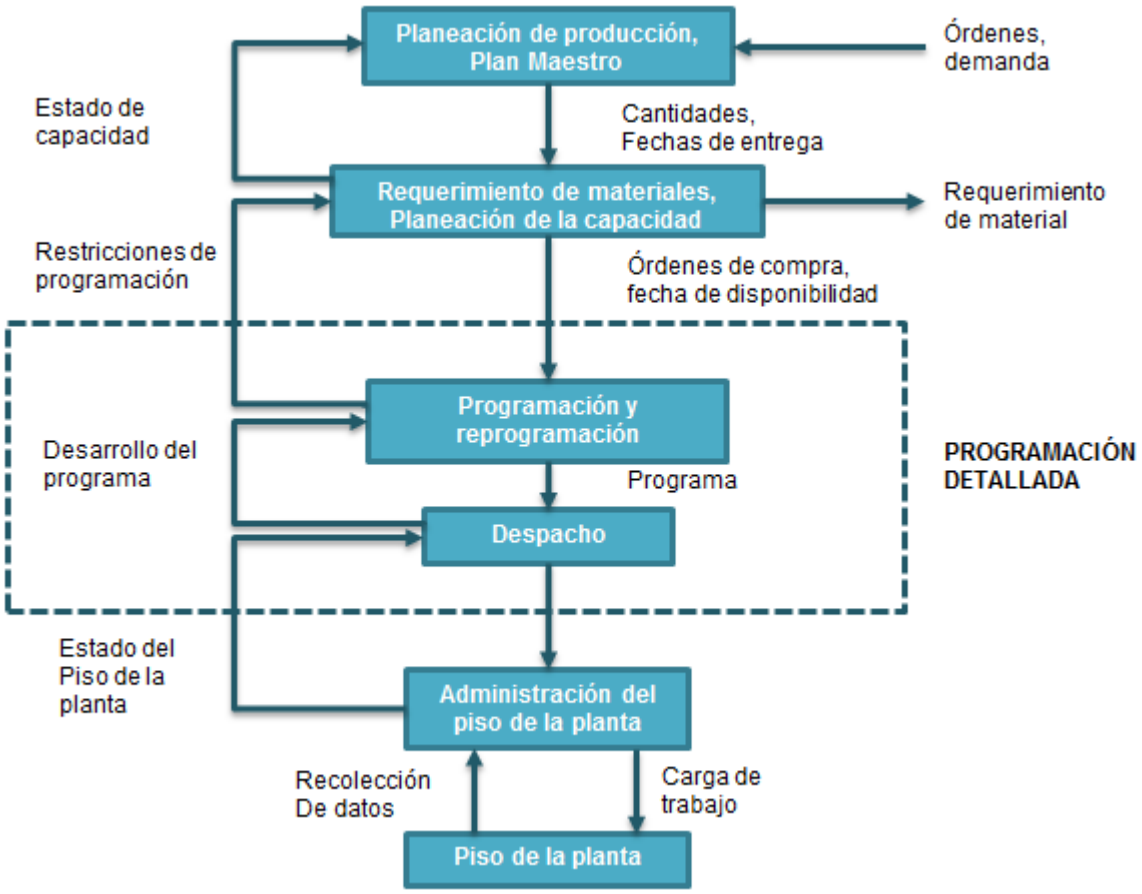


Figura 1. Diagrama de flujo de las actividades de información en un sistema de manufactura. Fuente: Pinedo, M., 2009.

Un problema de programación es descrito por una tripleta alfa/beta/gama: El campo alfa describe el ambiente de máquina y contiene una simple entrada; el campo beta proviene detalles de las características de procesamiento y restricciones y puede contener para todos múltiples entradas o ninguna; el campo gama describe el objetivo que se quiere minimizar y usualmente contiene una simple entrada.

1.1. Sistema de Manufactura Flexible

En la actualidad existe una gran competencia entre las empresas por satisfacer la demanda del mercado, por lo cual, la gama de productos ofrecidos a los consumidores es cada día más amplia. Cada producto lleva consigo un proceso de fabricación que puede parecerse o no a los procesos de fabricación de otros productos, lo cual ha hecho necesaria la creación de sistemas de manufactura flexibles que puedan responder ágilmente al mercado cambiante de hoy.

De acuerdo a la configuración de máquinas que tiene cada sistema de manufactura, éstos pueden ser clasificados en los siguientes modelos (Pinedo, 2009):

- Modelos de una sola máquina (Single machine models): Muchos sistemas de producción dan lugar a modelos de una sola máquina cuando existe una máquina cuello de botella, ya que la secuencia de dicha máquina generalmente determina el desarrollo del sistema.
- Modelos de máquinas paralelas (Parallel machine models). Constituyen una generalización de los modelos de una sola máquina. Muchos sistemas de producción agrupan las máquinas en centros de trabajo o estaciones que funcionan en paralelo, de tal forma que una operación puede procesarse en cualquiera de las máquinas de una misma estación. Sin embargo, las máquinas en paralelo pueden no ser idénticas, pueden tener diferentes velocidades, realizar el trabajo por medio de diferentes métodos o generar productos de diferente calidad.
- Modelos de taller de flujo continuo (Flow shop models): Si las rutas de todos los trabajos son idénticas, es decir, los trabajos tienen la misma secuencia sobre las máquinas, el ambiente es llamado taller de flujo. Las máquinas están configuradas en serie y cuando el procesamiento de un trabajo es completado sobre una máquina pasa de inmediato a hacer parte de la fila de la siguiente máquina. Una generalización del taller de flujo es el taller de flujo flexible (FJSP, por sus siglas en inglés), en donde hay un número de estaciones en serie, y en cada estación un número de máquinas en paralelo. En cada estación una operación puede ser procesada sobre cualquiera de las máquinas.
- Modelos de taller de trabajos (Job shop models): En estos modelos los trabajos tienen diferentes rutas en el sistema. En el modelo de taller de trabajos más simple cada trabajo pasa al menos una vez en su ruta a través del sistema. En otros, puede visitar una máquina muchas veces. Estos talleres pueden estar sujetos a recirculación, lo que incrementa la complejidad del modelo.
- Modelos de cadena de suministro (Supply chain models): Son modelos más generales en los que el ambiente de producción consiste en una red de instalaciones interconectadas y cada una de ellas puede ser un (flexible) flow shop o un (flexible) job shop.

En una economía globalizada, en donde los constantes cambios en la tecnología, requerimientos de los clientes, y exigencias gubernamentales, es necesario que los sistemas modernos tengan cierta flexibilidad que les permita variar tanto sus niveles de producción, así como también las líneas de sus productos. Para Krasa (1990), el concepto de flexibilidad es complejo, multidimensional y en algunos casos difíciles de entender; dicho concepto, en un momento dado, se convirtió en una consideración clave para el diseño, operación y administración de los sistemas de manufactura. Lavington (1921), fue el primero que estableció una conexión entre los cambios aleatorios y el valor de la flexibilidad, al considerar el riesgo derivado de la inmovilidad de los

recursos invertidos.

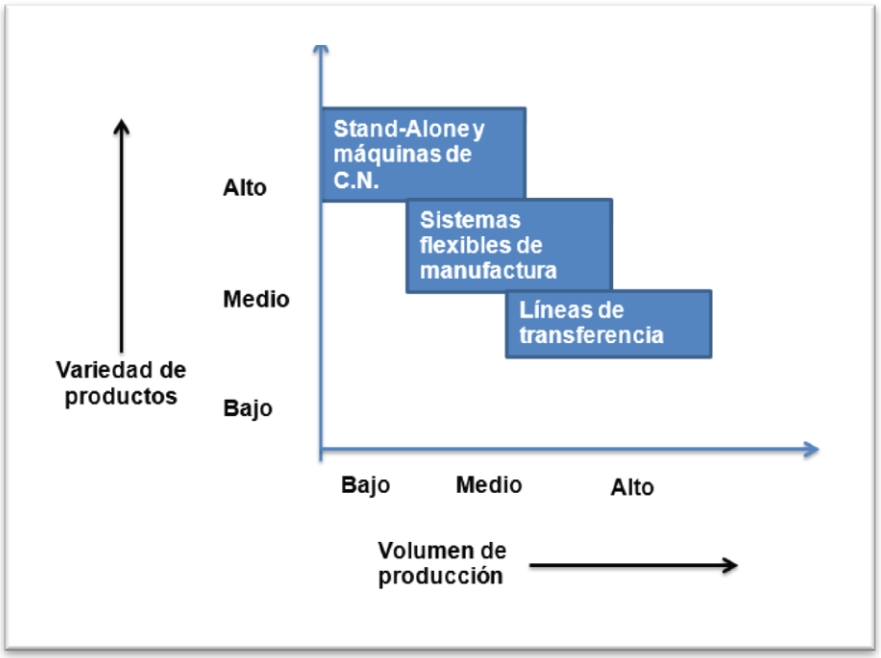
Desde el punto de vista económico, el deseo de flexibilidad de una planta se incrementa con el aumento de la variabilidad de los precios del mercado y la habilidad de predecirlos antes de tomar decisiones en los sistemas de producción (Marshak, T. y Nelson, R., 1962). Para Leaver, E.W. and Brown, J.J. (1946), la filosofía tradicional en el diseño de las máquinas, ponía énfasis en el desarrollo del producto más que en la operación; para romper con esto, sugirieron diseños de máquinas con base en las funciones a ejecutar, proponiendo una variedad de máquinas pequeñas orientadas a funciones y conectadas entre sí, los cuales para su evolución tuvieron que esperar el advenimiento de la tecnología del microprocesador.

En sus inicios, la flexibilidad generaba ineficiencia, y sólo a finales de los años 60 se supo cómo extender dicha flexibilidad para la producción en gran escala, sin necesidad de sacrificar la eficiencia. En este sentido, se buscó dotar de flexibilidad a las máquinas tanto como la tienen los humanos; apareciendo así las líneas de transferencia automatizadas, los caminos suaves (para el transporte), los vehículos automatizados, en el sentido de la mecanización y automatización (Simon, H.A., 1977). La flexibilidad en el campo de la manufactura significa ser capaz de reconfigurar los recursos del sistema, para producir eficientemente los diferentes productos con los mayores estándares de calidad (Krasa, 1990). De lo anterior, se desprende que la flexibilidad puede darse en las máquinas o en las rutas definidas para el desarrollo de cada uno de los productos, entendiendo como flexibilidad en las máquinas, la capacidad del sistema de producir varios tipos de productos y cambiar la secuencia de las operaciones a ejecutar, mientas que la flexibilidad en las rutas, como la capacidad que tiene el sistema de absorber los cambios a gran escala que se presenten, tales como en el volumen de producción y/o variedad de productos, que generan la necesidad de adaptar el sistema a las nuevas condiciones (Ver Figura 2). Sin embargo, existen otros enfoques para la flexibilidad, dependiendo del enfoque hacia la Manufactura, las Operaciones, los Clientes, las Estrategias y la Capacidad (Ver **Tabla 1**).

Pinedo (2009), puede ser visto como un (flexible) job shop con un número adicional de restricciones, por lo que definió a los Sistemas de Manufactura Flexible (Flexible Manufacturing System - FMS), como:

> "Sistema de producción que es capaz de producir una variedad de tipos de partes; éste consiste en máquinas que están conectadas a otras por un sistema automatizado de manejo de materiales. Usualmente, todo el sistema se encuentra bajo control computacional. Con un apropiado conjunto de herramientas, cada máquina en el sistema puede desarrollar diferentes operaciones. Una operación puede ser procesada en cualquiera de un número de máquinas. La ruta de un trabajo a través del sistema es también flexible y es una decisión que tiene que ser tomada en el proceso de

programación".



Firgura 2.

Aplicación de las características de los FMS

Fuente: Rajan, N. et al, (2014).

Tabla 1

Diferentes enfoques a la flexibilidad y sus significados

ENFOQUE	SIGNIFICADO DE LA FLEXIBILIDAD
Manufactura	La capacidad de producir diferentes partes sin mayor reorganización y cambios.
	Una medida de cuán rápido la compañía convierte una vieja línea de productos a producir una línea nueva.
	La habilidad de cambiar un programa de producción, para modificar una parte, o la manipulación de múltiples partes.
Operacional	La capacidad de producir eficientemente productos altamente personalizados que también son únicos.
Clientes	La capacidad de explotar las diversas dimensiones de la velocidad de entrega.
Estratégica	La capacidad de una empresa para ofrecer una amplia gama y variedad de productos para sus clientes.
Capacidad	La capacidad de aumentar o disminuir rápidamente los niveles de producción o cambiar la capacidad rápidamente de un servicio o producto a otro servicio o producto.

Fuente: Rajan, N. et al, (2014).

Al igual que en cualquier sistema de manufactura, en los FMS se debe realizar una correcta programación de sus actividades y asignación de sus recursos, con el fin de alcanzar los objetivos establecidos, tanto en términos de eficiencia como de eficacia. Por tanto, la creación y posterior implementación del diseño de un FMS, resulta útil para poder establecer y programar la producción a través del sistema.

En la literatura sobre programación de FMS, se evidencia la utilización de modelos matemáticos y técnicas heurísticas para la programación de los trabajos, estableciendo la mejor secuencia de visitas a las estaciones de trabajo y la asignación a las máquinas, dada la mezcla actual de productos a realiza (Acevedo, J. y Mejía, G., 2005). Low, C. y Wu, T. H. (2001), representaron simbólicamente a los FMS, como: conjunto de máquinas $M^T = \{m_1, m_2, \dots, m_M\}$ y un conjunto de trabajos $J = \{J_1\ ,\ J_2\ , \dots, J_N\}$, donde cada trabajo $J_j, \forall_j \in \{1, 2, \dots, N\}$ tiene una fecha de entrega d_j comprometida con el cliente y se compone de una secuencia de n_j operaciones. Cada operación $O_{i,j},\ \forall i \in \{1,2, \dots, N\}\ y\ \forall j \in \{1,2, \dots, n_i\}$, puede ser procesada de manera ininterrumpida en una máquina cualquiera m_k del conjunto de máquinas $M^S, \left(M^S \subseteq M^T\right)$ en un tiempo de procesamiento $p_{i,j,k}$ que incluye el tiempo de alistamiento sobre la máquina. Cada máquina $m_k,\ \forall k \in \{1, 2, \dots, M\}$ puede procesar sólo una operación y en cualquier instante solamente puede estar siendo procesada una operación de cada trabajo. Se asume que todos los $p_{i,j,k},\ d_j$ y n_j son determinísticos, así como también que todos los trabajos están disponibles cuando se hace su programación, es decir, su tiempo de disponibilidad r_j es igual a cero.

Los problemas de programación y secuenciación han sido clasificados según la complejidad computacional que se requiere para resolverlos; dentro de las categorías se encuentra la categoría NP-Hard en la que se ubican aquellos problemas que para su resolución no son suficientes tiempos computacionales polinomiales, son no determinísticos y que no se pueden resolver mediante el uso de algoritmos clásicos, se necesita la utilización de métodos heurísticos. Para Garey, M. and Johnson, D., (1976), la obtención de la programación óptima en los sistemas de manufactura puede ser muy compleja y en algunos casos imposible de hallar en un tiempo razonable, situación que no es ajena a los FMS. En sentido, dicho problema de programación se describe como un problema combinatorio de alta dificultad que los llevó a clasificarlo como NP-Hard.

Para realizar un modelamiento de un FMS se deben tener en cuenta las características que dichos sistemas presentan. Según Zakaria et al (2008), las características más importantes son:

- Concurrencia o paralelismo: Muchas operaciones pueden darse al mismo tiempo.
- Operaciones asincrónicas: Los eventos del sistema pueden no ser periódicos.
- Manejados por eventos: La finalización de una operación puede dar lugar a más de una

operación.

- Deadlock (Bloqueo mutuo): En este caso, se puede llegar a un estado en el que ninguno de los procesos pueda continuar. Esta situación es indeseable y usualmente es resultado de un mal diseño del sistema. Un buen modelo puede detectar los deadlocks, permitiendo así la corrección y rediseño del sistema.
- Conflicto: Dos o más procesos requieren para su realización un recurso común al mismo tiempo. Para resolver el conflicto se debería asignar un nivel de prioridad a cada proceso.

Teniendo en cuenta estas características, Silva et al., (2005) plantea que las Redes de Petri son una buena herramienta para el modelado de los FMS, manifestando para justificar su apreciación las siguientes razones: Las redes de Petri pueden modelar fácilmente la sincronización necesaria entre las operaciones debido a las rutas o secuencias de los productos; los FMS son sistemas discretos y su modelado debe basarse en eventos y actividades, en donde un evento corresponde a un cambio de estado y una actividad recoge lo que ocurre entre dos eventos; cuando se utilizan las redes de Petri los eventos estarían asociados con las transiciones y las actividades con las plazas.

1.2. Programación reactiva de la producción

Rodammer, F. A. & White Jr, K. P. (1988) y Baudin, M. (1990), plantearon que la programación de las operaciones es la función de atribuir eficientemente los recursos de un sistema de producción sobre la dimensión del tiempo, para la fabricación eficaz de los productos. Cuando se realiza la programación de las producción de un centro de trabajo o de un sistema de manufactura, se debe realizar con el fin de alcanzar algún(os) objetivo(s) propuesto para el desempeño del sistema; dentro de los objetivos más comúnmente buscados, algunos le apuestan a lograr la mayor eficiencia y otros a una mejor eficacia, sin embargo, en un mercado tan competitivo como el actual, lo ideal es la búsqueda de la mejor combinación en los mismos, por cuanto mientras las organizaciones buscan la mayor rentabilidad, los clientes persiguen a las organizaciones que le ofrezcan el mayor beneficio. En este sentido, cumplir con las fechas pactadas con los clientes, reducir los tiempos de producción, reducir el tiempo o costo asociado a la preparación de las máquinas, reducir el inventario de productos en proceso y maximizar la utilización de las máquinas o mano de obra, son algunos de los objetivos considerados (Chase et al., 2005).

Son muchas procedimientos y estrategias utilizadas para desarrollar una programación de la producción, que permita alcanzar dichos objetivos. Desde la modelación matemática del sistema y sus restricciones, la utilización de reglas de despacho, procedimientos heurísticos y metaheurísticos, entre otros, son utilizados para dicho propósito.

En lo que respecta a la modelación matemática, son muchos los autores que presentan modelos

matemáticos para resolver el problema de programación de un FMS. Para resaltar algunos: Liu, J. and MacCarthy, B.L. (1997), presentaron un modelo de programación lineal entera mixta (MILP) para la programación de un FMS; el modelo tiene una visión global del problema y donde se tiene en cuenta las limitaciones de almacenamiento y transporte, los cuales para este tipo de problemas son considerados como críticos y difíciles de modelar; Low, Ch., et al., (2006), presentan uno de los métodos de toma de decisiones más famosos para multiobjetivos (MODM), el método de criterio global, utilizado para resolver el problema de programación de un FMS, en donde tres medidas de desempeño son considerados al tiempo: el tiempo de flujo promedio, la tardanza promedio de los trabajos y el tiempo promedio de inactividad de las máquinas; y Caumond, A., et al., (2009), propone un MILP que se ocupa del problema de programación de un FMS con un solo vehículo, e introduce simultáneamente por primera vez las siguientes restricciones: i) Regla FIFO como estrategia de gestión del buffer, ii) prohíbe viajes por adelantado, iii) limitar el número de puestos de trabajo en el sistema a fin de evitar deadlock, y iv) trata con capacidad de buffer de salida distinto de 0.

Por la complejidad computacional para la búsqueda de la solución óptima en los modelos matemáticos, muchos programadores han optado por utilizan reglas de secuenciación u orden de prioridad que utilizan ciertos datos de los trabajos (Fecha de entrega, tiempo de procesamiento, entre otros), para establecer la secuencia de las operaciones en el sistema, dichas reglas son las denominadas “Reglas de Despacho”. Las reglas de despacho más comunes son (Chase et al., 2005):

1. FIFO (por sus siglas en inglés): Los pedidos son procesados siguiendo el orden en que llegan al departamento.
2. Tiempo más breve de operación (SOT, por sus siglas en inglés). Primero procesar la tarea que tarda menos tiempo en quedar terminada, después la siguiente que requiere menos tiempo y así sucesivamente. En ocasiones esto último también se conoce como tiempo más breve de procesamiento (SPT, por sus siglas en inglés). Esta regla con frecuencia se combina con la regla de la demora para evitar que las tareas que requieren más tiempo se retrasen demasiado.
3. Fecha de vencimiento; primera fecha de vencimiento primero (DDATE, por sus siglas en inglés), también llamada EDD. Procesar el trabajo tomando primero el que tiene la primera fecha de vencimiento.
4. Margen de tiempo restante (STR, por sus siglas en inglés). Éste se calcula como el tiempo que resta antes de la fecha de vencimiento menos el tiempo que resta para su procesamiento. Los pedidos que tienen menos margen de tiempo restante son procesados primero.

   SRT = Tiempo restante antes de fecha de vencimiento – tiempo restante para su procesamiento

5. Margen de tiempo restante por operación (SPR/OP, por sus siglas en inglés). Los pedidos que tienen menos margen de tiempo por cantidad de operaciones son procesados primero.
   STR/OP = STR/Número de operaciones restantes
6. Proporción crítica (Cr, por sus siglas en inglés). Ésta se calcula como la diferencia entre la fecha de vencimiento y la fecha corriente, dividida entre la cantidad de jornadas laborales restantes. Los pedidos con la proporción crítica más baja son procesadas primero.
7. Último en llegar, primero en salir (LCFS, por sus siglas en inglés). Esta regla ocurre con frecuencia por default. A medida que entran los pedidos, éstos son colocados arriba del montón; el operador normalmente toma primero el pedido que está arriba del montón para procesarlo.
8. Orden aleatorio o capricho. Los supervisores o los operadores normalmente escogen el trabajo que quieren para procesarlo.

Los objetivos o indicadores de desempeño que se utilizan para evaluar los programas de producción más importantes son (Pinedo, 2009):
Objetivos de rendimiento y makespan:

- Makespan, corresponde al tiempo en el que el último trabajo deja el sistema.

$$C_{max} = \max(C1, \dots, C_n)$$

En donde C_j es el tiempo de terminación del trabajo j.

El makespan está muy relacionado con el rendimiento del sistema, debido a que el rendimiento de un sistema es equivalente a la rata de salidas y ésta se encuentra determinada por la máquina cuello de botella (Si existe) que es la que tiene una menor capacidad.

Objetivos relacionados con la fecha de entrega:

- Retraso máximo, es el retraso máximo de todos los trabajos en el sistema.

$$L_{max} = \max(L_1, \dots, L_j)$$

El retraso de un trabajo L_j es la diferencia entre la fecha de entrega pactada y la fecha de terminación de un trabajo.
$L_j = C_j - d_j$, en donde d_j es la fecha de entrega.

- Tardanza total, es la suma de todas las tardanzas de los trabajos.

$$\sum_{j=1}^{n} T_j$$

La tardanza de un trabajo T_j es mayor valor entre la diferencia entre la fecha de entrega pactada y la fecha de terminación de un trabajo y cero.

$$T_j = \max(C_j - d_j, 0)$$

- Tardanza total ponderada, es la suma de todas las tardanzas de los trabajos teniendo en cuenta los pesos de los trabajos.

$$\sum_{j=1}^{n} w_j T_j$$

Costos de inventario de trabajo en proceso:

- Suma de la terminación de los trabajos:

$$\sum_{j=1}^{n} C_j$$

  Este objetivo es equivalente a minimizar el promedio de los trabajos en el sistema, lo que se ve reflejado en maximizar las salidas y a su vez el rendimiento.

- Suma de la terminación de los trabajos ponderada, es la suma de todos los tiempos de terminación de los trabajos teniendo en cuenta los pesos de los trabajos.

$$\sum_{j=1}^{n} w_j C_j$$

Se pueden obtener varios programas para la solución o secuenciación de un problema, es por esto que se han clasificado en (Pinedo, M., 2009):

- Programas Sin Retraso: Un programa factible es sin retraso si ninguna máquina se mantiene disponible cuando existe un trabajo esperando por ser procesado.
- Programas Activos: Son aquellos que no se pueden alterar en forma tal que alguna operación se complete antes ni otra operación se complete después. El que un programa sea activo implica que cuando un trabajo llega a una máquina, éste se procesa tan pronto como sea posible de acuerdo con la secuencia prevista. Un programa activo no puede tener un período muerto en el cual pueda caber la operación de un trabajo en espera Un programa activo tiene la propiedad de que es imposible reducir el makespan sin aumentar el tiempo de inicio de alguna operación. Por supuesto, hay muchos programas activos diferentes. Se puede demostrar que existe un programa activo que es el óptimo entre todos los posibles.
- Programas Semiactivos: Se caracterizan por el hecho de que el tiempo de inicio de cualquier operación de cualquier trabajo en cualquier máquina es igual al tiempo de finalización de la operación precedente del mismo trabajo en una máquina diferente o al tiempo de finalización de una operación de un trabajo diferente en la misma máquina. Además, en un programa semiactivo, un trabajo no puede hacerse iniciar antes sin cambiar la secuencia de trabajos o violar la factibilidad (condiciones de precedencia y fechas de entrega) (Aloulou, M. y Portmann, M., 2003).

Figura 2. Diagrama de Venn de clases de programas nonpreemptive para Job Shops. Fuente: Pinedo, M., 2009.

Además de las reglas de despacho mencionadas anteriormente, también se utilizan otros métodos para la obtención de programa factibles para la solución de los problemas de producción como son el recocido simulado, la colonia de hormigas, algoritmos genéticos, algoritmos meméticos, métodos de inteligencia artificial, entre otros.

Durante el desarrollo del programa de producción pueden suceder eventos aleatorios o imprevistos, lo cual implica que se debe hacer una reprogramación de lo planeado, también llamada programación reactiva, afectando en algunos casos la programación inicial y el nivel de servicio a los clientes. El ambiente dinámico del sistema de producción, necesita que cualquier programa que sea diseñado, deba estar sujeto a revisión frecuente para garantizar su alineación con las nuevas condiciones. Programar así es por lo tanto un proceso continuo, denominado "programación reactiva de la producción" (Dorn, J., Kerr, R., & Thalhammer G., 1995). Eventos aleatorios asociados a este tipo de problema de programación, están relacionados con fallas en las máquinas, llegada de nuevas órdenes de trabajo, cambio en las prioridades, cancelación de trabajos, incumplimiento de proveedores, mala calidad en los materiales, ausentismo de trabajadores, entre otros. La incertidumbre en la programación puede tomar muchas formas. Por ejemplo, se puede considerar la programación de un conjunto conocido de puestos de trabajo con la incertidumbre en los parámetros específicos de empleo, tales como los tiempos de procesamiento o por fechas. Otra forma de incertidumbre implica interrupciones en la entrega debido a fallas de equipo, la llegada de nuevos puestos de trabajo, o cambios en las fechas de

vencimiento, entre otros. Los trabajos pueden llegar periódicamente de acuerdo con alguna distribución de probabilidad de los tiempos entre llegadas, lo que conduce a una clase de problemas conocidos como problemas de programación en línea (Bibo, Y. y Geunes, J., 2008).

La mayoría de los acercamientos a la programación reactiva de la producción o reprogramación, se habían fundamentado en la generación de un programa predictivo básico, que mantenía su continuidad sirviendo como punto de referencia para la identificación y especificación de los cambios que lo modificaban progresivamente, cuando el evento aleatorio ocurría (Smith S. F., 1994).
Los tres métodos más comunes para realizar la reprogramación son: Regeneracion, reprogramación parcial, y programación de desplazamiento a la derecha (Right-shift scheduling). La regeneración contempla no sólo las operaciones (o trabajos) afectadas sino también las no afectadas; conlleva más tiempo de corrida pero también mejores programas. La reprogramación parcial por su parte sólo toma en cuenta las operaciones afectadas. El método right-shift pospone las operaciones restante en una cantidad de tiempo igual al dla falla; este método lleva a programas más estables (Viera et al, 2003).

Algunas de los trabajos o estudios realizados en el campo de la programación de la producción son:

- Mehta, S.V. y Uzsoy, R., (1999) realizaron un estudio cuyo objetivo fue minimizar el máximo retraso (Lmax) sobre una única máquina con llegada dinámica de trabajos (Los trabajos llegan aleatoriamente en cualquier momento, incluso cuando ya ha iniciado la producción de lo planeado) y fallas de máquinas. Se insertaron tiempos muertos mientras que simulaban la falla de la máquina. La propuesta predictiva supone mejoras importantes de la robustez a costa de una pequeña disminución de Lmax.
- Chen,J. y Chen,F.F., (2003), desarrollaron un algoritmo que debía hacer frente a los diversos eventos aleatorios del taller como las fallas en máquinas teniendo en cuenta las asignaciones de trabajos a los recursos y las herramientas a las operaciones. El algoritmo estaba dividido en dos etapas: asignación de un trabajo a una estación/máquina y asignación de una herramienta a una operación, en un taller de manufactura flexible. Para ello se utiliza una política de reprogramación dirigida por eventos, es decir, la secuencia está determinada por eventos aleatorios que pueden suceder en cualquier instante y como estrategias de secuenciación utilizan las reglas de despacho. Se plantean tres medidas de rendimiento: tiempo de flujo medio, retrasos de los trabajos y porcentaje de trabajos retrasados. El programa se modifica completamente en cada revisión si es necesario. Utilizan el software Arena para la simulación y visual basic como plataforma para la programación del algoritmo. Concluyen que reprogramaciones frecuentes para reaccionar ante disrupciones pueden hacer del comportamiento del sistema difícil de predecir y reducir la efectividad de la programación dinámica.
- Acevedo, J. y Mejía, G. (2005), realizaron un algoritmo genético para la programación

reactiva de la producción en sistemas de manufactura flexibles con arribos de nuevas órdenes de trabajo y cambio de prioridades. Utilizan las reglas de despacho para la obtención de un programa inicial y como estrategia de búsqueda de una mejor solución dentro de la población inicial en el algoritmo genético para la reprogramación. Concluyen que la variabilidad en el tiempo máximo de terminación de todas las ordenes de trabajo, al considerar el arribo de nuevas órdenes, depende del tipo de órdenes de trabajo que arriban, el tiempo en que lo hacen y el estado del sistema al momento de su ocurrencia. Cuando los arribos de suceden de manera temprana al inicio de la ejecución del programa inicial, la variación en el tiempo de terminación de todas las ordenes de trabajo es menor que cuando el arribo se sucede cuando la ejecución del programa está muy avanzada.

- Tanimizu, Y. et al (2006), desarrollaron un algoritmo genético que genera continuamente nuevos programas de producción cuando arriban nuevos trabajos durante la ejecución del programa inicial, hasta que uno de ellos satisface la restricción propuesta en el sistema o todas las operaciones de han comenzado su ejecución. Varios experimentos computacionales se llevaron a cabo mediante el uso de un sistema prototipo desarrollado para la programación reactiva que busca adecuados espacios de tiempo libre en el que las operaciones de los nuevos trabajos se insertan, para evaluar los retrasos de las operaciones de fabricación por la adición de nuevos trabajos, verificando así la efectividad del método propuesto, utilizando como indicador de desempeño el makespan. Los resultados experimentales muestran que los arribos generan un 30% de retraso del makespan (Valor promedio) y con el método propuesto se reduce en promedio el 61% de dicho retraso.

Uno de los eventos aleatorios más frecuentes en los sistemas de manufactura es la falla de las máquinas. Cuando éste evento sucede se pueden realizar dos tipos de reprogramación: reparación del programa y reelaboración total. La reparación se refiere a ajustes locales del programa actual mientras que en la reprogramación total se genera un nuevo programa desde la base. La reprogramación total puede implicar más esfuerzo computacional. Sin embargo, puede ser mejor manteniendo soluciones óptimas aunque raramente realizables en la práctica. Se debe tener en cuenta es “Cuándo reprogramar”; a este efecto, se tiene las siguientes políticas que corresponden a un enfoque predictivo-reactivo (Oueljadj y Petrovic, 2009):

- Por frecuencia: Se hace cada cierto tiempo.
- Dirigida por eventos: Cuando se presentan fallas o cambios (Fallas de máquina, cancelación de trabajos, llegada de órdenes urgentes y cambios en las fechas de entrega).
- Híbrida: Se hace frecuentemente y cuando se presentan fallas.

1.3. Fallas de máquinas

Las fallas de máquinas tienen más impacto en el desempeño del sistema que las variaciones en los tiempos de procesamiento, entre más baja sea la frecuencia de reprogramación menores son los

tiempos de alistamiento y la reprogramación eventual es mejor que las reprogramaciones frecuentes programadas (Viera et al, 2003).

Para abordar la reprogramación teniendo en cuenta fallas de máquinas se deben determinar cuáles tareas se han completado y cuales se afectan. Una tarea se llama "completa" si se ha finalizado antes de la presentación de la falla. Una tarea se considera "afectada" si necesita ser reubicada debido a la interrupción; el conjunto de tareas afectadas se genera con base en las relaciones de precedencia. En cualquier momento, si el cambio de una tarea no afecta a su sucesora, ésta no se considera afectada. Los programas reactivos están basados en las tareas afectadas que empiezan de acuerdo con el tiempo revisado de inicio de cada máquina; estos tiempos se calculan con base en los tiempos de finalización de las tareas completadas (Kamrul, S.M., 2011).

En un sistema de manufactura la no disponibilidad de una máquina puede conocerse con anterioridad y manejarse de mejor manera mientras la falla se presenta de forma inesperada cuando la máquina está procesando alguna operación. Al tratar de generar o simular una falla de máquina, es bueno considerar factores como la robustez y estabilidad del programa como elementos complementarios muy importantes. Estos elementos están siendo muy tenidos en cuenta para minimizar el efecto de las fallas de máquina. Para cada uno existen muchas definiciones, según Al-Hinai, H. y Elmekkawy, T.Y (2011) un programa es robusto si puede absorber perturbaciones externas sin pérdida de consistencia, sin amplificar los efectos de un cambio sobre todos los componentes del programa. Se relaciona con el grado de degradación del makespan bajo perturbación. Por su parte, la estabilidad se mide con referencia a la suma de las desviaciones absolutas de los tiempos de finalización de operaciones con respecto a los reales, si esta suma puede ser pequeña o no.

Para tener medidas adecuadas en este caso, es necesario tener en cuenta la probabilidad de falla de las máquinas que se cuantifica por el tiempo que la máquina ha estado ocupada dividido por el tiempo total que han estado ocupadas todas las máquinas en cuestión. Se asume que una máquina que ha estado más ocupada que otras tiene mayor probabilidad de fallar y su tiempo de falla y duración se basan en su tiempo ocupado transcurrido u horas de operación. Cuando se integra en los problemas la probabilidad de falla de las máquinas, se puede lograr un programa que pueda absorber el impacto de fallas futuros de máquina. Si un sistema de manufactura cuenta con datos históricos que provean una distribución aproximada de las fallas de las máquinas, se puede utilizar para predecir el instante de la falla de las máquinas y el tiempo de reparación de ellas con el fin de obtener un programa predictivo robusto y estable. Cuando no se tienen datos históricos, el tiempo en que se produce la falla se puede determinar aleatoriamente utilizando la distribución de probabilidad normal, la duración de la falla o tiempo de reparación de la máquina puede ser determinado utilizando la distribución de Gauss y la máquina en que sucede la falla

puede escogerse teniendo en cuenta la distribución exponencial, de acuerdo con las fórmulas establecidas para cada caso (Mehta, S.V. y Uzsoy, R., 1999).

Cuando sucede una falla, las operaciones restantes pueden ser desarrolladas en otras máquinas. En el momento de la interrupción, si se está ejecutando una tarea en la máquina que falla, se deben buscar máquinas disponibles; si se encuentra una libre, se le asigna la tarea pendiente. Si una máquina alternativa está cumpliendo una tarea, se evalúan las prioridades de las tareas y se asigna la de mayor prioridad a esa máquina. También debe tenerse en cuenta el tiempo de preparación de la actividad; en este caso debe compararse este tiempo en la máquina alternativa y el de la falla; si el tiempo de preparación es menor, se pasa a la máquina alternativa; si no, se debe esperar hasta que la máquina fallida sea reparada (Kumar et al., 2011).

En diversos estudios han sido utilizadas diversas distribuciones para simular las fallas de las máquinas. Kutanoglu, E. y Sabuncuoglu, I. (2001), realizaron un estudio para la programación reactiva en job shops dinámicos con fallas de máquinas, tomando como indicador de desempeño la tardanza total ponderada. Establecieron unas políticas para hacer un reruteo de los trabajos que no habían sido procesados al instante de la falla en las máquinas alternativas. En su estudio utilizan la distribución Gamma para establecer el tiempo de reparación de la máquina que falla y concluyen que la tardanza total ponderada del sistema se ve afectada significativamente dependiendo del instante en qué sucede la máquina, sin tener tanta importancia la longitud y la frecuencia de la falla; cuando la falla sucede más cerca de la fecha de entrega, el deterioro de la medida de rendimiento es más visible. Suwa, H. y Sandoh, H. (2007), utilizan en su estudio la distribución Log-Normal para establecer los tiempos de reparación de las máquinas, para lo cual se necesitan los tiempos promedio de procesamiento del sistema y la varianza; esto con el fin de evaluar la tardanza de los trabajos en un job shop sujeto a fallas de máquinas. Por medio de la distribución Log-Normal crean diferentes escenarios de fallas y concluyen que las tardanzas acumuladas de las tareas tienden a incrementarse considerablemente y de forma rápida con el tiempo transcurrido después de la falla.

1.4. Redes de Petri

Una Red de Petri (RdP) es un tipo particular de grafo dirigido, pesado y bipartito, consistente de plazas, transiciones y arcos, donde los arcos van de una plaza a una transición o de una transición a una plaza. En la representación gráfica las plazas son representadas con círculos, las transiciones con barras y los arcos por flechas. Un marcaje en la RdP indica la distribución de un número entero (cero o positivo) de token en cada plaza (Cho, H., 1998).

Murata en el año 1989 definió de manera formal a las Redes de Petri como la representación séxtuple $RdP = (P, T, O, I, M, W)$, donde:

- $P = \{p_1, p_2, ..., p_n\}$ es el conjunto finito de plazas representadas por círculos, para $n > 0$.

- $T = \{t_1, t_2, ..., t_m\}$ es el conjunto finito de transiciones representados por barras, con $P \cup T \neq 0$ y $P \cap T = 0$ para $m > 0$.
- $O : TxP \rightarrow N$, es el conjunto de arcos dirigidos desde T hasta P, donde $N = \{0,1,2,...\}$.
- $I : PxT \rightarrow N$, es el conjunto de arcos dirigidos desde P hasta T, donde $N = \{0,1,2,...\}$.
- $M(p) : P \rightarrow N$, es el marcaje que representa en el componente p-ésimo el número de token en la p-ésima plaza en algún instante. Un marcaje inicial de la RdP es denotado por Mo. Los token son representados por puntos.
- W, es el conjunto de pesos asociados a los arcos.

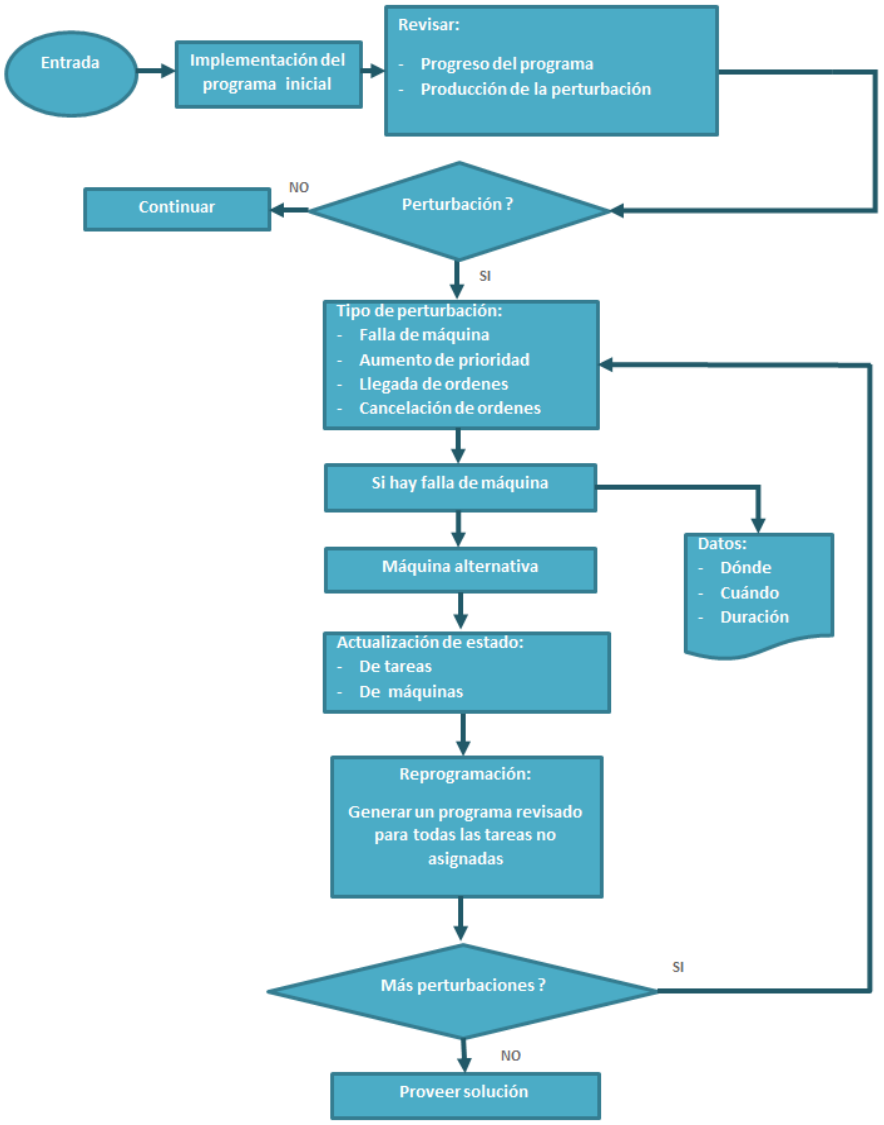


Figura 3. Marco de la fase de control empleado en los sistemas de manufactura para monitorear y ajustar el programa, con el fin de asegurar el progreso de las ordenes sobre el piso del taller. Fuente: Kumar et al., 2011.

En las RdP, una transición *t* se dice que está habilitada si todas sus plazas de entrada están marcadas con por lo menos *w(p, t)* tokens, donde *w(p, t)* es el peso del arco desde *p* hasta *t*; además, una transición puede dispararse si está habilitada. Cuando una transición se dispara, se remueven *w(p, t)* tokens de sus plazas de entrada y ponen *w(t, p)* tokens en sus plazas de salida, teniendo en cuenta las restricciones del sistema modelado. Una transición sin plazas de entrada es llamada una transición fuente y estará siempre habilitada, mientras que una transición sin plazas de salida es una transición final. Una transición t y una plaza p conforman un bucle si p es al mismo tiempo, plaza de entrada y salida de t.

Murata, T. (1989), utilizo las siguientes representaciones simbólicas para las redes de Petri las cuales han sido adoptadas para efectos de la presente investigación:

- $M(p)$, número de tokens en la plaza p o marcaje de la plaza p.
- $K(p)$, número máximo de tokens que puede albergar la plaza p.
- $w(p,t)$, peso del arco que comunica la plaza p con la transición t.
- $w(t,p)$, peso del arco que comunica la transición t con la plaza p.
- ${}^\bullet t$, conjunto de lugares de entrada a la transición t.
- $t^\bullet$, conjunto de lugares de salida de la transición t.
- ${}^\bullet p$, conjunto de transiciones de entrada a la plaza p.
- $p^\bullet$, conjunto de transiciones de salida de la plaza p.
- $R(Mo)$, conjunto de posibles marcajes alcanzados desde Mo.
- $L(Mo)$, conjunto de posibles secuencias de disparo desde Mo.
- n, número de plazas en la RdP.
- m, número de transiciones en la RdP.

$C = \{C_{i,j}\}$, matriz de incidencia de nxm, tal que si se asumen todos los pesos como uno, entonces: $C_{i,j} = 1$ si la plaza i es una plaza de salida de la transición j, $C_{i,j} = -1$ si la plaza i es una plaza de entrada de la transición j ó $C_{i,j} = 0$ en otro caso.

Una representación gráfica de una RdP para un sistema de manufactura, se puede evidenciar en la figura 4:

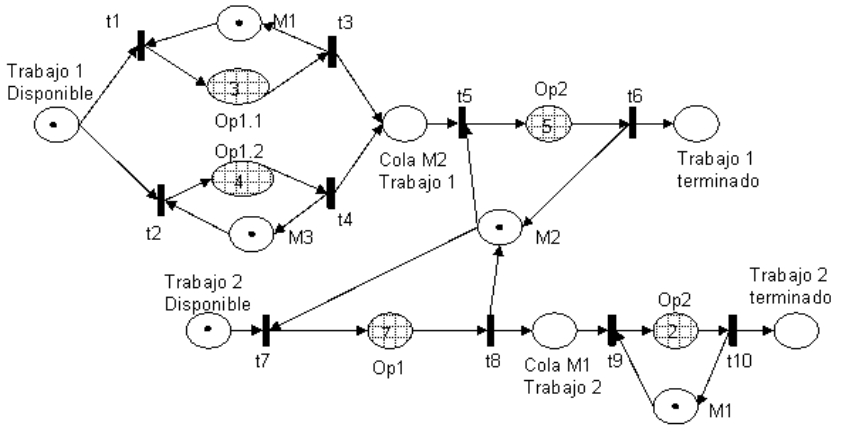


Figura 4. Modelo de RdP para un Job Shop con dos trabajos, tres máquinas y una secuencia de operaciones establecida. Fuente: Acevedo, J. y Mejía, G., 2005.

Las redes de Petri se clasifican en (Murata, 1989):

- Subredes: A su vez, estas se clasifican en: Máquinas de estado (SM) en donde cada transición tiene exactamente una plaza de entrada y exactamente una plaza de salida y los pesos de sus arcos siempre es 1, grafos de marcados(MG) en donde cada plaza tiene exactamente una transición de entrada y exactamente una transición de salida, redes de libre elección (Free-choice net) en donde cada plaza tiene un arco de entrada proveniente de una transición y tiene una arco de salida hacia una transición; redes de libre elección extendida (EFC) en donde existen dos o más plazas que tienen una misma plaza de salida y redes de elección asimétrica (AC) en donde existen dos o más plazas tales que la(s) transición(es) de salida de una es un subconjunto de la(s) transición(es) de salida de la otra.
- Temporizadas (Timed Petri nets o TPN): Son redes en las cuales se toma el tiempo en cuenta. Si los tiempos de disparo de las transiciones son determinísticos, la red es Temporizada, de lo contrario se considera estocástica. Los tiempos de los disparos pueden tomar alores enteros, en este caso las redes son llamadas Timed PN, y valores reales, llamadas Time PN.
- Estocásticas: Son redes de Petri en las que cada transición se asocia con una variable aleatoria distribuida exponencialmente que expresa el retraso entre la habilitación y el disparo de la transición. En el caso que se habiliten varias transiciones simultáneamente, la transición que tiene el menor tiempo de retraso se disparará primero
- De alto nivel: Estas se clasifican en Predicados/transiciones Redes (P/T N), Redes de Petri coloreadas (CPN) y Redes con tokens individuales.
- Puras: No tienen bucles.
- Ordinarias: Todos los pesos de los arcos son uno.
- Generalizadas: Los arcos pueden tener pesos mayores a uno.
- De capacidad finita: Cada una de las plazas puede tener sólo un número máximo de tokens

en cualquier instante.

- De capacidad Infinita: Las plazas no tienen número máximo de tokens en ningún instante.
- Autónomas: No se conoce o indica en qué momento se realizarán los disparos de las transiciones.
- Sincronizadas: Cuando los disparos van sincronizados con eventos externos.

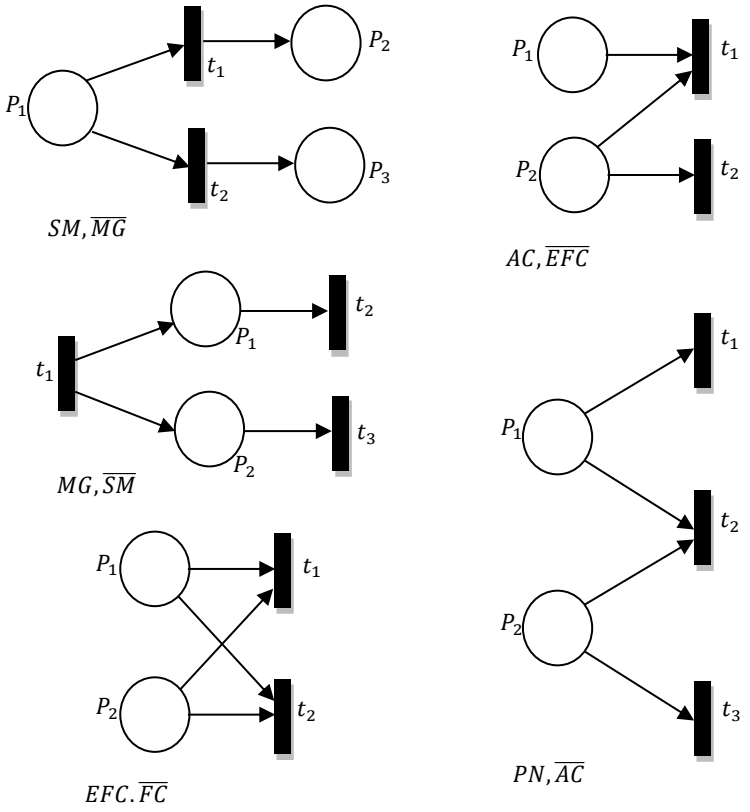


Figura 5. Subredes de Petri. Fuente: Murata, 1989.

La importancia de las Redes de Petri proviene del uso que hoy se les da en el estudio de espacios de búsqueda y su acotación. Además, entre los grandes beneficios de usar las Redes de Petri como herramienta de modelado de los sistemas se encuentra el conocimiento acerca del estado de éste en cada momento de su evolución por medio del uso de los vectores de marcajes. Se conoce como evolución de la red cada paso que da a través del tiempo, es decir, el movimiento de los tokens entre las plazas después de los disparos de las transiciones.

Se han identificado dos tipos de propiedades de las Redes de Petri, las que dependen del estado inicial o marcaje de la Red de Petri, llamadas conductuales, y las propiedades que dependen de la estructura de la red, llamadas estructurales. A continuación se describen las propiedades más importantes de la Red de Petri (Murata, 1989):

- Alcanzabilidad. La alcanzabilidad o accesibilidad es una base fundamental para el estudio de las propiedades dinámicas de cualquier sistema. El disparo de una transición habilitada cambiará el marcaje (Distribución de tokens) en una red de acuerdo a los vectores de marcaje. Una secuencia de disparos resulta en una secuencia de marcajes. El marcaje M' se dice que es inmediatamente alcanzable desde M si disparando una transición habilitada en M lleva a M'. Se usa R para representar el conjunto de todos los marcajes alcanzables desde Mo.
- Vivacidad. El concepto de vivacidad está cerradamente relacionado con la completa ausencia de puntos muertos (Deadlocks) en los sistemas. Una Red de Petri (N, Mo) se dice que está viva sí, no importa que marcaje ha sido alcanzado desde Mo, es posible disparar cualquier transición de la red a medida que se progresa a través de alguna secuencia de disparos.
- Reversibilidad. Una Red de Petri es reversible si, para cada marcaje M que pertenece a $R(Mo)$, Mo se puede alcanzar desde M. Por lo tanto, en una red reversible siempre se puede regresar al marcaje o estado inicial.
- Cobertura. Un marcaje M en una Red de Petri se dice que es factible de cobertura, sí existe un marcaje M' que pertenece a $R(Mo)$ que cumple con la condición $M'(p) \geq M(p)$ para cada plaza p de la Red de Petri.
- Persistencia. Una Red de Petri es persistente si para cualquier par de transiciones disponibles o habilitadas, el disparo de una de ellas no deshabilita la otra. Una transición que esté disponible en una Red de Petri persistente, se conserva disponible hasta que sea disparada.
- Acotamiento y Seguridad. Se dice que una Red de Petri está k-acotada si el número de tokens en cada plaza no excede el número entero finito k para cada marcaje alcanzable desde Mo, $M(p) \leq k$. Las redes 1-acotadas son conocidas como binarias. Una Red de Petri es segura si es 1-acotada.
- Distancia sincrónica: Es una métrica relacionada con el grado de dependencia mutua entre dos eventos en un sistema. La distancia sincrónica entre dos transiciones t_1 y t_2 en una Red de Petri (N, Mo) está dada por $d_{12} = max|\bar{\sigma}(t_1) - \bar{\sigma}(t_2)|$ donde σ es una secuencia de disparos que inicia en cualquier marcaje M en $R(N, Mo)$ y $\bar{\sigma}(t_i)$ es el número de veces que la transición $t_i (con\ i = 1,2)$ es disparada en σ para llegar de nuevo al mismo marcaje M (Murata, T., 1989).

Para realizar un análisis de las Redes de Petri se utilizan los siguientes métodos:

- El árbol de alcanzabilidad. El marcaje inicial Mo es la raíz del árbol de alcanzabilidad. Empezando desde la raíz se puede buscar por todas las transiciones activas, el disparo de una transición activa produce un nuevo marcaje el cual es representado como una nueva hoja o nodo del árbol, el procedimiento es iterativo. Identificando los nodos frontera del árbol, la

generación del árbol de alcanzabilidad envuelve un finito número de pasos, incluso si la Red de Petri no es acotada (Bobbio A., 1990).

- Análisis matricial: Las funciones de entrada y salida de una Red de Petri pueden ser representadas usando una notación matricial. Esta matriz es llamada matriz de incidencia y se representa simbólicamente por C. La matriz de incidencia está definida por la siguiente relación: $C = C^+ - C^-$, en donde C^- es la matriz de incidencia negativa o matriz de entradas y C^+ es la matriz de incidencia positiva o matriz de salidas. Las tres matrices corresponden a una matriz (nxm), en donde n es el número total de transiciones t_i y m el número total de plazas p_j. Para la matriz de incidencia negativa un elemento $c_{i,j}$ es igual al número de arcos que conectan la plaza j con la transición i y para la matriz de incidencia positiva un elemento $c_{i,j}$ es igual al número de arcos que conectan la transición i con la plaza j (Murata, T., 1989).
- Ecuación de estado: La ecuación de estado de una red de Petri nos indica el estado de la red en un determinado instante de tiempo, está definida inicialmente por la siguiente fórmula:

$$M_{k+1} = M_k + Cu_k$$

En donde M_k es el marcaje de la red en el tiempo k, C es la matriz de incidencia y u_k es el vector de transiciones disparadas. Sin embargo, otra ecuación de estado ha sido establecida para las redes de Petri temporizadas, en la cual se tiene en cuenta los tiempos de procesamiento remanentes, es decir, el tiempo que le falta a una operación para ser terminada. La ecuación se define por (Kordic, V., 2008):

$$X_{k+1} = A_k X_k + B_k u_k$$

En donde, u_k es el vector de control que determina cual transición es disparada en el instante k, X_k y A_k son matrices definidas a continuación:

$$X_k = \begin{vmatrix} Mp_k \\ Mr_k \end{vmatrix}$$

Mp_k es vector de marcaje después de k disparos de transiciones y Mr_k es vector de tiempos de procesamiento remanente después.

$$A_k = \begin{vmatrix} [I] & [0] \\ -\tau_k[P] & [I] \end{vmatrix}$$

$[0]$ Matriz de ceros.

$[I]$ Matriz idendtidad.

τ_k Tiempo del siguiente disparo.

$[P]$ Matriz diagonal que sirve para identificar plazas operacionales, plazas de recursos y plazas condicionales. $P_{ii} =$ {1 si la plaza es operacional, 0 en cualquier otro caso} y $P_{ij} = 0$ cuando $i \neq j$.

B_k es una matriz de distribución que dirige con el vector u_k la adición o remoción de tokens de las plazas.

$$B_k = \begin{vmatrix} C \\ [W] \quad C^+ \end{vmatrix}$$

$[W]$ es la matriz de los tiempos de procesamiento de las plazas operacionales.

El marcado de una red de Petri cambia después de cada disparo de una transición. Las restricciones de las redes de Petri son dadas por los invariantes de la red. Existen dos tipos de invariantes (Pawlewski, P., 2012):

- P-Invariante, también llamado marcado invariante es obtenido si el número de tokens de un subconjunto de plazas de la red se conserva. Las plazas contenidas en este subconjunto son llamados componentes conservativos y el conjunto de dichas plazas es el P-Invariante.
- Si después del disparo de una cierta secuencia de transiciones el marcaje resulta siendo el mismo del inicio, el grupo de dichas transiciones es llamado componente repetitivo. La secuencia de disparos del componente repetitivo es llamado el T-Invariante.

Las Redes de Petri son muy útiles para modelar los sistemas de manufactura ya que capturan las relaciones de precedencia e interacciones estructurales de eventos estocásticos, concurrentes y asíncronos, pueden modelar conflictos y tamaños de buffer, en un sistema se pude detectar un deadlock, representan una herramienta de modelado con buen fundamento matemático y práctico, los modelos de redes de Petri forman un marco estructurado para adelantar análisis sistemático de sistemas complejos y estos modelos también pueden utilizarse para implementar control en tiempo real de los FMS (Aized, T., 2010). Dentro de las aplicaciones de las Redes de Petri se encuentran las siguientes: evaluación de ejecución, protocolos de comunicación, modelado y análisis de software distribuido, sistemas de bases de datos distribuidas, FMS, sistemas de eventos discretos, sistemas de memoria multiprocesadores, sistemas de información de oficinas, lenguajes formales, programas lógicos, redes de área local, redes neuronales, filtros digitales, modelos digitales y otros más. Algunas de las aplicaciones realizadas de las Redes de Petri en los FMS son:

- Shih, H. y Sekiguchi, T. (1991), utilizaron una RdP temporizada y un método de búsqueda con inteligencia artificial para programar la producción en un FMS.
- Hatono I., Yamagata, K. & Tamura, H. (1991), emplearon una RdP estocástica para describir los eventos inciertos de comportamiento estocástico en los FMS, tales como fallas de herramientas de máquina, tiempo de reparación y tiempo procesamiento.
- Woo, Y., Suzuki, T. y Narikiyo, T., (2007), presentaron un nuevo método para la programación en los FMS basado en la modelación del sistema por medio de Redes de Petri y un algoritmo reactivo de búsqueda gráfica, buscando la minimización del makespan y de la tardanza total. Algunos experimentos numéricos se llevaron a cabo para demostrar la utilidad del algoritmo, en donde se obtuvieron soluciones semi-óptimas y factibles con pequeños esfuerzos computacionales.
- Un sistema integrado de Redes de Petri temporizadas (TPPN) fue utilizado por Mejía, G. y Poensgen, S. (2007), para modelar un FMS y realizar su programación óptima. Presentaron un prototipo que simulaba el plan de producción e implementaba reglas de prioridad de despacho (SPT, LPT, WSPT, EDD, CR, MS y ATC) para resolver conflictos eventuales (selección de trabajos en cola, prioridad de máquinas). Se comparan las diferentes reglas para escoger la mejor.

- Sharifi E., et al. (2008), presentaron igualmente el uso de RdP en la optimización del proceso de producción en FMS. Utilizan RdP con programación no lineal; consiguen minimizar el tiempo total de producción del FMS manteniendo las secuencias de operaciones y sincronizando los recursos compartidos. Se utiliza el software Netlab para la simulación de los eventos.

En los últimos años ha continuado la insistencia en el tema en especial en los campos de las aplicaciones, entre ellas, las Redes de Petri se han utilizado ampliamente para la programación y reprogramación de la producción en sistemas de manufactura. En un estudio realizado por Mejía (2004), se afirma que el proceso de reprogramación utilizando un sistema modelado por una Red de Petri es idéntico al de programación excepto que la RdP no se encuentra en su estado inicial. De esta manera, algunas máquinas pueden estar ocupadas y algunos trabajos se pueden encontrar ya en proceso. Se deben tener en cuenta los nuevos marcajes que se producen después del evento que dio pie a la reprogramación.

1.5. Algoritmos Meméticos (MA – Memetic Algorithms)

Los Algoritmos Meméticos (MA) se han definido como técnicas de optimización que combinan sinérgicamente conceptos tomados de otras metaheurísticas, tales como la búsqueda basada en poblaciones y la mejora local. La idea central de los MA está basada en mejoras individuales de las soluciones en cada uno de los agentes junto con procesos de cooperación y competiciones. Moscato, P. y Cotta, C., (2003) han remontado los orígenes de los Algoritmos Meméticos a finales de los años ochenta, cuando la computación evolutiva de donde parten las estrategias evolutivas y los algoritmos genéticos ya había empezado a afianzarse de forma sólida, al igual que técnicas de búsqueda local como el recocido simulado y la búsqueda tabú. El nombre de Memético proviene del término inglés *meme* introducido por Richard Dawkings para representar una unidad de evolución cultural. En el contexto de optimización un meme representa una estrategia de aprendizaje o de desarrollo.

Los algoritmos meméticos se han estudiado cada vez más en profundidad y hoy en día figuran como una estrategia muy utilizada para atacar el problema de la programación y reprogramación de los FMS. Los algoritmos meméticos pueden contribuir a la mejora de la programación en lo que tiene que ver con la obtención de soluciones óptimas o casi óptimas, evitando caer en mínimos locales o convergencia prematura (Neri et. al, 2012). Dicha convergencia se intenta reducir por medio del uso de técnicas de búsqueda local utilizadas en los algoritmos evolutivos que son considerados la base de los algoritmos meméticos (Garg, P., 2009).

Algunos de los estudios realizados utilizando algoritmos meméticos son:

- González, M., Vela, C., González, I. y Varela, R., (2006), trabajaron en un ambiente Job Shop y presentan un trabajo sobre la confrontación de los algoritmos genéticos hibridizados con un método de búsqueda local en un problema de programación. El

algoritmo genético busca sobre espacios de programas activos, mientras que el de búsqueda local lo hace sobre espacios semiactivos. Al mostrar sus resultados experimentales sobre un conjunto de casos- problema, concluyen que esta combinación de búsqueda de espacios es mejor que restringiendo ambos algoritmos a buscar sobre el mismo espacio.

- Marimuthu, S., Ponnambalam, S.G., y Jawahar, N., (2008), proponen dos algoritmos evolutivos: el algoritmo genético (AG) y el algoritmo evolutivo hibrido (AEH), con el fin de abordar un problema de programación y secuencia de toma de decisiones en un ambiente Flow Shop, los cuales simulan la mejor secuencia para el Makespan, criterio del tiempo total de flujo para 'm' maquinas en ambiente Flow shop relacionado con el lote calibrado y configuración de secuencias de tiempo.
- Kamrul, S., et al. (2011) estudiaron el problema de programación reactiva en un job-shop considerando fallas y no disponibilidad de máquinas, utilizando un algoritmo híbrido (Un algoritmo genético con una técnica de búsqueda local, llamada Shifted Gap-Reduction). Basándose en sus resultados experimentales, concluyen que el instante en que ocurre la falla de la máquina influye sobre el makespan del programa reactivo; cuando las interrupciones se presentan en las primeras etapas de la solución, las tareas afectadas pueden ser reprogramas con un pequeño incremento del makespan. Por otro lado, con tiempos de reparación pequeños es relativamente más fácil de minimizar el makespan, comparado con un solo tiempo largo, incluso si la suma de los tiempos pequeños es una cifra muy aproximada al tiempo extenso.

1.5.1. Algoritmos Genéticos

Como los algoritmos meméticos parten de un algoritmo genético, este juega un papel importante en su estructura. Constituyen la herramienta de búsqueda global aportando el cubrimiento de la diversidad mientras que la búsqueda local aporta la intensificación.

Los AG hacen parte de los algoritmos evolutivos que constituyen una técnica general de resolución de problemas de búsqueda y optimización (Araujo, L. y Cervigón, C., 2009). La manera en que trabajan los algoritmos evolutivos está relacionada en la forma en que evolucionan las especies de la naturaleza. Utilizan un conjunto de posibles soluciones o individuos a los que le calculan su medida de adaptación. A través de un proceso iterativo esta población va cambiando y cada iteración es llamada generación. Para que un individuo sobreviva y pase a la siguiente generación debe tener un gran nivel de adaptación y participar en las operaciones genéticas, con las que se crean nuevos individuos que constituyen la siguiente generación. Estos algoritmos permiten abordar problemas de gran complejidad de búsqueda y optimización como planificación de tareas, horarios y tráfico, búsqueda de caminos óptimos, entre otros.

Entre los algoritmos evolutivos más populares están los algoritmos genéticos que deben su éxito a su eficiencia y sencillez de implementación. Se caracterizan por representar las soluciones en forma de cadena de bits, que luego deben ser decodificadas de acuerdo al problema que abordan.

Kumar Manoj et al.(2010) en su revisión de los algoritmos genéticos para la programación de la producción, los caracteriza como un proceso estocástico que genera una población inicial de programas y luego principios de selección y supervivencia naturales son aplicados para mejorar dichos programas.

Los algoritmos genéticos en general tienen un funcionamiento representado mediante el siguiente pseudocódigo:

1. Selección de la población inicial de individuos
2. Evaluación de la adaptación de cada individuo en esa población
3. Repetir en esta generación hasta terminar (límite de tiempo, consecución del fitness deseado, etc)
    a. Seleccionar los individuos de mejor fitness para reproducción
    b. Generar nuevos individuos a través de las operaciones de crossover y mutación para dar nacimiento a los hijos
    c. Evaluar el fitness particular de los nuevos individuos
    d. Remplazar la población con fitness menores con los nuevos individuos

El comportamiento de los algoritmos genéticos está definido por una serie de parámetros que son valores de entrada para éste; entre los parámetros encontramos el tamaño de la población inicial, el número de generaciones, los porcentajes de mutación y crossover, entre otros. Después de que se han definido todos los parámetros necesarios, se empieza por generar una población inicial, en donde cada individuo es una posible solución del problema estudiado. A continuación, esta población es sometida a un ciclo de evolución en donde se le aplican operadores genéticos con el fin de generar nuevos individuos que formarán cada generación hasta terminar el ciclo. Dicho ciclo finaliza generalmente cuando se ha llegado hasta el número de generaciones requeridas (parámetro). Sin embargo, existen otras condiciones que finalizan este ciclo, como un tiempo límite o consecución del fitness requerido.

1.5.1.1. Componentes de un algoritmo genético

Los individuos de un algoritmo genético son cadenas binarias, se denotan por **b**, y representan a los puntos **x** del espacio de búsqueda del problema. Teniendo en cuenta la nomenclatura de la biología, a **b** se le denomina *genotipo* o *cromosoma* del individuo y a **x** se le denomina *fenotipo*. Por lo tanto, se usa *gen* para referirse a la codificación de una determinada característica del individuo. En los algoritmos genéticos se suele identificar un *gen* con cada posición de la cadena binaria, aunque no siempre es así. Se usa *alelo* para los diferentes valores que puede tomar un gen y *locus* para referirse a una determinada posición de la cadena binaria (Araujo, L. y Cervigón, C.,

2009). Otro método para la representación de los genotipos es codificar las soluciones por medio de cadenas de enteros o números decimales, en donde cada posición representa algún aspecto particular de la solución. Este método permite una mayor precisión y complejidad que el método comparativamente limitado de utilizar sólo números binarios, y a menudo "está intuitivamente más cerca del espacio de problemas" (Fleming, P., y Purshouse, R.C., 2002). Un tercer método consiste en codificar a los individuos por medio de cadenas de letras o combinación de estas con números, donde cada letra o número representa nuevamente un aspecto específico de la solución (Mitchell, M., 1996). Las ventajas de estos métodos es la facilidad de declarar operadores que generen los cambios aleatorios en los candidatos seleccionados.

Generalmente la población inicial utilizada en un algoritmo genético es generada de forma aleatoria. Sin embargo, si se posee información de individuos que tienen más probabilidades de llegar a ser solución, pueden ser parte de la población inicial, teniendo en cuenta que debe existir variedad de soluciones para poder explorar en todas las zonas del espacio de búsqueda. Cada individuo creado debe ser evaluado y se le debe asignar un valor de adaptación también conocido como *fitness*. El fitness es una medida que indica como el individuo optimiza la función. En otras palabras, el valor fitness es usado para determinar la probabilidad de selección de cada cromosoma (Lawrynowicz, A., 2011).

Como en todo proceso natural la creación de un individuo o hijo, implica la selección de dos padres. Este proceso de selección de padres puede ser llevado a cado de diferentes maneras, entre las más comunes tenemos:

- Selección por torneo: Inicialmente, se escoge una muestra de la población de manera aleatoria y es seleccionado el individuo con mejor fitness, generalmente ésta muestra es de dos individuos. Lo anterior, se repite hasta conseguir el número de individuos deseado. Este proceso de selección puede ser determinista o probabilístico en donde un individuo con mal fitness tiene probabilidad de ser seleccionado. (Araujo, L. y Cervigón, C., 2009).
- Selección por ruleta: La probabilidad de un cromosoma para ser seleccionado está estrechamente relacionada y es proporcional a su fitness (Lawrynowicz, A., 2011). Según (Araujo, L. y Cervigón, C., 2009) la probabilidad de que el individuo i sea seleccionada está dada por la fórmula:

  $$p_i = \frac{f(i)}{\bar{f}}$$

  Siendo $\bar{f}$ el valor promedio de los fitness de toda la población. Para proceder con este método de selección se necesita generar un número aleatorio a en el intevalo [0,1] luego de haber obtenido las puntuaciones acumuladas de la siguiente forma:

  $q_0 \coloneqq 0$

  $q_i \coloneqq p_1 + p_2 + \cdots + p_i \; \forall i = 1,2 \ldots, n$

  Se selecciona el individuo que cumpla:

  $q_{i-1} < a < q_i$

Este proceso se repite hasta que se complete el número de individuos que se desean seleccionar.

Después de haber seleccionado los padres de un individuo se aplican ciertos operadores genéticos. Dos operadores que generalmente están presentes en todo el algoritmo, son el cruce o crossover y la mutación. Entre los parámetros que se definen en el algoritmo, están los porcentajes de cruce y de mutación, esto con el fin de aplicar cada uno de estos operadores sólo si un valor generado aleatoriamente está por encima de dichos porcentajes.

- **Cruce o crossover:** Implica seleccionar dos individuos para que intercambien segmentos o partes de su código, produciendo una descendencia artificial cuyos individuos resultantes o hijos, son combinaciones de los individuos que se cruzan llamados padres. Las formas comunes de cruzamiento son: Selección de uno o dos puntos de corte (ver figura 6), Secuencias PMX (Partially Matched Crossover), OX (Ordered Crossover) y CX (Cycle Crossover) (Acevedo, J. y Mejía, G., 2006). En muchos estudios se coincide que el desempeño de los algoritmos genéticos está muy relacionado con el operador de cruce, afirman que si se utiliza un porcentaje de cruce alto, se tiene la posibilidad de explorar mejor el espacio de solución del problema, minimizando así la posible caída en mínimos locales (Gen, M. y Cheng, R., 1997).

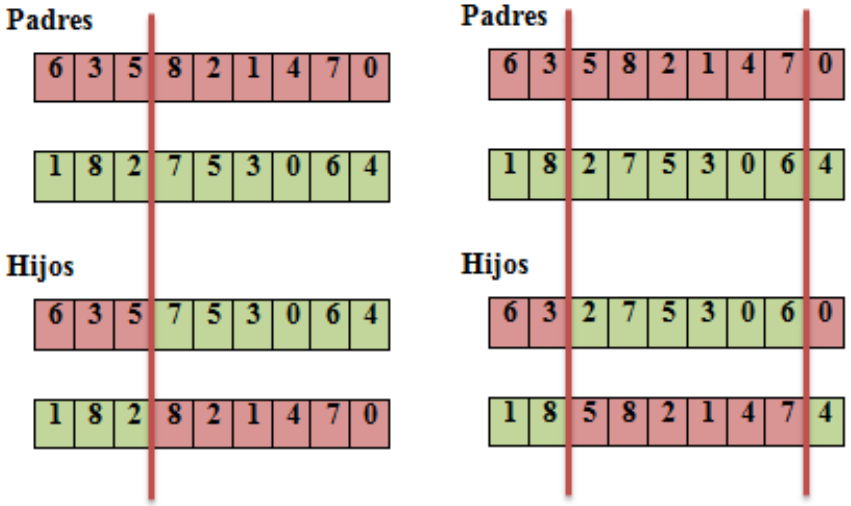


Figura 6. Cruce en un punto y dos puntos para cadenas de números enteros.

- **Mutación:** Es usada para producir perturbaciones sobre los cromosomas con el fin de mantener la diversidad en la población. Existen dos tipos de mutación principales: Mutación por inversión en la cual se invierten dos posiciones escogidas aleatoriamente y mutación por inserción en donde de igual forma dos elementos son escogidos aleatoriamente y se inserta uno detrás del otro. Mientras la mutación por inversión sirve para mantener la diversidad, la mutación por inserción no solo produce pequeñas perturbaciones, sino que también desarrolla búsquedas intensivas con el fin de encontrar mejores hijos (Lawrynowicz, A., 2011). En otros estudios, se han encontrado operadores de mutación en donde aleatoriamente es escogida una posición (gen) y se cambia su valor por otro dentro del rango de posibilidades

con el fin de generar una perturbación. Un porcentaje de mutación es utilizado como parámetro del algoritmo; si éste es muy bajo, se pierde la posibilidad de explorar genes útiles para mejorar el desempeño de los cromosomas, y si es muy alto proporcionará demasiada variabilidad (Mejía, G. 2004).

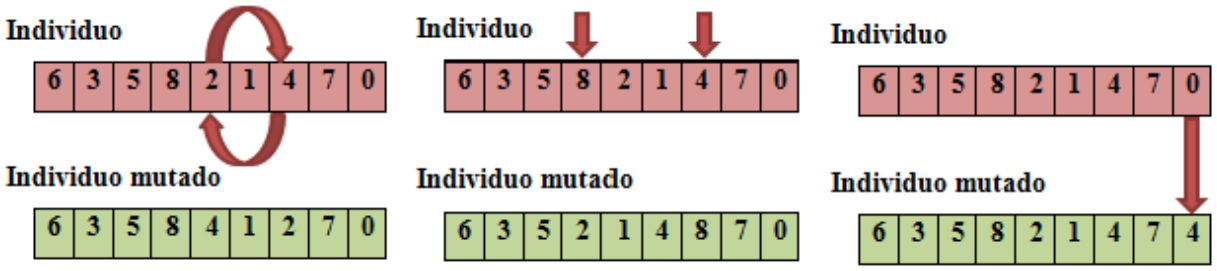


Figura 7. Mutación por inversión, insersión y cambio del valor del gen.

1.5.2. Búsqueda local

La búsqueda local es un proceso mediante el cual se busca en el espacio de soluciones candidatas a un problema dado. Se inicia seleccionando un solución al azar y a partir de ella se buscan soluciones vecinas de forma iterativa; se observa entre ellas cuál es la mejor de acuerdo con una función de evaluación en la vecindad escogida. Esta evaluación se puede hacer varias veces fijando un parámetro para ello y de acuerdo con una historia que se puede llevar en memoria de las soluciones candidatas visitadas (Hoos, H., 2005).

La búsqueda local tiene como componentes:

- Un espacio de búsqueda que es un conjunto finito de soluciones candidatas también llamadas estados, configuraciones, ubicaciones o posiciones de búsqueda.
- Un conjunto de soluciones factibles.
- Una relación de vecindad.
- Una función de inicialización, que se hace con base en distribuciones de probabilidad sobre posiciones iniciales de búsqueda.
- Una función de paso, que lleva cada posición de búsqueda a una distribución de probabilidad sobre su vecindad.
- Una condición de finalización.

Por su parte, la vecindad de una solución “S” está formada por todas las posibles soluciones de S que cumplan la relación dada. Una de estas posibles relaciones es la llamada one-flip (una vuelta), que define dos soluciones como vecinas si y solo si difieren sólo en el valor de una variable, quedando las demás iguales.

A través de los años y estudios se han definido muchas relaciones de vecindad. Entre las más comunes aparecen:

- K-exchange: Dos soluciones candidatas son vecinas si y sólo si difieren únicamente en k componentes.
- One-flip: En este caso, la diferencia se da sólo en un valor de verdad de una variable mientras las demás permanecen iguales.

Para abordar la búsqueda local, es necesario antes definir algunos conceptos que forman parte del argot propio de su estructura y análisis.

- Perturbación vs Construcción: El algoritmo puede perturbar una solución candidata (perturbativo) o crear soluciones (constructivo).
- Búsqueda Sistemática o local: En la forma sistemática, se examina todo el espacio de búsqueda, mientras que en el local, se empieza por algún lugar del espacio y se sigue examinando vecindades pequeñas. A la sistemática se le dice entonces completa y a la local incompleta.

Para la búsqueda local, la inicialización consiste en partir de una solución factible que se escoge al azar. Para ello se puede utilizar en general una cualquiera de las dos estrategias: Uninformed Random Picking que rastrea aleatoriamente todo el espacio de búsqueda para tomar una solución, o Uninformed Random Walk que lo hace sobre una vecindad reducida.
Otra estrategia es el Iterative Improvement o Mejoramiento Iterativo que empieza de un punto del espacio de búsqueda elegido al azar y rastrea la vecindad en búsqueda de mejores vecinos a través de la función de evaluación o fitness; también se llama Hill Climbing (o ascenso a la colina) o Iterative Descent. El problema que presenta consiste en que si en la vecindad definida no hay mejora, puede terminar el proceso de manera no satisfactoria. Podemos encontrar mínimos locales entonces que no son globales y que no permiten llegar a la solución quedando así el proceso "estancado". Para salir del problema, sólo las relaciones que contemplen toda la vecindad pueden garantizar una solución óptima, real. Sin embargo, el tamaño de la vecindad en estos casos crece exponencialmente haciendo imposible su rastreo eficiente en la práctica. Para ello entonces, se contemplan soluciones viables como el recorte (prunning) de las vecindades y el uso de algoritmos de alta eficiencia bien comprobados. También se usan en ellos las siguientes estrategias de escape.

Para escapar de los mínimos locales se puede utilizar: el reinicio cada vez que se encuentre un mínimo local o la relajación del criterio de terminación. Sin embargo, ellas no garantizan totalmente el éxito, pues se puede caer en un plateau o meseta donde todos los valores son iguales, creyendo así que se obtuvo la solución ideal sin saber que existen otras soluciones mejores al salir del plateau.

A pesar de lo anterior, a favor de estas estrategias y métodos que siguen se puede alegar que para problemas de optimización grandes o con grandes restricciones de tiempo y otros recursos, en muchos casos representan las únicas formas de hallar soluciones de alta calidad.

Entre los métodos propios de la búsqueda local se encuentran:

- Iterative Improvement: Estos métodos buscan utilizar grandes vecindades, ojalá exactas, pero que luego son sometidas a prunning (corte), omitiendo el análisis de vecinos que no conducen a mejoramiento. Entre las técnicas de prunning están: lista de candidatos y en segundo lugar el análisis de las propiedades del problema en estudio para rechazar los vecinos que no lleven a mejoramiento con base en la función de evaluación.
- Reglas de Pivote: Estas reglas son: Iterative Best Improvement (mejoramiento iterativo del mejor) y First Improvement (mejoramiento del primero). En el primer caso se evalúa todos los vecinos en cada paso. En el segundo caso, se evita evaluar todos los vecinos, seleccionando el primero que sea el mejor; si la evaluación se hace con un orden fijo en cada paso, se llega a los mismos vecinos por lo que se utiliza entonces el Random Order First Improvement. Esto permite obtener diferentes óptimos locales proporcionando así mayor diversificación. Cabe anotar que el tiempo de computación es menor para First Improvement porque corta el proceso de búsqueda más rápidamente.
- Variable Neigbourhood Descent-VND: Cada vez que se obtiene un mínimo local, se amplía la vecindad hasta un número k de pasos. Tiene como ventaja que mejora en la solución y en el tiempo para hallarla.

Adicional a los métodos anteriormente mencionados, también podemos encontrar en la literatura otras estrategias de mejoramiento como VDS (variable depth search o búsqueda de profundidad variable) y Dynasearch.

Entre los métodos más antiguos se encuentra el Branch and Bound. Este método fue utilizado en el año 2007 por Gallardo, J., Cotta, C. y Fernández, A. Sin embargo, teniendo en cuenta que los algoritmos B&B son conflictivos en cuanto a requerimientos de tiempo y memoria, optaron por una búsqueda exacta truncada, Beam Search. Generaron así un algoritmo híbrido de naturaleza heurística. Lo aplicaron luego a los problemas de la “Mochila” y el de “la supersecuencia común más corta”; los resultados fueron mejores en ambos casos. El B&B identifica las regiones del espacio de búsqueda probadamente buenas y el MA las explora.

En los casos anteriores se trabaja con la vecindad para obtener mejores resultados. Sin embargo, existen otros métodos que trabajan modificando la función de paso; se les llama simples porque trabajan con un solo tipo de paso de búsqueda entre los que podemos encontrar:

- Randomised Iterative improvement (RII): Se busca un vecino al azar en vez de uno mejorado; se determina un patrón de intercambio para intercalar búsquedas al azar con frecuencia determinada con los pasos previstos de mejoramiento.
- Probabilistic Iterative Improvement (PII): Para este caso, se establecen probabilidades de tal manera que los pasos que conllevan soluciones peores tiene bajos valores de las mismas.
- Otros más conocidos de amplio uso como Simulated Annealing (SA o recocido simulado), Tabu Search (TS, búsqueda Tabu), Dynamic Local Search (DLS, o búsqueda local dinámica)

y Hill Climbing (HC, escalado de montaña).

Debido a la complejidad de ciertos problemas se han construido métodos de búsqueda local híbridos, que como su nombre lo indica, son combinación de uno o varios métodos simples. Entre ellos, se tienen:

- Grasp (Greedy Randomised Adaptive Search Procedure, procedimiento de búsqueda ávido aleatorio adaptativo): A partir de una solución vacía, en cada paso se van agregando componentes de la solución para luego hacer la perturbación. Tiene en contra que sólo aporta un número limitado de soluciones candidatas.
- Aics (Adaptive iterated Construction Search, búsqueda constructiva iterada adaptativa): En éste método se utilizan pesos asociados con las decisiones posibles que se toman en el proceso de construcción; estos pesos se adoptan a través del paso de múltiples iteraciones con base en la experiencia ganada en ellas.

1.5.2.1. Recocido Simulado (SA – Simulated Annealing)

Fue propuesto por Kirkpatrick, Gelatt and Vecchi (1983) y Cerny (1985) motivado por el recocido de los sólidos, un proceso en el cual lo sólidos son calentados y posteriormente enfriados lentamente con el fin de obtener perfectas estructuras cristalinas.

La principal idea del recocido simulado consiste en la analogía observada entre optimización de un sistema complejo y la descripción del comportamiento físico de un sistema. Éste método es aplicado en muchos dominios: investigación operativa, programación de la producción y muchos otros (Ghoul, R., et al, 2007).

SA puede considerarse como un proceso en el que dada una estructura de vecindad, se intenta moverse de una solución actual a uno de esos vecinos. SA genera una solución nueva S' en la vecindad de la solución actual, luego calcula el cambio d = C(S')-C(s), siendo C(s) el valor de la función objetivo de la solución inicial (Kim, M. y Park, Y., 1998).

Los algoritmos de recocido simulado han sido exitosamente aplicados en varios problemas de dificultad combinatoria, Para aplicar un SA a un problema específico, se debe diseñar el algoritmo por medio de determinados métodos para representar soluciones, generar soluciones vecinas y reducir la temperatura, entre otros, y luego seleccionar los parámetros a usar en el algoritmo (Kim, M. y Park, Y., 1998).

En problemas de minimización (como el del makespan) se tiene que:

- Si d<0, se acepta la transición a la nueva solución (downhill).
- Si d>=0, se acepta con una probabilidad previamente especificada, la cual se obtiene con la función Exp(-d/T), donde T es un parámetro de control llamado la Temperatura (término

traído de la metalurgia). Esto le permite al SA escapar de mínimos locales al buscar un mínimo global.

El parámetro T disminuye gradualmente por una función de enfriamiento hasta que se satisface una condición de parada.

El procedimiento del SA es (Hoos, H y Stultze T., 2005):

Inicio

Escoger una solución inicial *S;*

Seleccionar la temperatura inicial y la temperatura final $T_0, T_f > 0$;

Mientras la condición de terminación no sea satisfecha **Hacer**

Generar un vecino *S'* de *S*

Si *S'* satisface el criterio de aceptación

S = S'

Retornar T de acuerdo con el criterio de decrecimiento

Note: El algoritmo puede tomar T como número de pasos de búsqueda.

Algunos de los estudios realizados utilizando SA son:

- Ghoul, R., et al. (2007). desarrollaron un algoritmo SA con el fin de realizar la programación de un sistema hibrido de manufactura flexible. Utilizaron redes de Petri coloreadas para modelar los sistemas. El algoritmo fue aplicado en los sistemas para resolver conflictos y compartir recursos. Sin embargo, encontraron que con la aplicación de este algoritmo se tiene una gran probabilidad de caer en óptimos locales.
- Varadharajan, T.K. et al. (2005). presentaron un algoritmo de recocido simulado multiobjetivo llamado MOSA (Por sus siglas en inglés) para resolver el problema de programación de los flowshops con el objetivo de minimizar el makespan y el tiempo total de flujo de los trabajos. El algoritmo encuentra buenas respuestas en menos tiempo computacional que los demás trabajos con los que fue comparado.

2. ALGORITMO PetNMA

El problema de programación reactiva debido a fallas de máquinas en un ambiente de sistema de manufactura flexible, se declara formalmente como sigue:

- El FMS está compuesto por, un conjunto de máquinas $M^T = \{m_1, m_2, ..., m_k, ..., m_M\}$ y un conjunto de órdenes de trabajo $J = \{J_1, J_2, ..., J_N\}$.
- Cada orden de trabajo J_j tiene una fecha de entrega d_j comprometida con el cliente, una prioridad definida según su nivel de importancia w_j y se compone de una secuencia de operaciones $O_j = \{O_{1,j}, O_{2,j}, ..., O_{i,j}, ..., O_{n_j,j}\} \quad \forall j$.
- Cada máquina m_k, $\forall k$ puede procesar sólo una operación al tiempo con posibilidad de falla $P(falla)_k$ en el horizonte de programación y en cualquier instante solamente puede estar procesando una operación de cada trabajo.
- Se asume que todos los $p_{i,j,k}$, d_j y w_j son determinísticos, así como también que todas las ordenes de trabajo están disponibles cuando se hace su programación y el tiempo de procesamiento $p_{i,j,k}$ incluye el tiempo de preparación o alistamiento sobre la máquina.
- k representa el índice de máquinas $(k = 1,2,..M)$, j el índice de órdenes de trabajo $(j = 1,2,..N)$, i el índice de operaciones $(i = 1,2,...n_j)$, S el índice de estaciones $(S = 1,2,...,s)$.
- Cada estación tiene un espacio a la entrada que se utiliza para el almacenamiento temporal de todos los trabajos que esperan ser procesados en ella.
- Cuando un trabajo es procesado completamente en una estación, este se traslada al espacio de entrada de la estación donde se debe ejecutar su siguiente operación o al sitio donde se almacenan los trabajos terminados.
- La interrupción de la programación se genera debido a fallas en las máquinas y no disponibilidad de éstas.
- En la programación reactiva las operaciones que se encontraban en procesamiento sobre una máquina que no presenta falla, no es interrumpida hasta tanto no termine. Cualquier operación que se esté procesando sobre una máquina que presente falla, se daña y tocará reprogramarla.
- Se conforma el conjunto de operaciones a reprogramar con: las operaciones de los trabajos que no se han realizado al momento en que sucede la falla o la no disponibilidad de la máquina, considerando a su vez que aquella operación que al momento de la ruptura del

programa original estaba en procesamiento sobre la máquina, al dañarse, toca iniciarla nuevamente.

- El objetivo de la programación es minimizar el Makespan, la Tardanza Total Ponderada y el objetivo ponderado en la programación del sistema de manufactura flexible. Al denotar como C_j como el tiempo de terminación de una orden de trabajo, los objetivos planteados se expresan como: C_{max} , $\sum_{j=1}^{N} w_j T_j$ y $\alpha C_{max} + \beta \sum_{j=1}^{N} w_j T_j$ en donde $\alpha + \beta = 1$, respectivamente.

El algoritmo PetNMA se fundamenta en una serie de subalgoritmos que se desarrollan en secuencia (Figura 8):

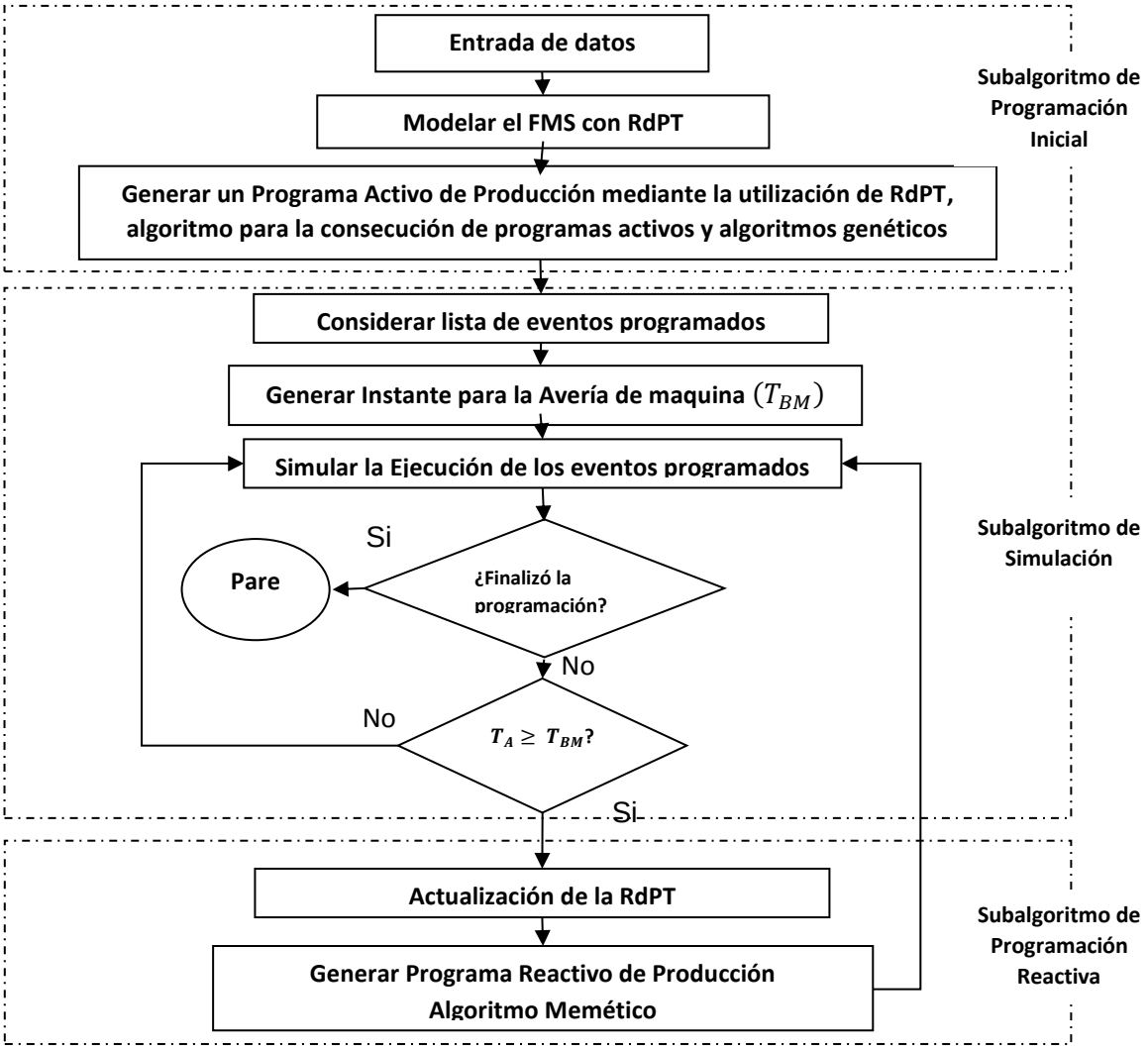


Figura 8. Algoritmo PetNMA: Subalgoritmos en la Programación Reactiva de la Producción de un FMS con Falla de máquinas.

- SubAlgoritmo para estructurar una programación de producción inicial en un FMS, con el conjunto de trabajos $J=\{J_1,J_2,...,J_N\}$, mediante la utilización del modelado de Redes de Petri Temporizadas, el algoritmo para la obtención de programas activos y los algoritmos genéticos.
- SubAlgoritmo para la simulación de la corrida del programa de producción inicial en el FMS, junto con las interrupciones aleatorias que representan las averías de las máquinas.
- SubAlgoritmo para la programación reactiva de la producción de las operaciones que no han sido procesadas al momento en que se presenta la avería y aquella operación que estaba siendo procesada en la máquina que presenta la avería y que toca procesarla nuevamente.

El SubAlgoritmo de Programación se fundamenta en la creación del FMS en el software LEKIN generando los archivos user.job y user.mch, los cuales posteriormente son leídos por PetNMA para construir la RdPT que servirá para construir el programa inicial de producción, mediante el establecimiento de la secuencia de disparo de las transiciones.

Durante el Subalgoritmo de Simulación se genera un instante para la avería de la máquina (T_{BM}); después de que cada orden de trabajo es procesada se realiza una comparación entre el tiempo de reloj (T_A) y el instante generado para la avería de la máquina, si T_A es mayor al instante T_{BM} significa que la interrupción ha sucedido y toca realizar la Programación Reactiva de la Producción por medio del Subalgoritmo de Programación Reactiva.

La notación utilizada para representar matemáticamente lo que sucede con el conjunto de operaciones que entran a la programación reactiva de la producción es la siguiente:

- $X_{i,j,k}$ Tiempo en que inicia la operación i $(i=1,2,...)$ de la orden de trabajo j programada sobre la máquina k en el programa inicial.
- $P_{i,j,k}$ Tiempo de procesamiento de la operación i $(i=1,2,...)$ de la órden de trabajo j sobre la máquina k.
- $C_{i,j,k}$ Tiempo de terminación de la operación i $(i=1,2,...)$ de la órden de trabajo j sobre la máquina k.
- $O_{i,j,S}$ i-ésima operación de la órden de trabajo j sobre una máquina de la estación S.
- $O^R_{i,j,k}$ i-ésima operación de la órden de trabajo j sobre la máquina k.
- d'_j Fecha de entrega de la órden de trabajo j que pertenece al conjunto de los trabajos en J_N y que entran en la programación reactiva.
- T^R_k Tiempo de disponibilidad de la máquina k para la programación reactiva

de la producción.

- α Porcentaje del makespan en el objetivo ponderado.
- β Porcentaje de la tardanza total ponderada en el objetivo ponderado.
- $CONP$ Conjunto de operaciones aún no programadas.

El Algoritmo Pentama fue desarrollado en una plataforma de programación Visual Studio 2010 con lenguaje de programación C++.

2.1 Subalgoritmo de programación inicial en PetNMA

Este subalgoritmo de programación inicial está basado en el subalgoritmo de Programación del Algortimo Afs PetriNet el cual se fundamenta en la creación del FMS con el software LEKIN, generando los archivos user.job y user.mch, los cuales posteriormente son leídos por el Algoritmo PetNMA para representar al FMS a través de una RdPT, con la cual se construye el programa inicial de producción, mediante el establecimiento de la secuencia de transiciones disparadas según una de las siguientes reglas de despacho: MRWT (Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo remanente de proceso) – LPT (Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo de proceso) – SPT (Cuando una máquina está disponible, se programan primero los trabajos con menor tiempo de proceso); este proceso da como resultado programas semiactivos de Programación de la Producción.

Sin embargo, el subalgortimo de Programación de PetNMA tiene como objetivo la obtención de programas activos de producción, es por eso que no utiliza las reglas de despacho para el establecimiento de la secuencia de disparo de las transiciones, en el subalgoritmo de Programación de PetNMA se utiliza el siguiente algoritmo para la obtención de programas activos (Pinedo, p. 433):

P1. Calcular el conjunto ***C*** de todas las operaciones que puedan programarse (todas sus precedencias satisfechas).
P2. Calcular el tiempo de terminación de cada operación en ***C*** y sea ***m**** la máquina en la cual ocurre el mínimo tiempo de terminación ***f***
P3. Sea ***G*** el conjunto de operaciones que pueden iniciarse en la máquina ***m**** y cuyo tiempo de ***inicio*** es menor a ***f***
P4. Seleccionar una operación de G y programarla;
P5. Eliminar la operación seleccionada de ***C*** y volver al paso 1

Es de notar que en el paso 4 se puede establecer una estrategia de selección de las operaciones dentro del grupo G. Para el presente estudio se utilizan reglas de despacho SPT, LPT, MRWT, EDD, CR, ATC y SNACK y se obtienen siete programas iniciales. Sin embargo, los programas activos no

siempre son los mejores, en ocasiones no están tan cerca del óptimo. Es por esto que se ha utilizado un algoritmo genético con el fin de obtener un séptimo programa inicial. De acuerdo a su valor objetivo se escoge el menor y éste se tiene en cuenta para realizar la programación inicial de las operaciones y su posterior simulación.

A continuación se presentan los pasos detallados del SubAlgoritmo de Programación:

P1. [*Crear el FMS en el software* "LEKIN – Scheduling Systems"]:

P1.0 Introducir en el software LEKIN toda la información del FMS.

P2. [*Crear* RdPT *del FMS*]

P2.0 Leer los archivos user.job y user.mch generados por el software LEKIN.

P2.1 Crear el conjunto finito de plazas $P=\{p_1, p_2,...,p_n\}$, a partir del número total de órdenes de trabajo a programar "No" (por cada orden de trabajo hay una plaza para representar, la orden de trabajo terminada), el número de estaciones de trabajo "S" (por cada estación de trabajo se crea una plaza para el almacenamiento de la orden de trabajo que requiere ser procesada en ella), el número total de máquinas en el sistema "M" (por cada máquina en el sistema se crea una plaza para representar su disponibilidad y una para representar su reparación después de ocurrida una falla) y el número de máquinas que pueden procesar cada operación de cada trabajo "$m_{i,j}$" (cada operación que llega a una estación tiene la posibilidad de ser procesada en alguna de las máquinas de la estación, por lo cual se crean plazas para representar en que máquina se puede procesar la operación). El número total de plazas de la RdPT se calcula por: $n = No + S + 2M + \sum_{j=1}^{No}\sum_{i=1}^{n_j} m_{i,j}$.

P2.2 Crear el conjunto finito de transiciones $T=\{t_1, t_2,...,t_m\}$, teniendo en cuenta el número total de máquinas que pueden procesar cada operación de cada trabajo y que cada máquina que esté procesando una operación puede presentar una falla, cada operación que represente la acción de procesamiento sobre una máquina implica la creación de tres transiciones y además cada máquina que falla debe habilitarse nuevamente después de terminarse su tiempo de reparación lo que implica la creación de un transición por cada máquina del sistema, por lo que $m = 3\sum_{j=1}^{No}\sum_{i=1}^{n_j} m_{i,j} + M$.

P2.3 Establecer el conjunto de arcos dirigidos desde T hasta P y el conjunto de arcos dirigidos desde P hasta T.

P2.4 Establecer el vector que representa el marcaje Mp para la RdPT, (0…*n*-1), inicializándolo con el marcaje inicial Mo de la red y teniendo presente que todas las plazas que representan la disponibilidad de las máquinas y las plazas de almacenamiento de la primera estación por donde inicia una orden de trabajo, tienen un token y las demás plazas ninguno.

P2.5 Establecer el vector "PT" equivalente al vector de tiempos de las plazas de la RdPT. Los tiempos asociados a las plazas de las máquinas, almacenamiento a la entrada de las estaciones y

terminación de la orden de trabajo, siempre son cero (0). Los tiempos asociados a todas las plazas de procesamiento de la operación sobre una máquina, son los tiempos de procesamiento de la operación en la estación a la cual pertenece la máquina y los asociados a las plazas de reparación de las máquinas se inicializan con un tiempo de procesamiento elevado para ser cambiado después de generada la falla por el tiempo de reparación correspondiente.

P2.6 Inicializar en cero (0) el vector de tiempos remanente para la posibilidad de disparo de la transición de salida de las Plazas "Mr".

P2.7 Establecer el vector asociado con el tiempo de trabajo remanente "rwt".

P2.8 Establecer la matriz de incidencia negativa $C\min$ y la matriz de incidencia positiva $Cplus$ de la RdPT.

P3. [*Generar el Programa de Producción Inicial mediante el algoritmo de programas activos*]

P3.0 Establecer el vector de transiciones que están habilitadas para ser disparadas.

P3.1 Seleccionar transición a disparar

P3.1.1 Calcular los tiempos de terminación de las operaciones disponibles para ser programadas. Organizar dichos tiempos de mayor a menor y guardarlos en una matriz junto con las transiciones que darían inicio a las operaciones.

P3.1.2. Realizar asignaciones de los recursos disponibles (Máquinas) teniendo en cuenta primero las operaciones que terminarían más tarde con el fin de tener una holgura en la programación. Para esto, se crea una matriz en donde se le asigna a cada máquina una operación a desarrollar junto con sus tiempos de terminación y de inicio.

P3.1.2.1 En caso de haber dos o más operaciones necesitando el mismo recurso, se utilizan las reglas de despacho para desempatar (SPT, LPT, MRWT, EDD, CR, ATC, M. SLACK).

P3.1.3 Seleccionar la transición de la matiz de asignaciones cuyo tiempo de inicio sea menor y dispararla.

P3.2 Mover la transición disparada del vector de transiciones habilitadas, a el vector de transiciones disparadas y actualizar los vectores "Mp", "Mr" y "rwt".

P3.3 Actualizar el vector de transiciones habilitadas. Si el vector de transiciones habilitadas está vacío, se ha obtenido un programa factible de los trabajos para el FMS, de lo contrario vaya a P3.1 y siga el algoritmo.

P3.4 Guardar las seis secuencias obtenidas en los vectores correspondientes, así como sus valores objetivos.

P4. [*Generar el Programa de Producción Inicial mediante el algoritmo genético*]

P4.0 Realizar la representación Genética del Sistema. Secuencia de "*ng*" genes obtenidos aleatoriamente en el intervalo [0, *ng* - 1], en un cromosoma de representación indirecta, el cual a continuación es decodificado para representar transiciones a disparar por medio de la política de programación sin retraso, con el fin de obtener programas en donde las maquinas nunca

permanecen inactivas mientras existan ordenes de trabajo en espera listos para ser programados.

El tamaño del cromosoma simbolizado en su número de genes depende del número de transiciones que deben dispararse para completar los trabajos. Como ya se han obtenido 7 secuencias iniciales, el tamaño será el mismo de cualquiera de esas 7 secuencias.

P4.1 Población Inicial. La población inicial generada contiene cuatro tipos de cromosomas: los siete primeros no utilizan los genes generados según la representación genética del paso P2.0, sino que son representados por una secuencia de transiciones disparadas según las reglas de despacho SPT (Cuando una máquina está disponible, se programan primero los trabajos con menor tiempo de proceso), LPT (Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo de proceso), MRWT (Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo remanente de proceso), EDD (Se programan primero los trabajos con fecha de entrega más cercana), CR (Diferencia entre la fecha de entrega y la fecha actual dividida entre el tiempo de procesamiento restante, se programan primero las ordenes con proporción crítica más baja), ATC (Cuando una máquina está disponible, se programan primero los trabajos con mayor costo aparente de tardanza), M. SLACK (Los trabajos se programan dependiendo del que tenga menor holgura), respectivamente; en los demás, se genera una población aleatoria de "*psize - 7*" cromosomas, según la representación genética del paso P2.0. $psize$ representa el parámetro del AG, asociado con el tamaño de la población.

P4.1.1 Decodificación: Se utiliza la función modulo(%) utilizando el tamaño del vector de transiciones habilitadas (ump) para obtener el vector de transiciones a disparar.

P4.2. Nivel de Adaptación. El nivel de adaptación del AG es el valor del objetivo que equivale al makespan ($C\max = (C_1, C_2, ..C_{No+Nn})$), la tardanza total ponderada de las ordenes de trabajo ($\sum_{j=1}^{N} w_j T_j$) o el valor ponderado de los dos objetivos ($\alpha C_{max} + \beta \sum_{j=1}^{N} w_j T_j$). En este paso se evalúa el nivel de adaptación de cada cromosoma contenido en la población.

P.4.3. Mejor individuo de la población: Se escoge el individuo de la población que posea el menor valor objetivo.

P4.4 Nueva Población. Genere una nueva población repitiendo los siguientes pasos hasta que la nueva población esté completa.

P4.4.1 Selección. Seleccione dos cromosomas padres de una población según la técnica de selección por torneo.

P4.4.2 Crossover. Con la probabilidad de recombinación o cruce entre los padres por un punto, formar una nueva descendencia (los hijos). Si ninguna recombinación fuera realizada, la descendencia es una copia exacta de los padres.

P4.4.3 Mutación. Con una probabilidad de mutación, mute la nueva descendencia en cada sitio o posición en el cromosoma que ha sido seleccionado aleatoriamente.

P4.4.3.1. Intensidad de mutación. De acuerdo a éste parámetro se realizan una o varias mutaciones al mismo individuo.

P4.4.4. Elitismo. Con el porcentaje de elitismo, se seleccionan los individuos con mejores

fitness de la población actual y se pasan a la siguiente generación sin realizar sobre ellos ninguna modificación.

P4.4.5 Aceptación. Coloque la nueva descendencia en una nueva población.

P4.5 Reemplace. Use la nueva población generada para correr nuevamente el algoritmo.

P4.6 Prueba. Si el número de generaciones (*maxGenerations*) no ha sido completada vaya a la etapa P.4.4, de lo contrario pare y retorne la mejor solución de la población actual.

P4.7. Considere la mejor solución de la población actual, como la séptima secuencia de transiciones.

P5 Seleccionar la secuencia de transiciones cuyo valor objetivo sea menor.

P6 Cambiar la secuencia de transiciones disparadas a una secuencia en Gant, según el formato del archivo user.seq del software LEKIN y siga al SubAlgoritmo de Simulación.

En la implementación del Subalgoritmo de Programación Inicial en Microsoft Visual Studio 2010, el formato de la clase Petrinet fue tomado del Algoritmo Afs Petrinet (Acevedo, J. y Mejía, G., 2005) y adaptado para representar las fallas de las máquinas como sigue a continuación:

- Sea una Red de Petri Temporizada en las plazas, $RdPT = (P, T, TxP, PxT, Mo, \tau)$ donde: $P = \{p_1, p_2, \ldots, p_n\}$ es el conjunto finito de plazas representadas por círculos, para $n > 0$; $T = \{t_1, t_2, \ldots, t_m\}$ es el conjunto finito de transiciones representados por barras, con $P \cup T \neq 0$ y $P \cap T = 0$ para $m > 0$; $TxP \rightarrow N$, es una función de entrada que especifica el conjunto de arcos dirigidos desde T hasta P, donde $N = \{0,1,2,\ldots\}$; $PxT \rightarrow N$, es una función de salida que define el conjunto de arcos dirigidos desde P hasta T, donde $N = \{0,1,2,\ldots\}$; $M0$ marcaje inicial de la RdPT; y τ el conjunto de tiempos asociados con las plazas.
- La red tiene *n* plazas y *m* transiciones. Las plazas son operacionales (O) o de recurso (R): $P = \{p_1, p_2, \ldots, p_n\} = O \cup R$. Las plazas operacionales representan la acción "Operación $O_{i,j,k}$ en proceso y las condiciones "Orden de trabajo disponible", "Orden de trabajo terminada" y "Orden de trabajo esperando inicializar su procesamiento en la estación S". Las plazas de recurso representan la disponibilidad de una máquina. El marcaje inicial de las plazas de recurso es 1 para todos los recursos. El tiempo asociado con las plazas de recurso o plazas de condición son 0.

Ejemplo: Un sistema de manufactura desarrolla dos (2) trabajos con dos (2) operaciones con una secuencia establecida y cuatro (4) máquinas en el sistema. La ruta de cada trabajo se muestra en la tabla 2, en donde los tiempos de proceso de cada operación se muestran en paréntesis.

Tabla 2

Ruta de las operaciones de dos trabajos en un sistema de manufactura con cuatro máquinas

OPERACIÓN	TRABAJO 1	TRABAJO 2
(Operación 1)	M1(6)	Mach2 (7)
(Operación 2)	M3 (8) o M4 (8)	Mach4 (5)

La modelación del sistema anteriormente descrito se presenta a continuación:

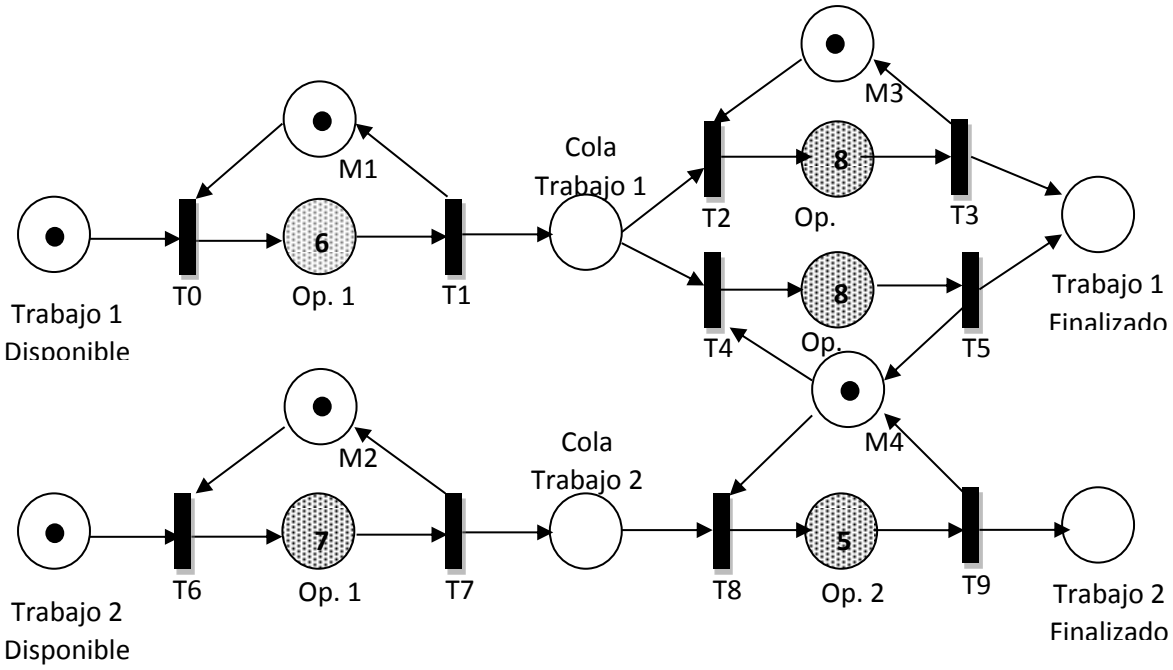


Figura 9. Modelo de RdP para un Job Shop con dos trabajos, cuatro máquinas y una secuencia de operaciones establecida.

Las funciones de entrada y salida de la Red de Petri utilizada para representar el sistema de manufactura en la figura 9, pueden ser representadas usando las siguientes matrices de incidencia:

$C^+ =$

	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}
t_0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
t_1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
t_2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
t_3	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
t_4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
t_5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
t_6	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
t_7	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
t_8	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
t_9	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

Figura 10. Matriz de incidencia positiva para el job shop representado en la figura 9.

$C^- =$

	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}
t_0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
t_1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
t_2	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
t_3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
t_4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
t_5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
t_6	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
t_7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
t_8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
t_9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Figura 11. Matriz de incidencia negativa para el job shop representado en la figura 9.

$$C = \begin{array}{c|cccccccccccccc} & p_0 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 & p_9 & p_{10} & p_{11} & p_{12} & p_{13} & p_{14} \\ \hline t_0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ t_1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ t_2 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ t_3 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ t_4 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ t_5 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ t_6 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ t_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ t_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 \\ t_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 1 \end{array}$$

Figura 12. Matriz de incidencia para el job shop representado en la figura 9.

En donde,

p_0	Trabajo 1 disponible	p_8	Cola trabajo 2
p_1	Op.1 del trabajo 1	p_9	Op.2 del trabajo 2
p_2	Cola trabajo 1	p_{10}	Trabajo 2 finalizado
p_3	Op. 2.1 del trabajo 1	p_{11}	Máquina 1 (M1)
p_4	Op. 2.2 del trabajo 1	p_{12}	Máquina 2 (M2)
p_5	Trabajo 1 finalizado	p_{13}	Máquina 3 (M3)
p_6	Trabajo 2 disponible	p_{14}	Máquina 4 (M4)
p_7	Op.1 del trabajo 2		

En el presente estudio se utilizaron las Redes de Petri temporizadas para modelar el sistema flexible de manufactura junto con las fallas aleatorias de las máquinas. En la figura 13, se puede evidenciar el sistema representado anteriormente (Figura 9) con plazas y transiciones adicionales que corresponden a las máquinas en reparación después de una falla.

Las transiciones de fallas de máquinas se disparan cuando en la simulación haya fallado la máquina correspondiente mientras se procesa la operación. Para esto, se debe tener en cuenta el vector de marcaje M_k en el momento en que falla la máquina. Al dispararse la transición correspondiente, es removido el token que se encuentra en la plaza de operación y dos tokens son agregados a las plazas correspondientes a la reparación de la máquina y a la plaza de trabajo disponible (si la operación era la inicial) o a la cola del trabajo. Al terminar el tiempo de reparación de la máquina, es disparada la transición correspondiente y el token es devuelto a la máquina para que ésta vuelva a estar disponible.

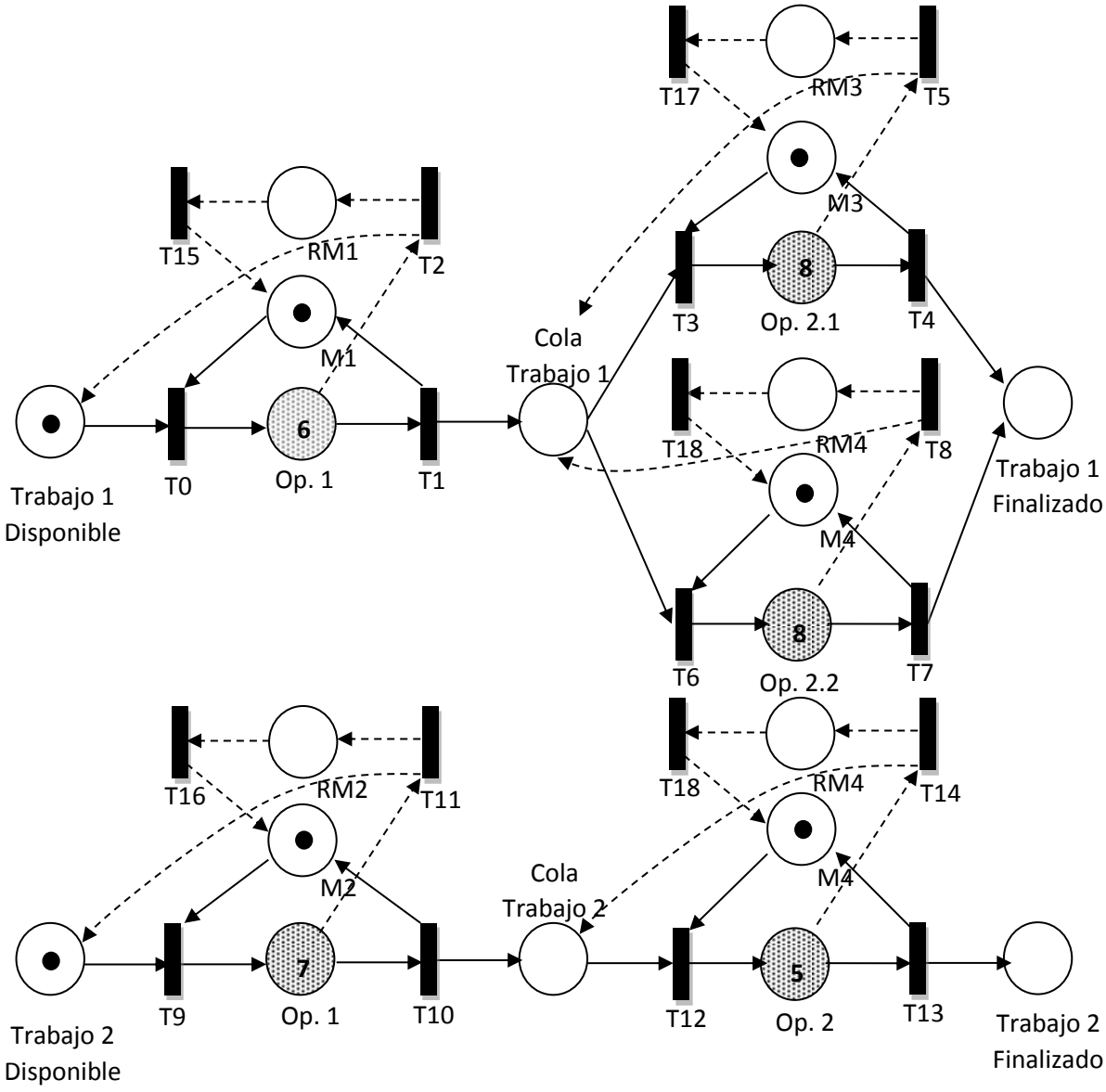


Figura 13. Modelo de RdP para un Job Shop con dos trabajos, cuatro máquinas y una secuencia de operaciones establecida, teniendo en cuenta fallas de máquinas.

En donde,

Op. 1	Operación 1
Op. 2	Operación 2
Op. 2.1	Operación 2 opción 1

Op. 2.2	Operación 2 opción 2
M1	Máquina 1
M2	Máquina 2
M3	Máquina 3
M4	Máquina 4
RM1	Reparación de la máquina 1
RM2	Reparación de la máquina 2
RM3	Reparación de la máquina 3
RM4	Reparación de la máquina 4
T0, T3, T6, T12	Transiciones que dan inicio a las operaciones
T1, T4, T7, T10, T13	Transiciones que finalizan las operaciones
T2	Falla M1 mientras trabajaba en Op. 1 del trabajo 1
T5	Falla M3 mientras trabajaba en Op. 2.1 del trabajo 1
T8	Falla M4 mientras trabajaba en Op. 2.2 del trabajo 1
T11	Falla M2 mientras trabajaba en Op. 1 del trabajo 2
T14	Falla M4 mientras trabajaba en Op. 2 del trabajo 2
T15	Habilita la M1 después de ser reparada
T16	Habilita la M2 después de ser reparada
T17	Habilita la M3 después de ser reparada
T18	Habilita la M4 después de ser reparada

Al generarse la falla de una máquina, se debe realizar la reprogramación de las actividades por medio de la utilización de muchas estrategias, por medio del Subalgoritmo de programación reactiva.

2.2 Subalgoritmo de simulación en PetNMA

Para realizar una simulación de los eventos debemos tener en cuenta el vector de transiciones disparadas, ya que éste vector, obtenido en el Subalgoritmo de Programación, se encuentra relacionado con el vector de tiempos que indica el momento en que debe ser realizada cada

operación o cada evento.

Esta lista de eventos a realizar va variando con el tiempo a medida que cada evento se va realizando. Sin embargo, las operaciones se van realizando en unas máquinas diseñadas con el fin de lograr un objetivo que para que sea cumplido son constituidas por diferentes componentes y a la vez deben estar en condiciones ideales de funcionamiento, pero gracias a muchos factores estas máquinas después de cierto tiempo pueden presentar fallas. Es por esto que se debe realizar una reprogramación o programación reactiva de la producción. Previamente, es necesario realizar una simulación de la ejecución de las operaciones que han sido programadas y de las fallas de máquinas.

Antes de realizar la simulación de eventos se debe generar la falla aleatoria de la máquina. Se debe tener en cuenta que no se cuenta con datos históricos, es por ello que para generar la falla se utiliza la metodología empleada por Mehta y Uzsoy (1999). Para determinar el tiempo en que se produce la falla se utiliza la distribución de probabilidad normal, al igual que para determinar la máquina que falla, ya que todas las máquinas tienen la misma probabilidad de falla. Sin embargo, en este estudio para determinar el tiempo de reparación de la máquina empleamos la distribución Log-Normal para establecer el tiempo de falla (Tiempo de reparación), siguiendo a Suwa, H. y Sandoh, H. en el año 2007 en su estudio para la programación reactiva en job shops con fallas de máquinas "Capability of cumulative delay based reactive scheduling for job shops with machine breakdowns". En este estudio, el tiempo de cada falla o tiempo de reparación, está dado por:

$$LN\left(\ln(\gamma_1 p), \gamma_2\right)$$

En donde, $\ln(\gamma_1 p)$ es la media de la función, p es el tiempo de procesamiento promedio del sistema, γ_1 es el porcentaje de p que hace parte del tiempo de reparación y γ_2 es la varianza.
Para esto, se adicionó la libreria que nos ayuda a realizar la función Log Normal "cdf_inv.h".

El SubAlgoritmo de Simulación de los eventos programados y las fallas de máquinas se compone por los siguientes pasos:

P1. [*Lista de Eventos Programados*]

P1.0 Crear la lista de eventos programados teniendo en cuenta el vector de transiciones disparadas.

P1.1 Formar una lista de eventos aleatorios que constituyen las fallas de máquinas. Esta lista debe contener la máquina que falla, la estación de trabajo correspondiente, el tiempo en que se presenta la falla, el tiempo de reparación de la máquina, la operación que estaba siendo procesada por dicha máquina y las correspondientes transiciones a disparar.

P2. [*Simulación de la ejecución de Eventos Programados*]

P2.0 Pasar a la lista de eventos ejecutados la primera entrada de la lista de eventos programados.

P2.1 Actualizar el tiempo del reloj al tiempo relacionado con el evento ejecutado y a su vez actualizar el estado del sistema debido a la ocurrencia del evento.
P2.2 Evaluar si la lista de eventos programados se encuentra vacía. Si la respuesta es afirmativa "PARE", de lo contrario vaya a P2.3.
P2.3 Evaluar si el tiempo del reloj es mayor o igual que el tiempo asociado al evento aleatorio de la falla de máquina. Si la respuesta es afirmativa "Ejecutar el SubAlgoritmo de Programación Reactiva", de lo contrario vaya a P2.0.

Conjunto de operaciones no procesadas (CONP)

Para realizar la posterior Programación Reactiva de la Producción en PetNMA se deben tener en cuenta todas aquellas operaciones que al momento de presentarse la(s) falla(s) no habían sido ejecutadas, al igual que la(s) operación(es) que se estaba(n) procesando en dicha(s) máquina(s) en ese preciso instante. Las operaciones que se estaban procesando durante ese momento en máquinas que no presentan fallas no se interrumpen hasta cuando no hayan sido finalizadas. A continuación se presenta la serie de pasos para obtener la lista de esas operaciones:

P1. [*Conformación de* $CONP$]:
P1.0 Inicie $CONP$ como un conjunto vacío.
P1.1 Agregue a $CONP$ la operación que al momento dla falla de máquina estaba siendo procesada sobre la máquina que presenta la falla, $CONP = \{O^{R}{}_{i,j,k}\}$.
P1.2 Agregue a $CONP$ las operaciones de los trabajos del programa inicial que al momento dla falla de máquina, no han inicializado su tiempo de procesamiento $CONP = CONP + \{O_{r,j,k}\} \quad \forall r, j, k$ donde se satisfaga que $X_{r,j,k} \geq t_a$ y $j \leq N$.

2.3. Subalgoritmo de programación reactiva en PetNMA

Al presentarse una falla de una máquina durante el desarrollo del Programa Inicial de Producción, se debe realizar la Programación Reactiva por medio del Subalgoritmo de Programación Reactiva. Para esta programación, se deben tener en cuenta todas las operaciones en el Conjunto de Operaciones No Procesadas (CONP).

El Algoritmo Memético utilizado dentro del Subalgoritmo de Programación Reactiva presentado a continuación, consta de un algoritmo evolutivo y una técnica de búsqueda local. El algoritmo genético fue seleccionado para la parte evolutiva y consta de los operadores genéticos de crossover, mutación y elitismo. Por su parte, como técnica de búsqueda local se utiliza un algoritmo de recocido simulado, el cual se ejecuta después de cada generación.

El Subalgoritmo de Programación Reactiva está compuesto por los siguientes pasos:

P1. [*Actualización de la* RdPT]

P1.0. Establecer el tiempo del reloj en el tiempo en que se presenta la falla de máquina.

P1.1. Actualizar el vector de tiempos de proceso remanente para la RdPT, según el tiempo del reloj en la simulación.

P2. [*Generar Programa de Producción Reactiva* - AM]

P2.0. Realizar la representación Genética del Sistema. Secuencia de "*ng*" genes obtenidos aleatoriamente en el intervalo [0, *ng* - 1], en un cromosoma de representación indirecta, el cual a continuación es decodificado para representar transiciones a disparar por medio de la política de programación sin retraso, con el fin de obtener programas en donde las maquinas nunca permanecen inactivas mientras existan ordenes de trabajo en espera listos para ser programados.

El tamaño del cromosoma simbolizado en su número de genes depende del estado de la RdPT en el instante en que se presenta la falla de máquina y estará dado por: $ng_r = NTPI - NES(t_{ar}) + 2$ donde:

r *r*-ésima actividad de programación reactiva, debido a la falla número $r = \{1,2,...,nbd\}$. nbd representa el número de fallas de máquinas simuladas.

t_{bd} Instante t en que se presenta la falla r.

$NTPI$ Número de transiciones programadas para ser disparadas en el programa inicial de producción.

$NES(t_{bd})$ Número de transiciones disparadas durante la simulación hasta el momento de la falla r.

ng_r Número de genes que constituyen el cromosoma para la programación reactiva, debido a la falla r.

ng_{r-1} Número de genes que constituyeron el cromosoma en la programación reactiva, debido a la falla $r-1$. Cuando $r = 1$, $ng_0 = 2\sum_{j=1}^{No} n_j$

P2.1. Población Inicial. La población inicial generada contiene cuatro tipos de cromosomas: los siete primeros no utilizan los genes generados según la representación genética del paso P2.0, sino que son representados por una secuencia de transiciones disparadas según las reglas de despacho SPT (Cuando una máquina está disponible, se programan primero los trabajos con menor tiempo de proceso), LPT (Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo de proceso), MRWT (Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo remanente de proceso), EDD (Se programan primero los trabajos con fecha de entrega más cercana), CR (Diferencia entre la fecha de entrega y la fecha actual dividida entre el tiempo de procesamiento restante, se programan primero las ordenes con proporción crítica más baja), ATC (Cuando una máquina está

disponible, se programan primero los trabajos con mayor costo aparente de tardanza), M. SLACK (Los trabajos se programan dependiendo del que tenga menor holgura), respectivamente; en los demás, se genera una población aleatoria de "*psize - 7*" cromosomas, según la representación genética del paso P2.0. $_{psize}$ representa el parámetro del AG, asociado con el tamaño de la población.

P2.1.1 Decodificación: Se utiliza la función módulo(%) utilizando el tamaño del vector de transiciones habilitadas (um) para obtener el vector de transiciones a disparar. Se debe tener en cuenta que si la máquina que presenta la falla todavía está en reparación al momento en que se termina la última operación, dicha máquina queda en reparación y el cromosoma tendrá un gen menos. Sin embargo, si al aplicar la función modulo, es escogida una transición de terminación de una operación, se debe escoger la transición de terminación más próxima a ser disparada, esto con el fin de que ninguna operación sea procesada por más tiempo del necesario. Cada vez que se decodifica un individuo, se debe evaluar el tiempo de disparo de cada transición que se va disparando, con el fin de disparar la transición que habilita la máquina que falló en el momento exacto en el que termina su reparación. Como los tres primeros individuos no son decodificados de esta manera, se deben codificar los individuos y reemplazar en la población codificada para guardar uniformidad. Guardar en matrices los individuos codificados y decodificados.

P2.2. Nivel de Adaptación. El nivel de adaptación del AG es el valor del objetivo que equivale al makespan ($C\max = (C_1, C_2, ..C_{No+Nn})$), la tardanza total ponderada de las ordenes de trabajo ($\sum_{j=1}^{N} w_j T_j$) o el valor ponderado de los dos objetivos ($\alpha C_{max} + \beta \sum_{j=1}^{N} w_j T_j$). En este paso se evalúa el nivel de adaptación de cada cromosoma contenido en la población.

P2.3. Mejor individuo de la población: Se escoge el individuo de la población que posea el menor valor objetivo ponderado y en dos vectores se guardan tanto el individuo codificado como el decodificado.

P2.4. RECOCIDO SIMULADO (SA) Se aplica la técnica de búsqueda local.

P2.4.1. Se escoge la mejor solución obtenida de la generación recientemente generada, ésta es el óptimo local.

P2.4.2 Se inicializa la temperatura inicial y la temperatura final.

P2.4.3 Se genera un vecino intercambiando dos genes del óptimo local.

P.2.4.3.1. Se evalúa el fitness del vecino y se genera un delta que equivale al fitness del vecino menos el fitness del óptimo local.

P.2.4.3.2 Se evalúa el criterio de finalización, si el fitness del vecino es menor al fitness del óptimo local el nuevo óptimo local será el vecino, sino se evalúa el criterio de aceptación probabilístico.

P.2.4.2.3 Se actualiza la temperatura.

P.2.4.3 Si la temperatura es mayor a la temperatura final regresar a P2.3.3, de lo contrario continuar con P2.5.

P2.5. Nueva Población. Genere una nueva población repitiendo los siguientes pasos hasta que la nueva población esté completa.

P2.5.1. Selección. Seleccione dos cromosomas padres de una población según la técnica de selección por torneo.

P2.5.2. Crossover. Con la probabilidad de recombinación o cruce entre los padres por un punto, formar una nueva descendencia (los hijos). Si ninguna recombinación fuera realizada, la descendencia es una copia exacta de los padres.

P2.5.3. Mutación. Con una probabilidad de mutación, mute la nueva descendencia en cada sitio o posición en el cromosoma que ha sido seleccionado aleatoriamente.

P2.5.3.1. Intensidad de mutación. De acuerdo a éste parámetro se realizan una o varias mutaciones al mismo individuo.

P2.5.4. Elitismo. Con el porcentaje de elitismo, se seleccionan los individuos con mejores fitness de la población actual y se pasan a la siguiente generación sin realizar sobre ellos ninguna modificación.

P2.5.4. Aceptación. Coloque la nueva descendencia en una nueva población.

P2.6. Reemplace. Use la nueva población generada para correr nuevamente el algoritmo.

P2.7. Prueba. Si el número de generaciones (*maxGenerations*) no ha sido completada vaya a la etapa P.2.3, de lo contrario pare y retorne la mejor solución población actual.

P2.8. Considere la mejor solución, como el nuevo vector de transiciones a disparar en el SubAlgoritmo de Programación Reactiva.

3. DISEÑO DE EXPERIMENTOS DEL ALGORITMO PETNMA

Las pruebas experimentales fueron desarrolladas mediante corridas realizadas en un computador Dell Inspiron con un procesador Intel Inside CORE I5, 8 gigas de RAM, 1 TB de memoria y Windows 7.

3.1. Pruebas Experimentales del subalgoritmo de programación inicial en PetNMA

Con la finalidad de obtener los programas iniciales con el Subalgoritmo de Programación en PetNMA, se utilizaron diez (10) problemas que caracterizan a FMS (ver anexo A). Los problemas que se utilizaron para hacer las pruebas fueron adaptados de los problemas desarrollados por Caballero, J. P. & Mejía, G. (2004) y los problemas clásicos LA y ORB (Singer, 1999).

En general, los problemas tienen las siguientes características:

- Cada problema tiene un número de trabajos a programar en un número de estaciones de trabajo determinado y en número de máquinas (ver tabla a continuación).

Tabla 3

Descripción de los 10 problemas escogidos para realizar las pruebas experimentales.

No. del problema	No. de trabajos	No. de estaciones de trabajo	No. de máquinas
1	6	10	19
2	9	8	20
3	10	5	10
4	10	5	12
5	10	5	12
6	10	7	18
7	10	10	20
8	15	5	12
9	15	6	17
10	20	6	13

- Los trabajos se pueden identificar por medio del siguiente sistema de codificación:

FMSXX_J_S_M con: XX. Identificación del Problema (01, 02, …)
J. Número de Trabajos a Programar
S. Número de Estaciones en el FMS
M. Número de Máquinas en el FMS

- La ruta de los trabajos por las estaciones del FMS está constituida por los tiempos de cada una de sus operaciones. Dichos tiempos fueron establecidos aleatoriamente dentro del intervalo [10, 50], con el fin de obtener variabilidad en los problemas.
- En los problemas clásicos, se realiza la asignación de las máquinas a cada una de las estaciones de trabajo de forma aleatoria dentro del intervalo [1, 3]. Durante la ejecución de pruebas experimentales, se ha encontrado que para el tipo de problemas considerados, una asignación de máquinas a las estaciones del sistema superior a tres (3), produce tiempos muertos en máquinas durante todo el proceso de programación de las ordenes de trabajo.
- Los pesos de los trabajos fueron asignados de forma aleatoria teniendo en cuenta la distribución uniforme dentro del intervalo [1, 3].
- Para determinar los tiempos de entrega de cada trabajo, se utilizó la regla de trabajo total (TWK), en donde dj = kPj, siendo Pj la suma de los tiempos de proceso de todas las operaciones del trabajo j y k un parámetro. De acuerdo a Buitrago, O. et al, 2009, se asignaron tiempos de entrega con k = 1.3 y k = 1.5 seleccionados aleatoriamente. Sin embargo, se utilizó una distribución normal con un rango [1.3,1.5] con el fin de dar mayor diversificación a los tiempos de entrega.
- Para realizar el diseño de experimentos con el fin de validar los parámetros, se clasifican los problemas de acuerdo a la cantidad de trabajos a desarrollar en el sistema en pequeños, medianos y grandes. Teniendo en cuenta la clasificación los problemas del 1 al 3 corresponden a los problemas pequeños, del 4 al 7 a los problemas medianos y del 8 al 10 a los problemas grandes.

Con miras a evaluar el desempeño del Subalgoritmo de Programación en PetNMA, los resultados que han sido obtenidos en cada una de las corridas de los problemas, se han comparado con los resultados obtenidos al aplicar al mismo problema las reglas de despacho SPT (Cuando una máquina está disponible, se programan primero los trabajos con menor tiempo de proceso), LPT (Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo de proceso), MRWT (Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo remanente de proceso), EDD (Se programan primero los trabajos con fecha de entrega más cercana), CR (Diferencia entre la fecha de entrega y la fecha actual dividida entre el tiempo de procesamiento restante, se programan primero las ordenes con proporción crítica más baja), ATC (Cuando una máquina está disponible, se programan primero los trabajos con mayor costo aparente de tardanza), M. SLACK (Los trabajos se programan dependiendo del que tenga menor holgura), y con los resultados obtenidos con el cuello de botella. La comparación se realiza entre las funciones objetivo evaluadas, en este caso el makespan o tiempo de terminación de la última operación programada, la tardanza total ponderada y el objetivo ponderado.

Para cada uno de los objetivos se ha realizado un diseño de experimentos factorial con el fin de escoger los mejores parámetros para el algoritmo genético. Los parámetros escogidos para las

corridas fueron tomados de los estudios “Combinación de métodos heurísticos con redes de Petri para la programación de sistemas de manufactura flexible” realizado por Castro, A. y Mejía, G. (2006), “Optimization of scheduling problems: A genetic algorithm survey” desarrollado por Kumar, A. y Dhingra, A. (2012), “Solving distributed flexible manufacturing systems scheduling problems subject to maintenance Memetic algorithms approach” realizado por Yadollahi, M. y Rahmani, A.M. (2009), “Programación Reactiva y Robusta de la Producción en un Ambiente Sistema de Manufactura Flexible: Llegada de Nuevas Órdenes y Cambios en la Prioridad de las Órdenes de Trabajo” desarrollado por Acevedo, J. y Mejía, G. (2005).

Tabla 4

Parámetros escogidos para realizar las pruebas del Algoritmo Genético del Subalgoritmo de programación inicial de PetNMA

Parámetros de PetNMA para la programación inicial	Nomenclatura	Valores
Numero de Generaciones	***Maxgen***	200
Tamaño de la Población Inicial	***P***	100
Probabilidad de Cruce	***Tc***	60%, 90%
Probabilidad de Mutación	***Tm***	10%, 50%
Intensidad de Mutaciones	***Im***	1, 2, 4
Elitismo	$\boldsymbol{Te_1}$	Off, On (25%,50%)

En el caso del objetivo ponderado, se realizaron las pruebas con todas las opciones de ponderación para alfa y beta con un decimal, con el fin de escoger la combinación que encuentra los mejores resultados, las combinaciones se muestran a continuación:

Tabla 5

Combinaciones para las ponderaciones del objetivo ponderado.

Alfa (α)	Beta (β)
0.0	1.0
1.0	0.0
0.5	0.5
0.1	0.9
0.2	0.8
0.3	0.7
0.4	0.6
0.6	0.4
0.7	0.3
0.8	0.2
0.9	0.1

De los 10 problemas se escogieron 3 (Un problema pequeño (2), un problema mediano (7) y un problema grande (9)) para realizar las corridas de parametrización en donde se escogen los mejores niveles de los parámetros evaluados. Los resultados de las corridas, el diseño de experimento y sus gráficas se encuentran en el anexo B.

Teniendo en cuenta los resultados obtenidos, se escogieron los siguientes valores para los parámetros del algoritmo genético del Subalgoritmo de Programación Inicial:

Tabla 6

Parámetros seleccionados para el Algoritmo Genético del Subalgoritmo de programación inicial de PetNMA para evaluar el Makespan

Parámetros de PetNMA para la programación inicial (Makespan)				
Parámetros	**Nomenclatura**	**Problemas Pequeños**	**Problemas Medianos**	**Problemas Grandes**
Numero de Generaciones	Maxgen	100	100	100
Tamaño de la Población Inicial	P	100	100	100
Probabilidad de Cruce	Tc	60%	60%	60%
Probabilidad de Mutación	Tm	50%	50%	50%
Intensidad de Mutaciones	Im	2	2	2
Elitismo	Te1	50%	50%	Off

Tabla 7

Parámetros seleccionados para el Algoritmo Genético del Subalgoritmo de programación inicial de PetNMA para evaluar la Tardanza Total Ponderada

Parámetros de PetNMA para la programación inicial (TTP)				
Parámetros	**Nomenclatura**	**Problemas Pequeños**	**Problemas Medianos**	**Problemas Grandes**
Numero de Generaciones	Maxgen	200	200	200
Tamaño de la Población Inicial	P	100	100	100
Probabilidad de Cruce	Tc	90%	90%	90%
Probabilidad de Mutación	Tm	50%	50%	50%
Intensidad de Mutaciones	Im	4	4	4
Elitismo	Te1	50%	50%	50%

Con respecto al objetivo ponderado se realizaron las pruebas para todas las combinaciones, los parámetros seleccionados para cada combinación se muestran a continuación:

Tabla 8

Parámetros seleccionados para cada combinación de ponderación del Problema Pequeño del Algoritmo Genético del Subalgoritmo de Programación Inicial de PetNMA

Parámetros seleccionados para cada combinación de ponderación del Problema Pequeño					
Alfa	**Beta**	**Probabilidad de cruce (%)**	**Probabilidad de mutación (%)**	**Intensidad de mutación**	**Porcentaje de elitismo (%)**
50	**50**	60	50	4	50
10	**90**	60	50	4	25
20	**80**	90	50	4	25
30	**70**	60	50	4	25
40	**60**	90	50	2	25
60	**40**	90	50	4	50
70	**30**	60	10	4	50
80	**20**	60	50	4	25
90	**10**	90	10	4	25

Tabla 9

Parámetros seleccionados para cada combinación de ponderación del Problema Mediano del Algoritmo Genético del Subalgoritmo de Programación Inicial de PetNMA

Parámetros seleccionados para cada combinación de ponderación del Problema Mediano					
Alfa	**Beta**	**Probabilidad de cruce (%)**	**Probabilidad de mutación (%)**	**Intensidad de mutación**	**Porcentaje de elitismo (%)**
50	**50**	60	50	4	50
10	**90**	60	50	4	25
20	**80**	90	50	4	25
30	**70**	60	50	4	25
40	**60**	90	50	2	25
60	**40**	90	50	4	50
70	**30**	60	10	4	50
80	**20**	60	50	4	25
90	**10**	90	10	4	25

Tabla 10

Parámetros seleccionados para cada combinación de ponderación del Problema Grande del Algoritmo Genético del Subalgoritmo de Programación Inicial de PetNMA

Parámetros seleccionados para cada combinación de ponderación del Problema Grande					
Alfa	**Beta**	**Probabilidad de cruce (%)**	**Probabilidad de mutación (%)**	**Intensidad de mutación**	**Porcentaje de elitismo (%)**
50	**50**	90	50	4	50
10	**90**	60	50	4	25
20	**80**	60	50	4	50
30	**70**	60	50	4	50
40	**60**	90	50	4	50
60	**40**	60	50	4	50
70	**30**	60	50	2	25
80	**20**	60	50	2	25
90	**10**	60	50	1	25

Con los resultados obtenidos se realizaron las gráficas de Pareto con el fin de seleccionar la mejor combinación. Dichas pruebas fueron realizadas sobre las 10 instancias (Ver anexo D). A continuación se presentan las gráficas de Pareto de los problemas tipo:

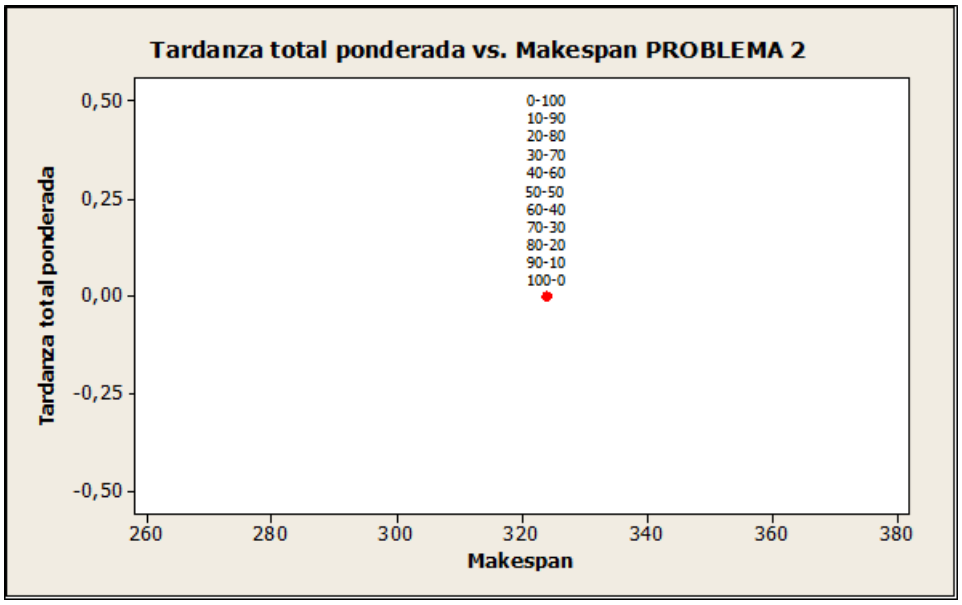


Figura 14. Pareto realizado del Objetivo Ponderado (Makespan – TTP) del problema tipo pequeño.

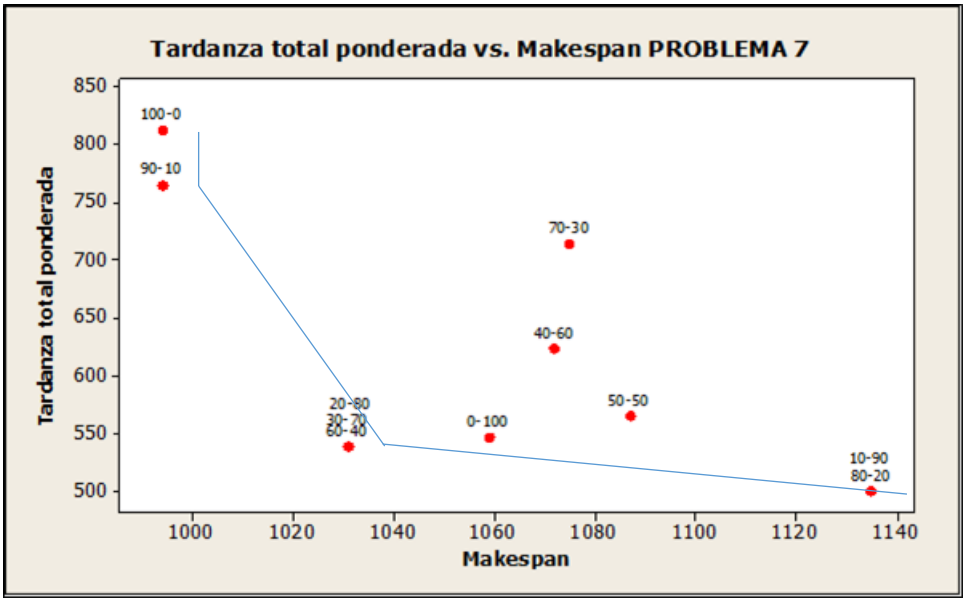


Figura 15. Pareto realizado del Objetivo Ponderado (Makespan – TTP) del problema tipo mediano.

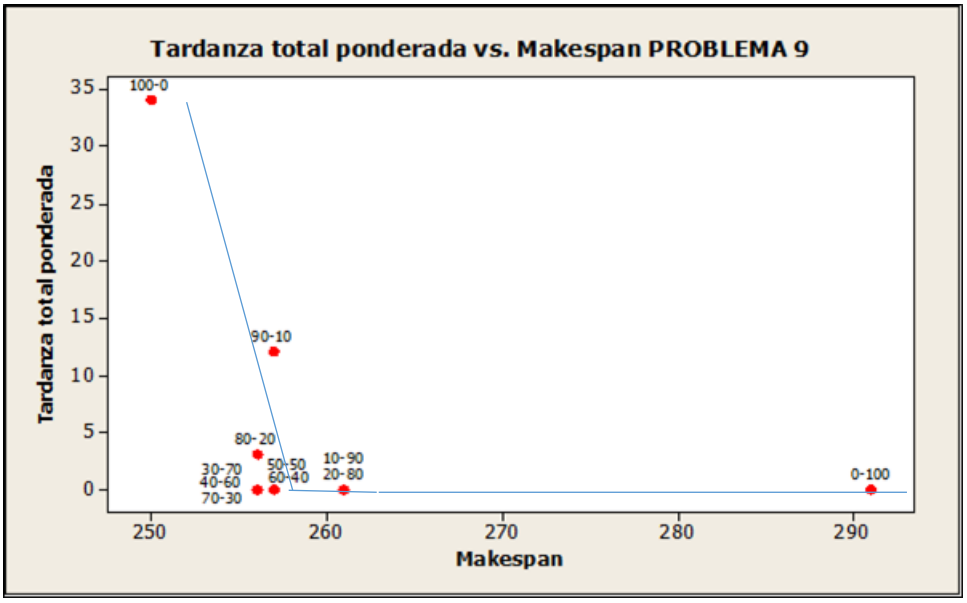


Figura 16. Pareto realizado del Objetivo Ponderado (Makespan – TTP) del problema tipo grande.

De acuerdo a los resultados se seleccionó la combinación 50 - 50 y los parámetros seleccionados para el algoritmo genético con el fin de evaluar el objetivo ponderado son:

Tabla 11

Parámetros seleccionados para el Algoritmo Genético del Subalgoritmo de programación inicial de PetNMA para evaluar el Objetivo Ponderado

Parámetros de PetNMA para la programación inicial (Objetivo Ponderado)				
Parámetros	**Nomenclatura**	**Problemas Pequeños**	**Problemas Medianos**	**Problemas Grandes**
Numero de Generaciones	Maxgen	200	200	200
Tamaño de la Población Inicial	P	100	100	100
Probabilidad de Cruce	Tc	60%	60%	90%
Probabilidad de Mutación	Tm	50%	50%	50%
Intensidad de Mutaciones	Im	4	4	4
Porcentaje de elitismo	Te1	50%	50%	50%

Tabla 12

Resultados del Subalgoritmo de Programación inicial para el objetivo de minimización del Makespan

RESULTADOS DEL SUBALGORITMO DE PROGRAMACIÓN INICIAL EN PetNMA PARA EL MAKESPAN (Continua… **)**

Problema	CUELLO DE BOTELLA	REGLAS DE DESPACHO							
		SPT	LPT	MRWT	EDD	CR	ATC	M. SLACK	Mínimo
FMS01_6_10_19	412	436	464	436	464	436	436	464	436
FMS02_9_8_20	316	377	346	346	377	324	324	361	324
FMS03_10_5_10	536	726	757	536	703	632	632	593	536
FMS04_10_5_12	395	470	483	401	487	483	489	418	401
FMS05_10_5_12	412	476	472	390	516	447	447	506	390
FMS06_10_7_18	342	332	382	357	375	352	352	349	332
FMS07_10_10_20	1143	1152	1146	1149	1110	1132	1155	1209	1110
FMS08_15_5_12	470	561	559	459	574	569	552	555	459
FMS09_15_6_17	269	288	288	263	298	261	275	299	261
FMS10_20_6_13	753	836	935	759	814	912	873	779	759

(…continuación **) RESULTADOS DEL SUBALGORITMO DE PROGRAMACIÓN INICIAL EN PetNMA PARA EL MAKESPAN**

Problema	PROGRAMAS ACTIVOS CON REGLAS DE DESPACHO (PetNMA)								ALGORITMO GENÉTICO		
	SPT	LPT	MRWT	EDD	CR	ATC	M. SLACK	Mínimo	Mínimo	Máximo	Promedio
FMS01_6_10_19	480	456	412	480	456	456	456	412	436	436	436
FMS02_9_8_20	369	357	326	377	341	367	361	326	324	324	324
FMS03_10_5_10	736	754	536	856	632	662	771	536	536	536	536
FMS04_10_5_12	483	478	395	524	465	569	478	395	394	394	394
FMS05_10_5_12	541	475	379	555	515	469	512	379	390	390	390
FMS06_10_7_18	408	468	368	394	380	398	379	368	332	332	332
FMS07_10_10_20	1291	1198	1164	1149	1158	1160	1323	1149	994	1036	1019,3
FMS08_15_5_12	682	581	487	671	582	633	693	487	442	459	453,8
FMS09_15_6_17	394	344	287	396	318	328	291	287	249	256	252,7
MS10_20_6_13	924	977	800	995	845	999	809	800	753	753	753

Donde:

SPT:	Cuando una máquina está disponible, se programan primero los trabajos con menor tiempo de proceso obtenido con la regla de despacho de mejor desempeño.
LPT:	Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo de proceso.
MRWT:	Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo remanente de proceso.
EDD:	Cuando una máquina está disponible, se programan primero los trabajos con menor tiempo de entrega.
CR:	Cuando una máquina está disponible, se programan primero los trabajos con menor diferencia entre la fecha de vencimiento y la fecha actual, dividida entre las unidades de tiempo restantes.
ATC:	Cuando una máquina está disponible, se programan primero los trabajos con mayor costo aparente de tardanza.
M.SLACK:	Los trabajos se programan dependiendo del que tenga menor holgura.
PROMEDIO:	Valor promedio de la Suma de los Tiempos de Terminación de las órdenes de trabajo al desarrollar diez (10) réplicas de Programación Inicial.

Tabla 13

Resultados del Subalgoritmo de Programación inicial para el objetivo de minimización de la Tardanza Total Ponderada

RESULTADOS DEL SUBALGORITMO DE PROGRAMACIÓN INICIAL EN PetNMA PARA LA TTP (Continua… **)**									
Problema	**CUELLO DE BOTELLA**	**REGLAS DE DESPACHO**							
		SPT	**LPT**	**MRWT**	**EDD**	**CR**	**ATC**	**M. SLACK**	**Mínimo**
FMS01_6_10_19	0	0	0	0	0	0	0	0	0
FMS02_9_8_20	0	0	0	0	0	0	0	0	0
FMS03_10_5_10	1716	1919	2525	2387	2332	2177	2130	2183	1919
FMS04_10_5_12	50	208	390	205	326	152	404	183	152
FMS05_10_5_12	156	226	669	472	347	410	519	240	226
FMS06_10_7_18	0	0	130	0	0	0	0	0	0
FMS07_10_10_20	359	1852	1946	2672	1325	1917	2415	2369	1325
FMS08_15_5_12	304	548	1882	1043	648	477	995	439	439
FMS09_15_6_17	0	4	310	421	1	69	205	1	1
FMS10_20_6_13	5891	8449	11815	12099	7347	10642	11293	8647	7347

(…continuación **) RESULTADOS DEL SUBALGORITMO DE PROGRAMACIÓN INICIAL EN PetNMA PARA LA TTP**											
Problema	**PROGRAMAS ACTIVOS CON REGLAS DE DESPACHO (PetNMA)**								**ALGORITMO GENÉTICO**		
	SPT	**LPT**	**MRWT**	**EDD**	**CR**	**ATC**	**M. SLACK**	**Mínimo**	**Mínimo**	**Máximo**	**Promedio**
FMS01_6_10_19	0	10	0	0	0	0	0	0	0	0	0
FMS02_9_8_20	0	0	0	0	3	29	0	0	0	0	0
FMS03_10_5_10	2008	2656	2418	3676	2158	2462	2733	2008	1577	1604	1581,6
FMS04_10_5_12	202	454	349	295	159	706	131	131	40	40	40
FMS05_10_5_12	446	1172	1014	829	679	702	614	446	144	169	155,5
FMS06_10_7_18	210	436	0	27	37	85	0	0	0	0	0
FMS07_10_10_20	3761	3079	3375	1793	2300	2664	4041	1793	476	630	563,5
FMS08_15_5_12	1896	2518	1933	1634	755	2065	936	755	150	294	230,2
FMS09_15_6_17	902	1126	654	396	186	731	0	0	0	0	0
FMS10_20_6_13	11041	14153	13321	9160	11586	15680	9309	9160	6159	6944	6564,5

Donde:

SPT:	Cuando una máquina está disponible, se programan primero los trabajos con menor tiempo de proceso obtenido con la regla de despacho de mejor desempeño.
LPT:	Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo de proceso.
MRWT:	Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo remanente de proceso.
EDD:	Cuando una máquina está disponible, se programan primero los trabajos con menor tiempo de entrega.
CR:	Cuando una máquina está disponible, se programan primero los trabajos con menor diferencia entre la fecha de vencimiento y la fecha actual, dividida entre las unidades de tiempo restantes.
ATC:	Cuando una máquina está disponible, se programan primero los trabajos con mayor costo aparente de tardanza.
M.SLACK:	Los trabajos se programan dependiendo del que tenga menor holgura.
PROMEDIO:	Valor promedio de la Tardanza Total Ponderada de las órdenes de trabajo al desarrollar diez (10) réplicas de Programación Inicial.

Tabla 14

Resultados del Subalgoritmo de Programación inicial para el objetivo de minimización del Objetivo Ponderado

RESULTADOS DEL SUBALGORITMO DE PROGRAMACIÓN INICIAL EN PetNMA PARA EL OBJETIVO PONDERADO (Continua… **)**								
Problema	**REGLAS DE DESPACHO**							
	SPT	**LPT**	**MRWT**	**EDD**	**CR**	**ATC**	**M. SLACK**	**Mínimo**
FMS01_6_10_19	218	232	218	232	218	218	232	218
FMS02_9_8_20	188,5	173	173	188,5	162	162	180,5	162
FMS03_10_5_10	1322,5	1641	1461,5	1517,5	1404,5	1381	1388	1322,5
FMS04_10_5_12	342,5	466	372	409,5	312	637,5	304,5	304,5
FMS05_10_5_12	351	570,5	431	431,5	428,5	483	373	351
FMS06_10_7_18	166	256	178,5	187,5	176	176	174,5	166
FMS07_10_10_20	1502	1546	1910,5	1217,5	1524,5	1785	1789	1217,5
FMS08_15_5_12	554,5	1220,5	751	611	523	773,5	497	497
FMS09_15_6_17	146	299	342	149	165	240	150	146
FMS10_20_6_13	4642,5	6375	6429	4080	5777	6083	4713	4080

(…continuación **)** **RESULTADOS DEL SUBALGORITMO DE PROGRAMACIÓN INICIAL EN PetNMA PARA EL OBJETIVO PONDERADO**											
Problema	**PROGRAMAS ACTIVOS CON REGLAS DE DESPACHO (PetNMA)**								**ALGORITMO GENÉTICO**		
	SPT	**LPT**	**MRWT**	**EDD**	**CR**	**ATC**	**M. SLACK**	**Mínimo**	**Mínimo**	**Máximo**	**Promedio**
FMS01_6_10_19	240	233	206	240	228	228	228	206	218	218	218
FMS02_9_8_20	184,5	178,5	163	188,5	172	198	180,5	163	162	162	162
FMS03_10_5_10	1372	1705	1477	2264,5	1395	1562	1752	1372	1114,5	1123	1115,35
FMS04_10_5_12	339	436,5	303	406,5	317,5	446,5	300,5	300,5	242,5	242,5	242,5
FMS05_10_5_12	493,5	823,5	696,5	692	597	585,5	563	493,5	303,5	320	306,75
FMS06_10_7_18	309	452	184	210,5	208,5	241,5	189,5	184	166	166	166
FMS07_10_10_20	2526	2138,5	2269,5	1471	1729	1912	2682	1471	784,5	847,5	823,1
FMS08_15_5_12	1289	1549,5	1210	1152,5	668,5	1349	814,5	668,5	358,5	404	376,7
FMS09_15_6_17	648	740	470,5	396	252	529,5	145,5	145,5	128	130,5	128,7
FMS10_20_6_13	5982,5	7565	7060,5	5077,5	6215,5	8339,5	5059,5	5059,5	3559,5	3735,5	3632,9

Donde:

SPT:	Cuando una máquina está disponible, se programan primero los trabajos con menor tiempo de proceso obtenido con la regla de despacho de mejor desempeño.
LPT:	Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo de proceso.
MRWT:	Cuando una máquina está disponible, se programan primero los trabajos con mayor tiempo remanente de proceso.
EDD:	Cuando una máquina está disponible, se programan primero los trabajos con menor tiempo de entrega.
CR:	Cuando una máquina está disponible, se programan primero los trabajos con menor diferencia entre la fecha de vencimiento y la fecha actual, dividida entre las unidades de tiempo restantes.
ATC:	Cuando una máquina está disponible, se programan primero los trabajos con mayor costo aparente de tardanza.
M.SLACK:	Los trabajos se programan dependiendo del que tenga menor holgura.
PROMEDIO:	Valor promedio de los Objetivos Ponderados de las órdenes de trabajo al desarrollar diez (10) réplicas de Programación Inicial.

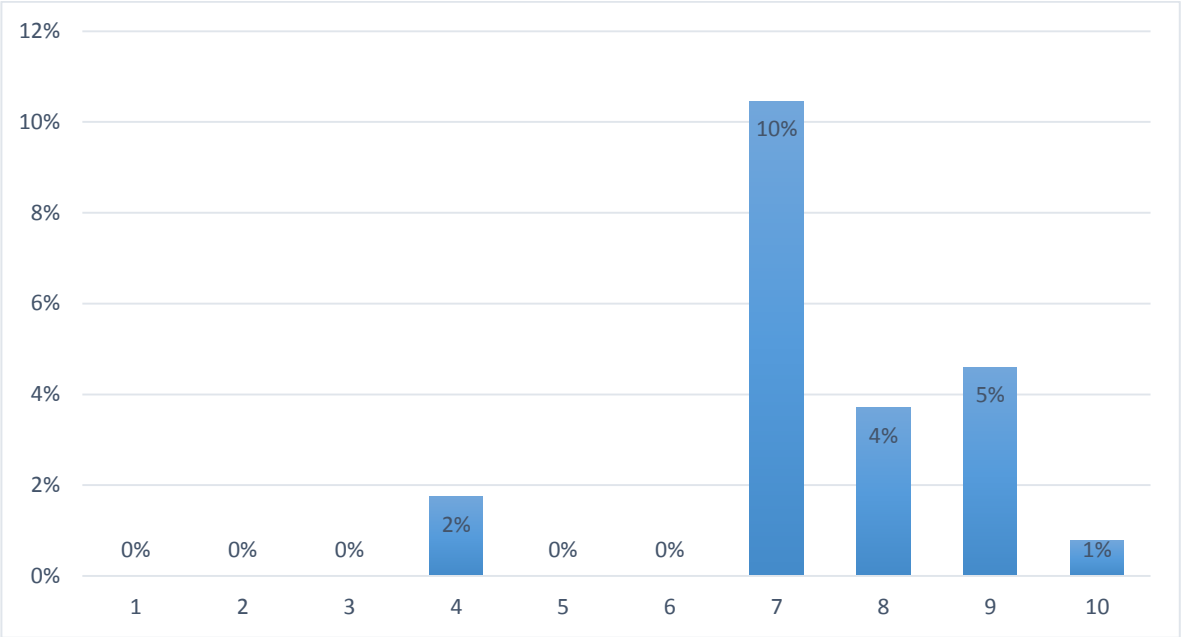


Figura 17. Variación del Makespan obtenido por el AG con respecto a las reglas de despacho.

De acuerdo a la figura 17, en la instancia 7 el algoritmo genético logró mejorar en un 10% el Makespan comparándolo con las reglas de despacho. Por otro lado, se logró mejorar el Makespan en el 50% de las instancias utilizando el algoritmo genético.

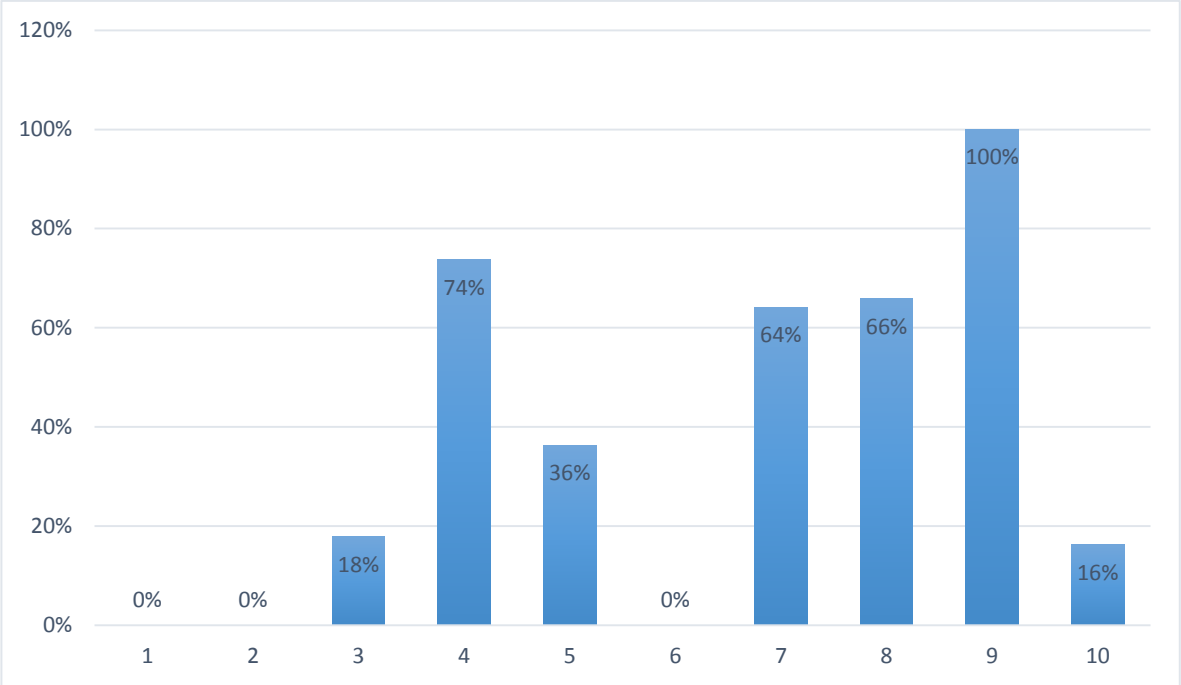


Figura 18. Variación de la TTP obtenido por el AG con respecto a las reglas de despacho.

Con respecto a la Tardanza Total Ponderada, el algoritmo genético logró mejorar el Makespan en el 70% de las instancias.

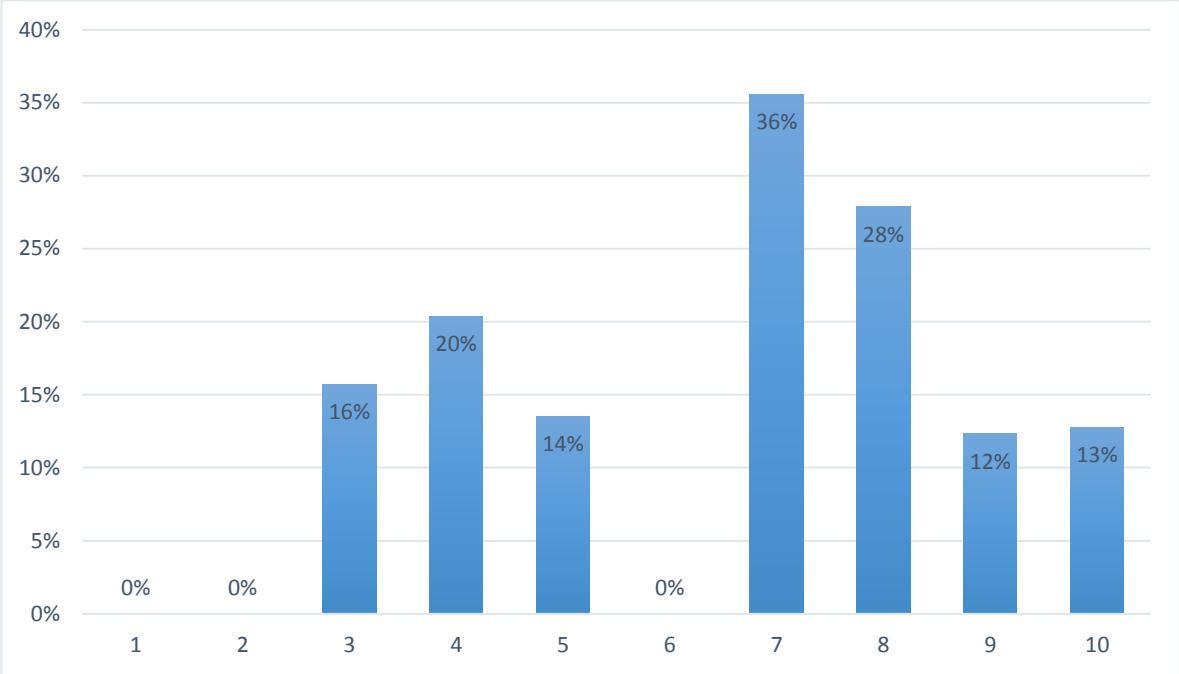


Figura 19. Variación del Objetivo Ponderado obtenido por el AG con respecto a las reglas de despacho.

El algoritmo genético de la programación inicial logró mejorar el resultado en el 70% de las instancias evaluando el Objetivo Ponderado.

3.2. Pruebas experimentales del subalgoritmo de programación reactiva en PetNMA

Con la finalidad de obtener los programas reactivos con el Subalgoritmo de Programación Reactiva en PetNMA, se realizó un diseño de experimentos utilizando los tres problemas seleccionados.
Para cada uno de los objetivos se ha realizado un diseño robusto Taguchi con el fin de escoger los mejores parámetros para el algoritmo genético teniendo en cuenta los factores que no se pueden controlar al momento de tener una vería de máquina. "El objetivo del diseño robusto de parámetros (Taguchi) es lograr productos y procesos robustos frente a las causas de variabilidad (ruidos), que hacen que las características funcionales de los productos se desvíen de sus valores óptimos provocando costos de calidad" (Gutiérrez, H. y De La Vara, R., 2008), por lo que al seleccionar éste tipo de diseño se busca realizar un análisis del efecto que tienen los factores de ruido (Factores que no podemos controlar) sobre los programas de producción y sus valores objetivos.

Los factores que se tuvieron en cuenta para evaluar el desempeño de algoritmo se encuentran en la tabla a continuación:

Tabla 15
Factores para evaluar el desempeño de algoritmo memético

Factores de control	Factores de ruido
A: Número de generaciones	*H: Instante de la falla*
B: Tamaño de la población	*I: Variación del tiempo de reparación*
C: Probabilidad de cruce	*J:Tiempo medio de reparación*
D: Probabilidad de mutación	
E: Intensidad de mutaciones	
F: Elitismo	
G: Factor de decrecimiento	

Se generaron escenarios de fallas constituidos por 2 medios dependiendo del momento en el horizonte de programación en el que se presente la falla, el porcentaje del promedio del tiempo de procesamiento que se utiliza para obtener el tiempo de reparación de la máquina y la variación del tiempo de reparación. Los escenarios se pueden identificar por medio del siguiente sistema de codificación: $B(Cx, \gamma_1, \gamma_2)$ en donde, Cx corresponde al medio y toma valores de 1 a 2, γ_1 corresponde al porcentaje del promedio del tiempo de procesamiento y toma valores 0.5 y 1.0, y γ_2 representa la variación con valores de 0.02 y 0.05.

Tabla 16

Parámetros de PetNMA para el Makespan

Parámetros de PetNMA	Nomenclatura	Valores para el programa inicial (Problemas pequeños y medianos)	Valores para el programa inicial (Problemas grandes)	Valores para parametrizar en la programación reactiva
Numero de Generaciones	***maxgen***	100	100	100, 200
Tamaño de la Población	***P***	100	100	50, 100
Probabilidad de Cruce	***Tc***	60	60	0.6, 0.9
Probabilidad de Mutación	***Tm***	50	50	0.1, 0.5
Intensidad de Mutaciones	***Im***	2	2	1,2
Elitismo	Te_1	ON(50%)	Off	Off, On (50%)
Factor de decrecimiento				0.01, 0.1
Instante de la falla				1, 2
Variación del tiempo de reparación				0.02, 0.05
Tiempo medio de reparación				0.5, 1.0

Tabla 17

Parámetros de PetNMA para la Tardanza Total Ponderada

Parámetros de PetNMA	Nomenclatura	Valores para el programa inicial (Todos los problemas)	Valores para parametrizar en la programación reactiva TTP
Numero de Generaciones	***maxgen***	200	100, 200
Tamaño de la Población	***P***	100	50, 100
Probabilidad de Cruce	***Tc***	90	0.6, 0.9
Probabilidad de Mutación	***Tm***	50	0.1, 0.5
Intensidad de Mutaciones	***Im***	4	2, 4
Elitismo	Te_1	On (50%)	Off, On (50%)
Factor de decrecimiento			0.01, 0.1
Instante de la falla			1, 2
Variación del tiempo de reparación			0.02, 0.05
Tiempo medio de reparación			0.5, 1.0

Tabla 18

Parámetros de PetNMA para el Objetivo Ponderado

Parámetros de PetNMA para la programación inicial	Nomenclatura	Valores para el programa inicial (Problemas pequeños y medianos)	Valores para el programa inicial (Problemas grandes)	Valores para parametrizar en la programación reactiva (Problemas medianos)	Valores para parametrizar en la programación reactiva (Problemas grandes)
Numero de Generaciones	***Maxgen***	200	200	100, 200	100, 200
Tamaño de la Población	***P***	100	100	50, 100	50, 100
Probabilidad de Cruce	***Tc***	60	90	0.6, 0.9	0.6, 0.9
Probabilidad de Mutación	***Tm***	50	50	0.1, 0.5	0.1, 0.5
Intensidad de Mutaciones	***Im***	4	4	2, 4	1, 4
Elitismo	Te_1	On (50%)	On (50%)	Off, On (50%)	Off, On (50%)
Factor de decrecimiento				0.01, 0.1	0.01, 0.1
Instante de la falla				1, 2	1, 2
Variación del tiempo de reparación				0.02, 0.05	0.02, 0.05
Tiempo medio de reparación				0.5, 1.0	0.5, 1.0

A continuación, se pueden observar los resultados de las corridas de PetNMA para la selección de los valores de los parámetros para el algoritmo memético del Subalgoritmo de Programación Reactiva:

Tabla 19

Resultados del diseño de experimentos robusto (Taguchi) para la evaluación del Makespan del problema pequeño

							Factores de ruido	H	1	2	2	1	Media	Desviación Estándar	Razón señal/ruido
								I	0.5	0.5	1.0	1.0			
								J	0.02	0.05	0.02	0.05			
Factores controlables													$\bar{Y}$	S	$-10\log\left[\frac{1}{n}\sum_{i=1}^{n} Y_i^2\right]$
A	B	C	D	E	F	G									
100	50	60	10	1	0	0.01		**2,78**	**13,58**	**11,73**	**0,00**	7,02	6,65	-19,16
100	50	60	50	2	50	0.1		**5,25**	**0,00**	**0,00**	**4,01**	2,31	2,72	-10,38
100	100	90	10	1	50	0.1		**0,00**	**0,00**	**0,00**	**0,00**	0,00	0,00	0,00
100	100	90	50	2	0	0.01		**5,25**	**0,00**	**0,31**	**0,00**	1,39	2,58	-8,39
200	50	90	10	2	0	0.1		**0,00**	**0,00**	**6,17**	**5,56**	2,93	3,40	-12,37
200	50	90	50	1	50	0.01		**0,00**	**4,94**	**0,00**	**0,00**	1,23	2,47	-7,85
200	100	60	10	2	50	0.01		**0,00**	**0,00**	**2,78**	**0,00**	0,69	1,39	-2,85
200	100	60	50	1	0	0.1		**6,79**	**7,10**	**5,56**	**0,00**	4,86	3,31	-15,03

Tabla 20

Resultados del diseño de experimentos robusto (Taguchi) para la evaluación del Makespan del problema mediano

							Factores de ruido	H	1	2	2	1	Media	Desviación Estándar	Razón señal/ruido
								I	0.5	0.5	1.0	1.0			
								J	0.02	0.05	0.02	0.05			
Factores controlables													$\bar{Y}$	S	$-10\log\left[\frac{1}{n}\sum_{i=1}^{n} Y_i^2\right]$
A	B	C	D	E	F	G									
100	50	60	10	1	0	0.01			**9,08**	**7,06**	**0,00**	**6,18**	5,58	3,91	-16,30
100	50	60	50	2	50	0.1			**14,39**	**0,00**	**13,35**	**4,41**	8,04	6,98	-20,05
100	100	90	10	1	50	0.1			**9,58**	**0,00**	**2,92**	**12,07**	6,14	5,63	-17,89
100	100	90	50	2	0	0.01			**1,98**	**0,30**	**0,00**	**0,50**	0,69	0,88	-0,26
200	50	90	10	2	0	0.1			**7,37**	**0,00**	**0,00**	**0,00**	1,84	3,68	-11,33
200	50	90	50	1	50	0.01			**7,92**	**2,93**	**1,98**	**7,65**	5,12	3,11	-15,24
200	100	60	10	2	50	0.01			**0,98**	**6,08**	**4,86**	**2,80**	3,68	2,25	-12,39
200	100	60	50	1	0	0.1			**13,06**	**0,00**	**7,28**	**0,00**	5,09	6,33	-17,48

Tabla 21

Resultados del diseño de experimentos robusto (Taguchi) para la evaluación del Makespan del problema grande

							Factores de ruido	H	1	2	2	1	Media	Desviación Estándar	Razón señal/ruido
								I	0.5	0.5	1.0	1.0			
								J	0.02	0.05	0.02	0.05			
Factores controlables															
A	B	C	D	E	F	G							$\bar{Y}$	S	$-10\log\left[\frac{1}{n}\sum_{i=1}^{n} Y_i^{\,2}\right]$
100	50	60	10	1	0	0.01		**2,37**	**7,87**	**6,69**	**1,97**	4,73	3,00	-14,63
100	50	60	50	2	50	0.1		**5,20**	**3,59**	**9,45**	**4,00**	5,56	2,68	-15,60
100	100	90	10	1	50	0.1		**1,60**	**14,96**	**2,36**	**5,14**	6,02	6,15	-18,10
100	100	90	50	2	0	0.01		**4,02**	**16,54**	**12,20**	**3,92**	9,17	6,26	-20,55
200	50	90	10	2	0	0.1		**6,30**	**0,00**	**15,75**	**9,84**	7,97	6,59	-19,83
200	50	90	50	1	50	0.01		**7,48**	**16,21**	**9,84**	**2,36**	8,97	5,74	-20,22
200	100	60	10	2	50	0.01		**4,33**	**3,15**	**9,88**	**2,39**	4,94	3,39	-15,19
200	100	60	50	1	0	0.1		**3,60**	**11,76**	**14,57**	**1,97**	7,98	6,14	-19,63

Tabla 22

Resultados del diseño de experimentos robusto (Taguchi) para la evaluación de la TTP del problema pequeño

							Factores de ruido	H	1	2	2	1	Media	Desviación Estándar	Razón señal/ruido
								I	0.5	0.5	1.0	1.0			
								J	0.02	0.05	0.02	0.05			
Factores controlables													$\bar{Y}$	S	$-10\log\left[\frac{1}{n}\sum_{i=1}^{n} Y_i^2\right]$
A	B	C	D	E	F	G									
100	50	60	10	2	0	0.01		**0,00**	**0,00**	**0,00**	**0,00**	0,00	0,00	0,00	
100	50	60	50	4	50	0.1		**0,00**	**0,00**	**0,00**	**0,00**	0,00	0,00	0,00	
100	100	90	10	2	50	0.1		**0,00**	**0,00**	**0,00**	**0,00**	0,00	0,00	0,00	
100	100	90	50	4	0	0.01		**0,00**	**0,00**	**0,00**	**0,00**	0,00	0,00	0,00	
200	50	90	10	4	0	0.1		**0,00**	**0,00**	**0,00**	**0,00**	0,00	0,00	0,00	
200	50	90	50	2	50	0.01		**0,00**	**0,00**	**0,00**	**0,00**	0,00	0,00	0,00	
200	100	60	10	4	50	0.01		**0,00**	**0,00**	**0,00**	**0,00**	0,00	0,00	0,00	
200	100	60	50	2	0	0.1		**0,00**	**0,00**	**0,00**	**0,00**	0,00	0,00	0,00	

Tabla 23

Resultados del diseño de experimentos robusto (Taguchi) para la evaluación de la TTP del problema mediano

							Factores de ruido	H	1	2	2	1	Media	Desviación estándar	Razón señal/ruido
								I	0.5	0.5	1.0	1.0			
								J	0.02	0.05	0.02	0.05			
Factores controlables															
A	B	C	D	E	F	G							$\bar{Y}$	S	$-10\log\left[\frac{1}{n}\sum_{i=1}^{n} Y_i^2\right]$
100	50	60	10	2	0	0.01			**110,43**	**0,00**	**16,23**	**161,63**	72,07	77,04	-39,84
100	50	60	50	4	50	0.1			**45,18**	**8,36**	**49,06**	**110,59**	53,30	42,37	-36,22
100	100	90	10	2	50	0.1			**-3,07**	**47,11**	**7,35**	**235,01**	71,60	111,06	-41,58
100	100	90	50	4	0	0.01			**50,00**	**5,63**	**67,22**	**33,11**	38,99	26,24	-33,09
200	50	90	10	4	0	0.1			**120,65**	**16,73**	**0,00**	**79,87**	54,31	56,02	-37,25
200	50	90	50	2	50	0.01			**64,53**	**8,62**	**5,81**	**94,22**	43,30	43,40	-35,17
200	100	60	10	4	50	0.01			**30,47**	**0,55**	**3,75**	**99,67**	33,61	46,04	-34,34
200	100	60	50	2	0	0.1			**55,17**	**50,08**	**4,82**	**0,00**	27,52	29,13	-31,44

Tabla 24

Resultados del diseño de experimentos robusto (Taguchi) para la evaluación de la TTP del problema grande

							Factores de ruido	H	1	2	2	1	Media	Desviación estándar	Razón señal/ruido
								I	0.5	0.5	1.0	1.0			
								J	0.02	0.05	0.02	0.05			
Factores controlables															
A	B	C	D	E	F	G							$\bar{Y}$	S	$-10\log\left[\frac{1}{n}\sum_{i=1}^{n} Y_i^2\right]$
100	50	60	10	2	0	0.01		**2100**	**0**	**0**	**2100**	1050,00	1212,44	-63,43
100	50	60	50	4	50	0.1		**8100**	**3300**	**8400**	**7500**	6825,00	2379,60	-77,06
100	100	90	10	2	50	0.1		**2100**	**0**	**5100**	**3900**	2775,00	2223,17	-70,57
100	100	90	50	4	0	0.01		**0**	**5400**	**0**	**4700**	2525,00	2929,59	-71,08
200	50	90	10	4	0	0.1		**0**	**0**	**0**	**1300**	325,00	650,00	-56,26
200	50	90	50	2	50	0.01		**0**	**0**	**0**	**0**	0,00	0,00	0,00
200	100	60	10	4	50	0.01		**11400**	**0**	**6000**	**2100**	4875,00	5010,24	-76,29
200	100	60	50	2	0	0.1		**100**	**0**	**600**	**1200**	475,00	550,00	-56,56

Tabla 25

Resultados del diseño de experimentos robusto (Taguchi) para la evaluación del Objetivo Ponderado del problema pequeño

							Factores de ruido	H	1	2	2	1	Media	Desviación estándar	Razón señal/ruido
								I	0.5	0.5	1.0	1.0			
								J	0.02	0.05	0.02	0.05			
Factores controlables													$\bar{Y}$	S	$-10\log\left[\frac{1}{n}\sum_{i=1}^{n} Y_i^{\,2}\right]$
A	B	C	D	E	F	G									
100	50	60	10	2	0	0.01		**0,00**	**12,65**	**11,42**	**4,32**	7,10	5,99	-18,88
100	50	60	50	4	50	0.1		**4,32**	**2,16**	**10,49**	**0,00**	4,24	4,52	-15,23
100	100	90	10	2	50	0.1		**0,00**	**0,00**	**12,65**	**0,00**	3,16	6,33	-16,02
100	100	90	50	4	0	0.01		**0,00**	**13,89**	**3,09**	**0,00**	4,24	6,59	-17,04
200	50	90	10	4	0	0.1		**0,00**	**0,00**	**0,00**	**3,09**	0,77	1,54	-3,77
200	50	90	50	2	50	0.01		**5,25**	**0,00**	**1,54**	**8,33**	3,78	3,75	-13,95
200	100	60	10	4	50	0.01		**0,00**	**0,00**	**12,65**	**0,00**	3,16	6,33	-16,02
200	100	60	50	2	0	0.1		**0,00**	**13,89**	**3,09**	**0,00**	4,24	6,59	-17,04

Tabla 26

Resultados del diseño de experimentos robusto (Taguchi) para la evaluación del Objetivo Ponderado del problema mediano

							Factores de ruido	H	1	2	2	1	Media	Desviación estándar	Razón señal/ruido
								I	0.5	0.5	1.0	1.0			
								J	0.02	0.05	0.02	0.05			
Factores controlables															
A	B	C	D	E	F	G							$\bar{Y}$	S	$-10\log\left[\frac{1}{n}\sum_{i=1}^{n} Y_i^{\,2}\right]$
100	50	60	10	2	0	0.01		**11,28**	**1,31**	**4,36**	**87,61**		26,14	41,19	-32,91
100	50	60	50	4	50	0.1		**50,15**	**11,33**	**40,96**	**12,54**		28,74	19,77	-30,49
100	100	90	10	2	50	0.1		**30,15**	**0,45**	**1,73**	**30,69**		15,75	16,94	-26,66
100	100	90	50	4	0	0.01		**22,63**	**11,48**	**3,95**	**5,61**		10,92	8,45	-22,37
200	50	90	10	4	0	0.1		**11,74**	**56,55**	**48,13**	**9,88**		31,58	24,24	-31,58
200	50	90	50	2	50	0.01		**5,50**	**9,69**	**25,80**	**32,86**		18,46	12,99	-26,70
200	100	60	10	4	50	0.01		**90,57**	**4,24**	**0,00**	**20,81**		28,90	42,08	-33,35
200	100	60	50	2	0	0.1		**33,86**	**31,67**	**5,06**	**107,44**		44,51	43,95	-35,35

Tabla 27

Resultados del diseño de experimentos robusto (Taguchi) para la evaluación del Objetivo Ponderado del problema grande

							Factores de ruido	H	1	2	2	1	Media	Desviación estándar	Razón señal/ruido
								I	0.5	0.5	1.0	1.0			
								J	0.02	0.05	0.02	0.05			
Factores controlables															
A	B	C	D	E	F	G							$\bar{Y}$	S	$-10\log\left[\frac{1}{n}\sum_{i=1}^{n} Y_i^2\right]$
100	50	60	10	1	0	0.01		**1,95**	**0,00**	**17,51**	**44,92**		16,10	20,75	-27,65
100	50	60	50	4	50	0.1		**7,03**	**7,00**	**12,11**	**145,70**		42,96	68,54	-37,30
100	100	90	10	1	50	0.1		**21,40**	**0,00**	**0,00**	**69,14**		22,64	32,60	-31,17
100	100	90	50	4	0	0.01		**27,48**	**61,87**	**20,70**	**147,27**		64,33	58,15	-38,24
200	50	90	10	4	0	0.1		**27,13**	**9,38**	**17,69**	**39,45**		23,41	12,92	-28,28
200	50	90	50	1	50	0.01		**14,84**	**1,56**	**20,70**	**1,56**		9,67	9,66	-22,13
200	100	60	10	4	50	0.01		**43,58**	**15,23**	**2,34**	**7,42**		17,14	18,40	-27,39
200	100	60	50	1	0	0.1		**22,27**	**0,00**	**3,83**	**24,22**		12,58	12,44	-24,38

De acuerdo a los resultados obtenidos en las corridas se han seleccionado los siguientes valores para el algoritmo memético del Subalgoritmo de Programación Rectiva en PetNMA:

Tabla 28

Valores seleccionados para el algoritmo memético del Subalgortimo de Programación Reactiva

	OBJETIVO EVALUADO								
	MAKESPAN			TARDANZA TOTAL PONDERADA			OBJETIVO PONDERADO (50%-50%)		
FACTOR DE CONTROL	Problema Pequeño	Problema Mediano	Problema Grande	Problema Pequeño	Problema Mediano	Problema Grande	Problema Pequeño	Problema Mediano	Problema Grande
Número de generaciones	100	100	100	200	200	200	200	100	200
Tamaño de población	100	100	50	100	100	50	50	100	50
Probabilidad de cruce (%)	90	90	60	60	60	90	90	90	90
Probabilidad de mutación (%)	10	50	10	50	50	50	10	50	50
Intensidad de mutación	1	2	1	2	2	2	4	4	1
Porcentaje de elitismo (%)	50	0	0	0	0	50	0	0	50
Factor de decrecimiento de la temperatura	0.1	0.01	0.01	0.1	0.1	0.01	0.1	0.01	0.01

El problema pequeño obtiene siempre el mejor resultado tanto para el makespan como para la tardanza total ponderada en la primera generación debido a que son pocas operaciones las que se deben programar en pocas máquinas por lo que las reglas de despacho obtienen buenos programas. Con respecto a la tardanza total ponderada, tanto el problema pequeño como el grande obtienen buenas respuestas, el problema mediano tiene tardanzas elevadas debido a que la mayoría de los trabajos inician en las mismas estaciones por lo que se embotellan en ellas. Los resultados y gráficas de los diseños de experimentos realizados se pueden observar en los anexos.

A continuación se muestran los resultados de una corrida de cada uno de los 10 problemas estudiados con los parámetros escogidos para

cada problema:

Tabla 29

Resultados del PetNMA para la evaluación del Makespan (Instancias 1 a 5)

MAKESPAN											
Instancia	**Escenario de la falla**	**Programación Inicial**	**SPT**	**LPT**	**MRWT**	**EDD**	**CR**	**ATC**	**M. SLACK**	**Algoritmo Memético**	**Variación (%)**
FMS01_6_10_19	1	412	468	468	468	1566	1497	1578	1000	452	9,71
	2	412	413	413	413	826	751	810	630	413	0,24
	3	412	412	412	412	537	492	513	487	412	0,00
	4	412	412	412	412	1162	1074	1141	695	412	0,00
FMS02_9_8_20	1	324	324	324	324	2215	2009	2217	591	324	0,00
	2	324	324	324	324	647	578	636	489	324	0,00
	3	324	324	324	324	533	505	529	400	324	0,00
	4	324	324	375	324	1467	1296	1454	636	324	0,00
FMS03_10_5_10	1	536	632	624	536	1711	1388	1726	949	536	0,00
	2	536	623	623	623	623	623	623	623	623	16,23
	3	536	536	536	536	573	573	573	537	536	0,00
	4	536	719	711	623	1581	1264	1522	1001	623	16,23
FMS04_10_5_12	1	394	406	426	394	1655	1471	1675	1136	394	0,00
	2	394	394	394	394	918	763	918	762	394	0,00
	3	394	394	394	394	873	763	868	736	394	0,00
	4	394	417	432	401	2224	1934	2247	1162	394	0,00
FMS05_10_5_12	1	379	437	472	406	1665	1314	1672	856	406	7,12
	2	379	408	408	408	417	408	408	417	408	7,65
	3	379	420	420	420	420	431	431	431	420	10,82
	4	379	491	472	438	2222	2048	2198	807	438	15,57

Tabla 30

Resultados del PetNMA para la evaluación del Makespan (Instancias 6 a 10)

MAKESPAN											
Instancia	**Escenario de la falla**	**Programación Inicial**	**SPT**	**LPT**	**MRWT**	**EDD**	**CR**	**ATC**	**M. SLACK**	**Algoritmo Memético**	**Variación (%)**
FMS06_10_7_18	1	332	332	349	332	1536	1279	1541	581	332	0,00
	2	332	404	439	367	1226	1040	1238	573	359	8,13
	3	332	332	335	332	1133	858	1117	566	332	0,00
	4	332	332	351	333	2314	2030	2278	632	332	0,00
FMS07_10_10_20	1	1012	1032	1091	1061	3878	3704	3884	2780	1032	1,98
	2	1012	1015	1051	1015	1998	1806	1969	1573	1015	0,30
	3	1002	1002	1002	1002	1808	1592	1811	1401	1002	0,00
	4	1002	1007	1089	1007	3746	3502	3686	2660	1007	0,50
FMS08_15_5_12	1	451	464	489	477	1806	1471	1772	1073	464	2,88
	2	451	544	544	544	544	544	544	544	544	20,62
	3	456	583	526	493	1479	1351	1523	1012	493	8,11
	4	448	580	586	514	2842	2441	2871	1335	499	11,38
FMS09_15_6_17	1	253	264	269	259	1736	1513	1754	861	259	2,37
	2	254	274	274	274	288	299	299	299	274	7,87
	3	254	271	271	271	286	299	299	299	271	6,69
	4	254	273	276	259	1578	1282	1578	930	259	1,97
FMS10_20_6_13	1	753	758	767	753	1634	1422	1605	1478	753	0,00
	2	753	804	839	782	1274	1130	1274	1293	782	3,85
	3	753	813	813	813	813	813	813	813	813	7,97
	4	753	758	935	754	2722	2387	2698	2291	754	0,13
									Variación Promedio		**4,21**

Tabla 31

Resultados del PetNMA para la evaluación de la Tardanza Total Ponderada (Instancias 1 a 5)

TARDANZA TOTAL PONDERADA											
Instancia	**Escenario de la falla**	**Programación Inicial**	**SPT**	**LPT**	**MRWT**	**EDD**	**CR**	**ATC**	**M. SLACK**	**Algoritmo Memético**	**Variación (%)**
FMS01_6_10_19	1	0	0	0	0	4452	5962	5086	549	0	0,00
	2	0	0	0	0	407	1179	720	160	0	0,00
	3	0	0	0	0	2	0	0	0	0	0,00
	4	0	0	0	0	4349	5532	4951	978	0	0,00
FMS02_9_8_20	1	0	0	0	0	9700	9690	11525	2997	0	0,00
	2	0	0	0	0	0	0	0	0	0	0,00
	3	0	0	0	0	78	0	0	0	0	0,00
	4	0	0	0	0	10433	8994	11349	2003	0	0,00
FMS03_10_5_10	1	1577	1602	1832	1728	10215	9791	10539	4415	1602	1,59
	2	1577	1721	1721	1721	1721	1721	1721	1721	1721	9,13
	3	1577	1625	1625	1625	1820	1644	1820	1761	1625	3,04
	4	1577	1814	2350	2273	18336	11920	15360	9844	1751	11,03
FMS04_10_5_12	1	40	58	58	58	4188	5518	6625	2439	40	0,00
	2	40	79	64	79	808	1115	1311	1008	64	60,00
	3	40	108	108	108	1461	2977	3262	1208	108	170,00
	4	40	211	660	480	19068	19116	18957	4644	211	427,50
FMS05_10_5_12	1	155	390	558	694	16057	15124	14067	4848	390	151,61
	2	155	326	326	326	587	794	794	587	326	110,32
	3	155	392	392	392	392	392	392	392	392	152,90
	4	155	339	567	577	10366	11112	10916	6662	339	118,71

Tabla 32

Resultados del PetNMA para la evaluación de la Tardanza Total Ponderada (Instancias 6 a 10)

TARDANZA TOTAL PONDERADA											
Instancia	**Escenario de la falla**	**Programación Inicial**	**SPT**	**LPT**	**MRWT**	**EDD**	**CR**	**ATC**	**M. SLACK**	**Algoritmo Memético**	**Variación (%)**
FMS06_10_7_18	1	0	3	10	0	17552	18933	20755	970	0	0,00
	2	0	14	14	14	38	92	92	26	14	1400,00
	3	0	10	93	0	6386	5559	6605	1190	0	0,00
	4	0	10	93	0	6313	6363	7370	1486	0	0,00
FMS07_10_10_20	1	580	900	1216	1433	30793	35793	37894	11996	900	55,17
	2	623	935	935	935	1113	1572	1572	1113	935	50,08
	3	623	653	653	653	4551	5703	9386	2295	653	4,82
	4	565	806	743	565	12214	19047	20570	8494	565	0,00
FMS08_15_5_12	1	6302	8390	8263	8764	20039	17132	22439	15851	7120	12,98
	2	6371	6415	6424	6415	6446	6484	6441	6446	6415	0,69
	3	6304	6308	6308	6308	6351	6370	6474	6364	6308	0,06
	4	6602	9351	9319	9903	22234	26654	31504	17629	7846	18,84
FMS09_15_6_17	1	0	0	100	270	39606	30797	35605	11571	0	0,00
	2	0	0	0	0	258	728	728	258	0	0,00
	3	0	0	0	0	816	1216	1328	700	0	0,00
	4	0	12	0	12	8784	8577	11671	3643	0	0,00
FMS10_20_6_13	1	6187	8486	10114	12188	41807	61377	72272	15509	7501	21,24
	2	6660	6714	6771	6714	8752	8109	9562	8898	6714	0,81
	3	6465	6475	6475	6475	6475	6475	6475	6475	6475	0,15
	4	6420	8318	10100	11189	31016	45209	63663	19604	7454	16,11
									Variación Promedio		**69,92**

Tabla 33

Resultados del PetNMA para la evaluación del Objetivo Ponderado (50% del Makespan y 50% de la Tardanza Total Ponderada) (Instancias 1 a 5)

OBJETIVO PONDERADO (50% - 50%)											
Instancia	**Escenario de la falla**	**Programación Inicial**	**SPT**	**LPT**	**MRWT**	**EDD**	**CR**	**ATC**	**M. SLACK**	**Algoritmo Memético**	**Variación (%)**
FMS01_6_10_19	1	206	222	232	222	2519	3459	3126	1376,5	213	3,40
	2	206	206	206	206	296	368	370,5	253,5	206	0,00
	3	206	209	209	209	227	246,5	246,5	223	209	1,46
	4	206	220	224	224	1814,5	1628	1733,5	844,5	220	6,80
FMS02_9_8_20	1	162	162	162	162	6355,5	5001,5	7551,5	790	162	0,00
	2	162	162	162	162	1580	1461	2017,5	229,5	162	0,00
	3	162	162	162	162	452	729	907,5	240	162	0,00
	4	162	168,5	167	167	9826,5	7813,5	9952,5	478,5	167	3,09
FMS03_10_5_10	1	1114,5	1559,5	1686,5	1569	5821,5	4393	4736,5	2535,5	1490,5	33,74
	2	1114,5	1338	1314,5	1300,5	3011	2103,5	3672	1674,5	1300,5	16,69
	3	1123	1123	1123	1123	1594,5	1290	1614	1240	1123	0,00
	4	1114,5	1284	1481,5	1404,5	12349	10229,5	10847,5	4929,5	1284	15,21
FMS04_10_5_12	1	242,5	313,5	348	351	9730	9588	10172	4516	271	11,75
	2	242,5	242,5	242,5	242,5	894,5	890,5	1046	532	242,5	0,00
	3	242,5	242,5	242,5	242,5	259,5	295	295	270	242,5	0,00
	4	242,5	344,5	433,5	296,5	10458	9714	10139	4654	284,5	17,32
FMS05_10_5_12	1	306,5	389,5	379	382	7171	6262	6309	3213	379	23,65
	2	307	374	374	374	432	432	432	432	374	21,82
	3	306,5	324	324	324	532	447,5	447,5	432,5	324	5,71
	4	307,5	409	522	497,5	6498	6651	6458,5	4973	393	27,80

Tabla 34

Resultados del PetNMA para la evaluación del Objetivo Ponderado (50% del Makespan y 50% de la Tardanza Total Ponderada) (Instancias 6 a 10)

OBJETIVO PONDERADO (50% - 50%)											
Instancia	**Escenario de la falla**	**Programación Inicial**	**SPT**	**LPT**	**MRWT**	**EDD**	**CR**	**ATC**	**M. SLACK**	**Algoritmo Memético**	**Variación (%)**
FMS06_10_7_18	1	166	171	174,5	171	6256,5	6059	6718,5	2239,5	171	3,01
	2	166	192	180,5	178,5	3736,5	4332,5	4431,5	877	178,5	7,53
	3	166	182	190	182	2026	2131,5	2662,5	658,5	182	9,64
	4	166	206,5	198	175,5	8319	9067,5	9477	936,5	175,5	5,72
FMS07_10_10_20	1	833	1064	1292	1021,5	7489	8891,5	8768	3740	1021,5	22,63
	2	823	917,5	917,5	917,5	1119,5	1342,5	1342,5	1119,5	917,5	11,48
	3	849	888	882,5	888	1009,5	1415,5	1611	1263	882,5	3,95
	4	838	1008	1513,5	1167,5	17187	17997,5	19797	9656	885	5,61
FMS08_15_5_12	1	386,5	559	742	735,5	16469	16095	19489,5	4278,5	559	44,63
	2	390	465,5	465,5	465,5	1852,5	2191	2723,5	940,5	465,5	19,36
	3	389	422	422	422	1057,5	908,5	908,5	669	422	8,48
	4	389	575	763	788,5	5443	6725	9196,5	2788	559,5	43,83
FMS09_15_6_17	1	128	195,5	245,5	227,5	13500,5	12903	13781	4510,5	147	14,84
	2	128	130	130	130	137,5	145,5	145,5	142	130	1,56
	3	128	175	164,5	161	2961	2712,5	3650	1085	154,5	20,70
	4	128	130,5	146	204	8533,5	8093,5	9545,5	3243,5	130	1,56
FMS10_20_6_13	1	3707,5	4911,5	6044	6446,5	16751,5	25896,5	28613,5	8555,5	4479,5	20,82
	2	3589	3665,5	3665,5	3665,5	3833,5	3696	3767	3833,5	3665,5	2,13
	3	3609,5	3806,5	3924,5	4158	8959	7162,5	9012	6029	3608,5	-0,03
	4	3721	4661	5643	6199	22585	35414,5	36346,5	10281,5	4147,5	11,46
									Variación Promedio		**11,18**

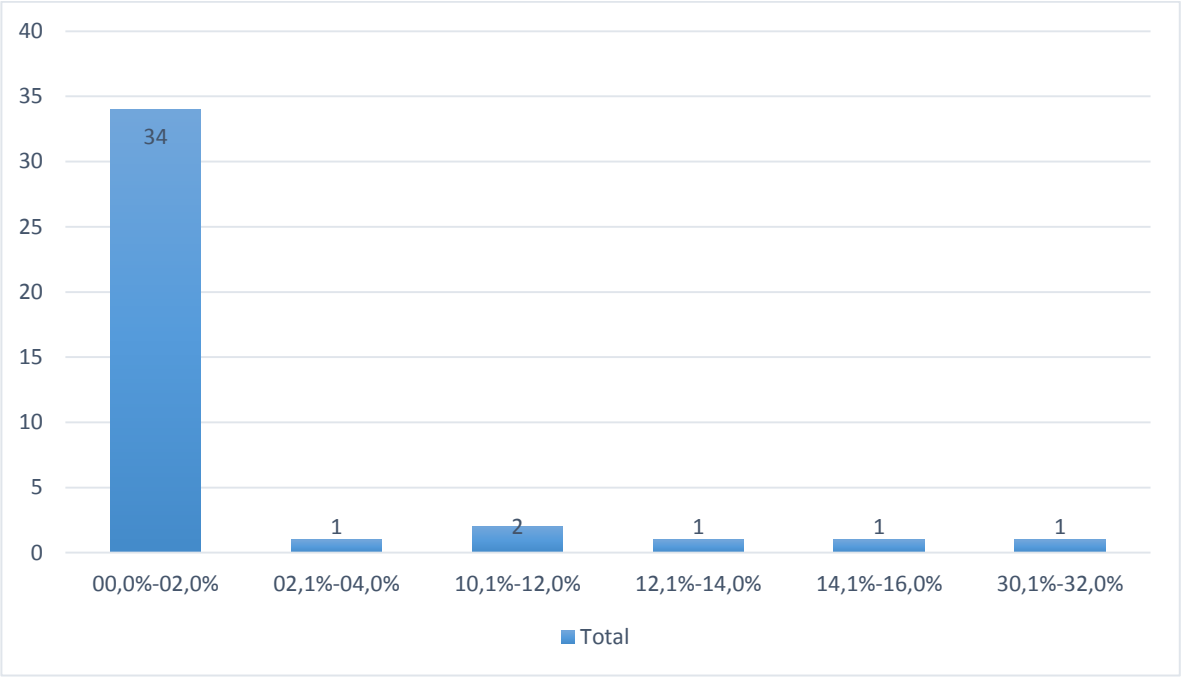


Figura 20. Porcentaje de Mejora del objetivo final con respecto a las reglas de despacho - OBJETIVO PONDERADO.

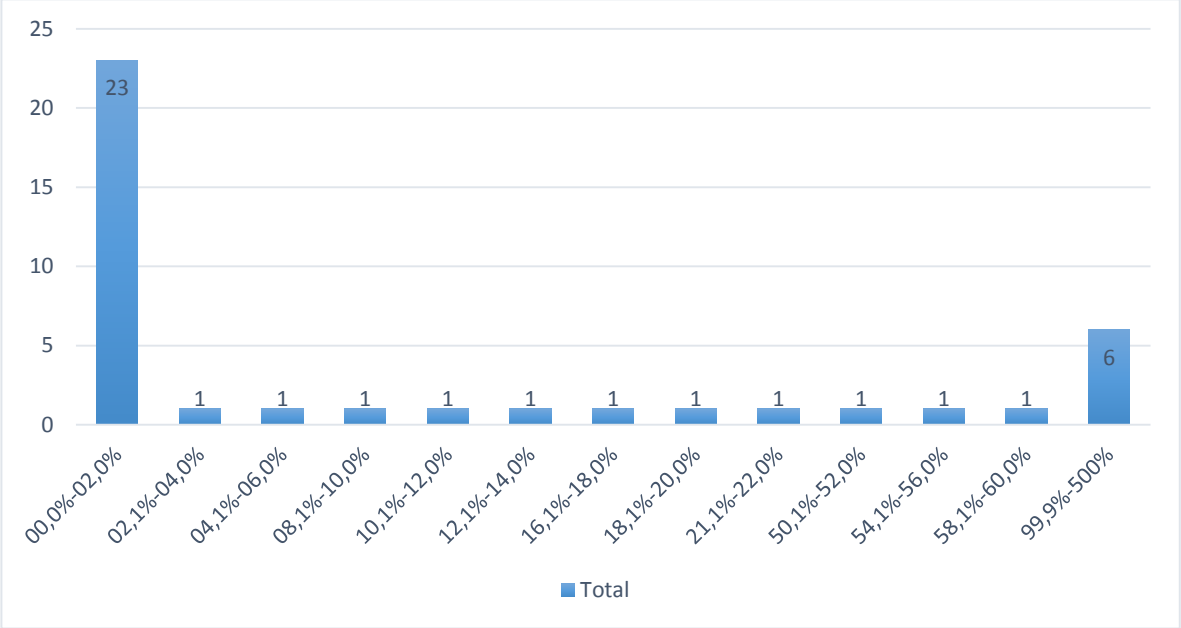


Figura 21. Porcentaje de variación del objetivo final con respecto a la programación inicial - OBJETIVO PONDERADO.

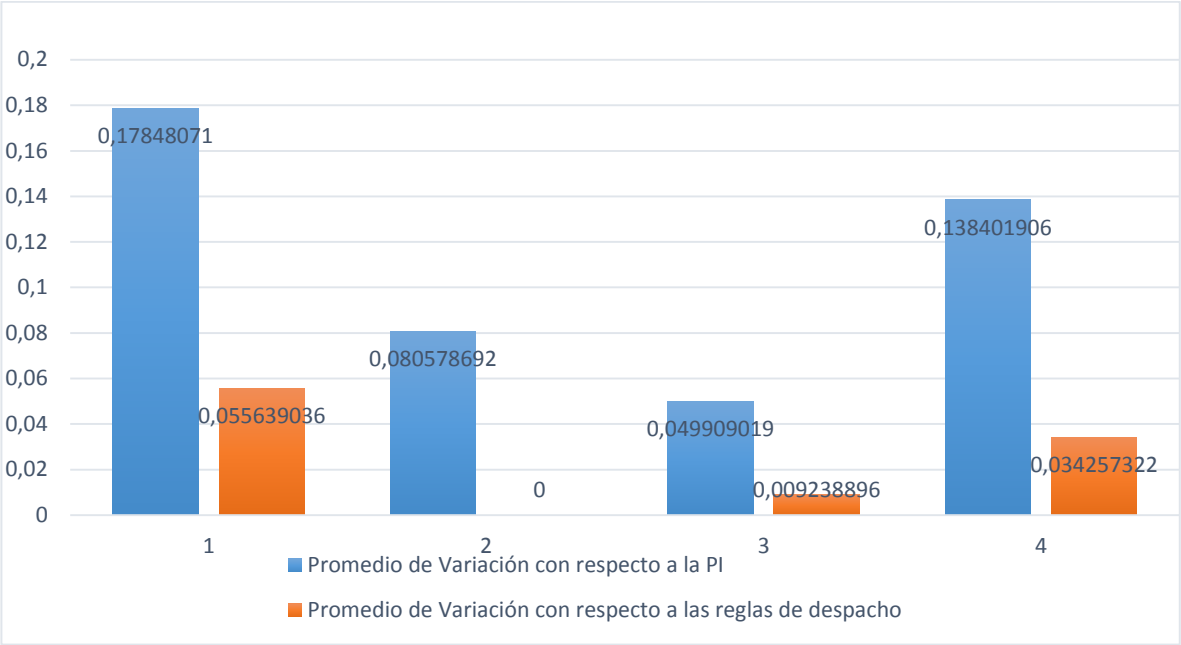


Figura 22. Promedio de variación del objetivo final con respecto a la porgramación inicial y a las reglas de despacho en los escenarios de fallas de máquinas - OBJETIVO PONDERADO.

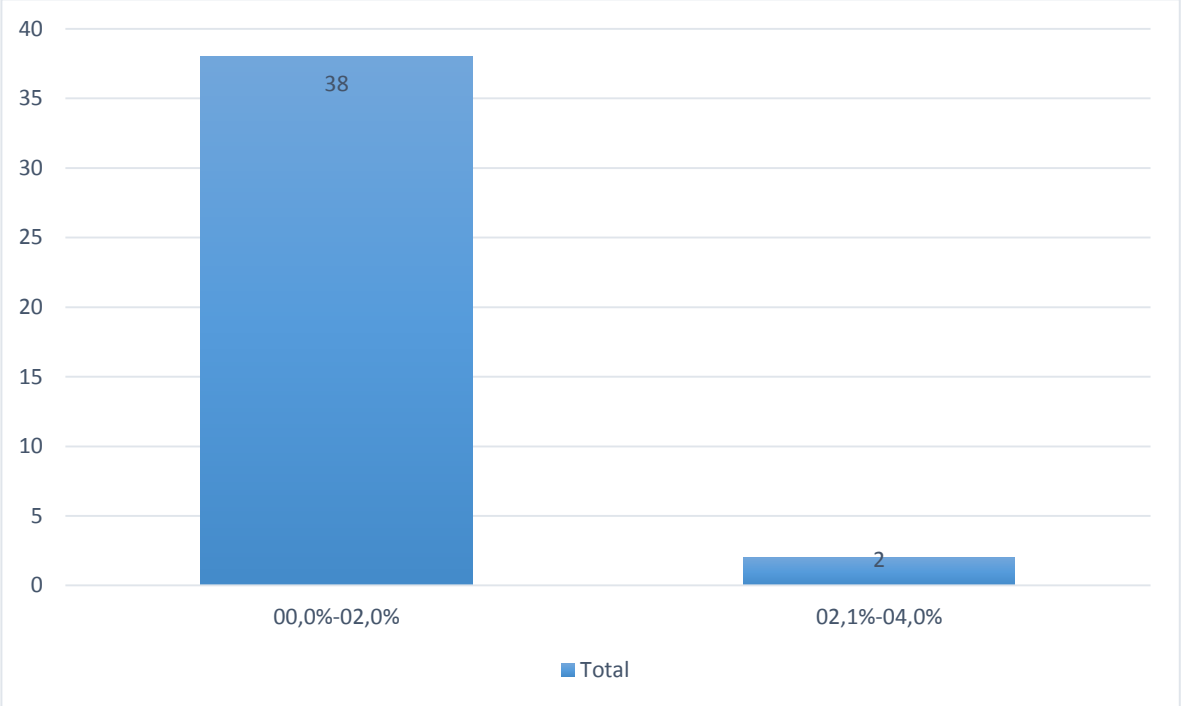


Figura 23. Porcentaje de Mejora del objetivo final con respecto a las reglas de despacho - MAKESPAN.

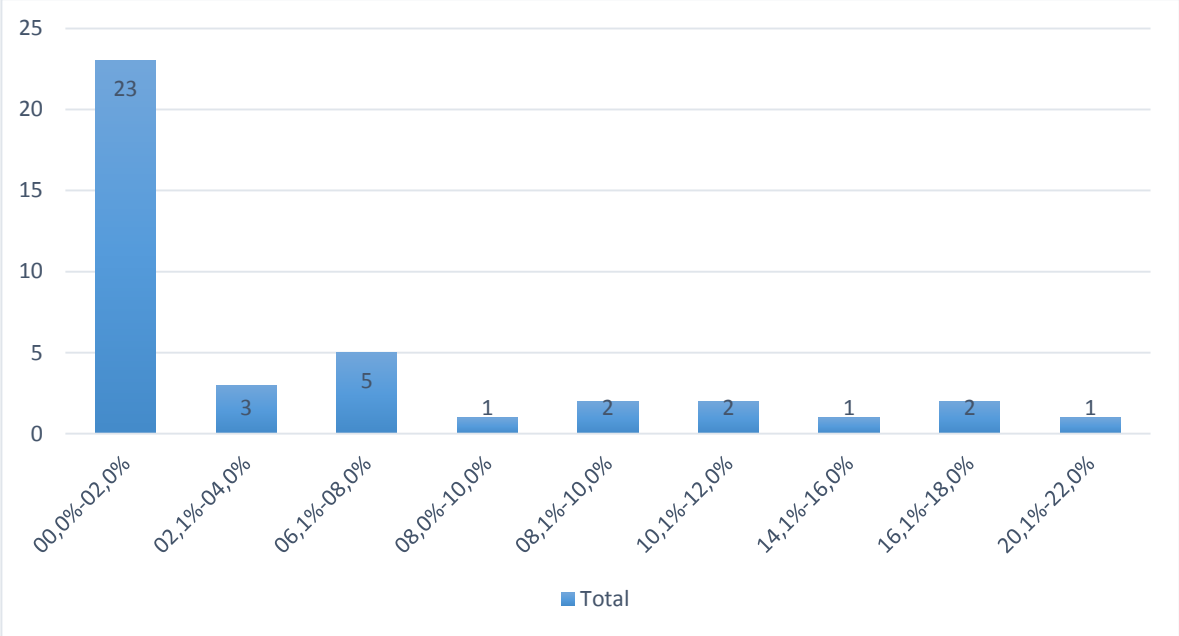


Figura 24. Porcentaje de variación del objetivo final con respecto a la programación inicial - MAKESPAN.

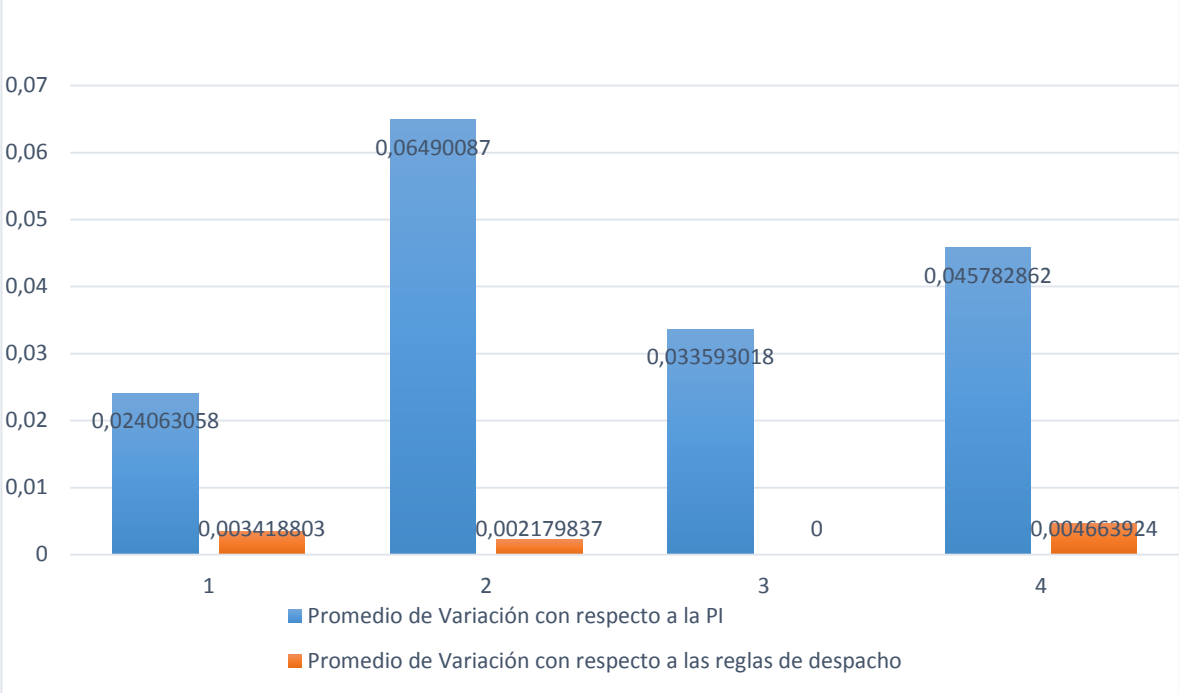


Figura 25. Promedio de variación del objetivo final con respecto a la progrmación inicial y a las reglas de despacho en los escenarios de máquinas - MAKESPAN.

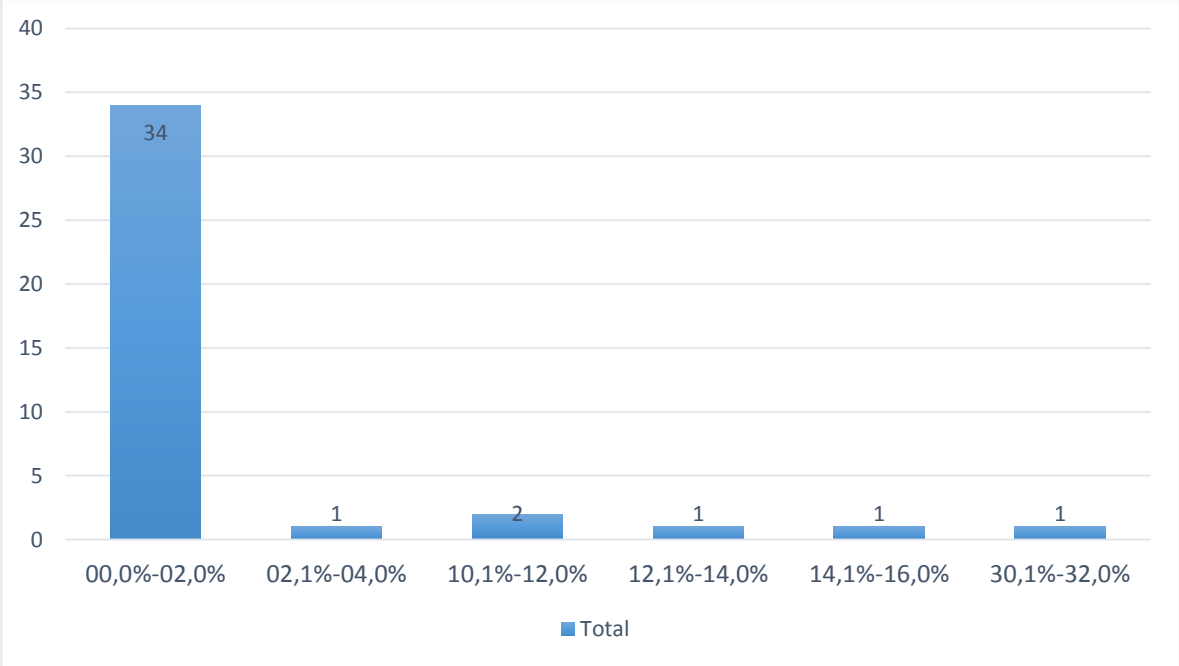


Figura 26. Porcentaje de Mejora del objetivo final con respecto a las reglas de despacho - TARDANZA TOTAL PONDERADA.

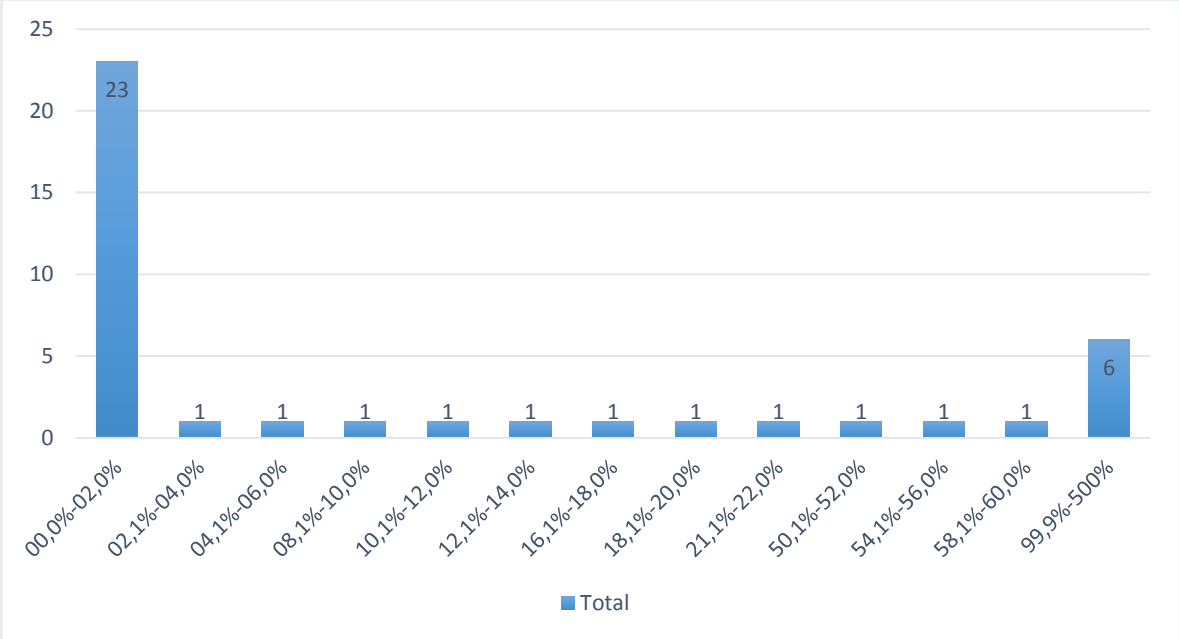


Figura 27. Porcentaje de variación del objetivo final con respecto a la programación inicial - TARDANZA TOTAL PONDERADA.

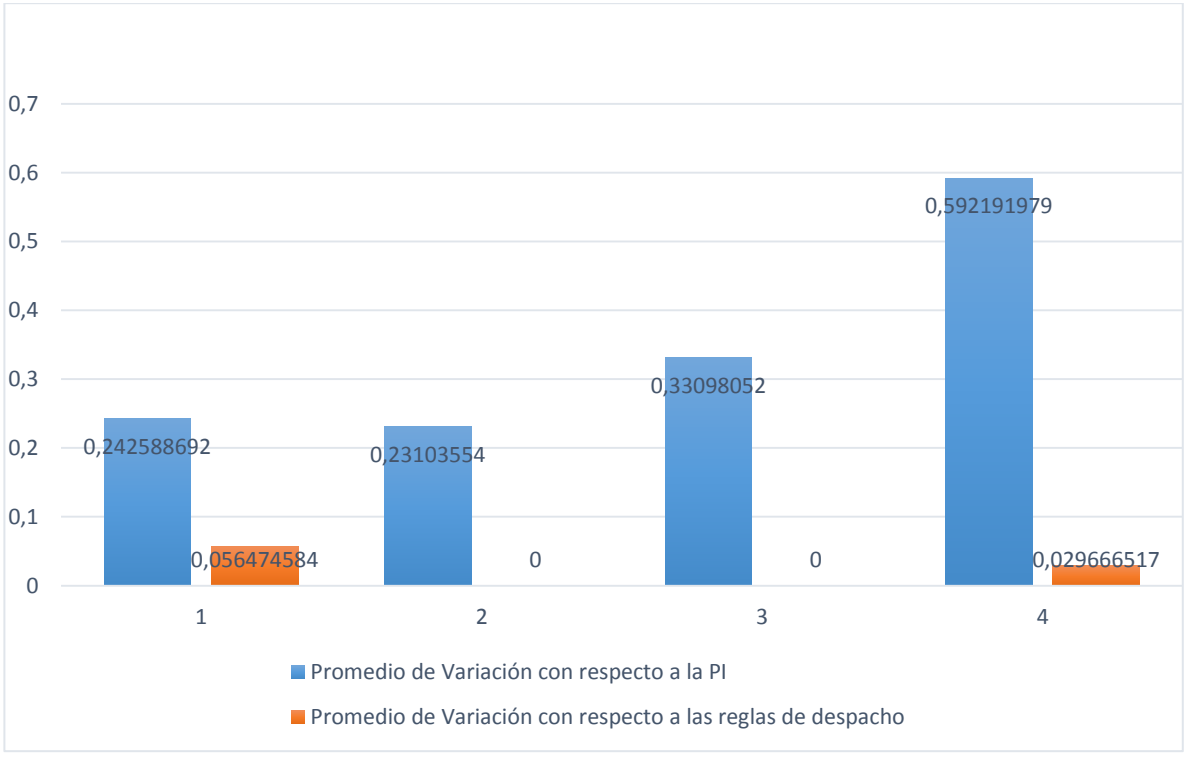


Figura 28. Promedio de variación del objetivo inicial con respecto a la programación inicial y a las reglas de despacho en los escenarios de máquinas - TARDANZA TOTAL PONDERADA.

Teniendo en cuenta los resultados podemos presentar las siguientes conclusiones del diseño de experimentos dependiente del objetivo evaluado:

Objetivo Ponderado:

- El 85% de los resultados presentan tan solo una variación entre el 0% y el 2% con respecto a los resultados obtenidos con las reglas de despacho, un resultado obtiene un porcentaje de mejora entre el 30% y el 32%.
- El 57% de los resultados presentan tan solo una variación entre el 0% y el 2% con respecto a la programación inicial.
- En los escenarios 1 y 4 en donde la falla de la máquina se presenta en la primera mitad del horizonte de la programación inicial es donde la variación de la programación final con respecto a la programación inicial es mayor.

Makespan:

- El 95% de los resultados presentan tan solo una variación entre el 0% y el 2% con respecto a los resultados obtenidos con las reglas de despacho.
- El 57% de los resultados tan solo una variación entre el 0% y el 2% con respecto a la programación inicial.

- En los escenarios 2 y 4 en donde la variación del tiempo de la falla de la máquina mayor es donde la variación de la programación final con respecto a la programación inicial es mayor.

Tardanza Total Ponderda:

- El 85% de los resultados presentan tan solo una variación entre el 0% y el 2% con respecto a los resultados obtenidos con las reglas de despacho, un resultado obtiene un porcentaje de mejora entre el 30% y el 32%.
- El 57% de los resultados presentan tan solo una variación entre el 0% y el 2% con respecto a la programación inicial.
- En los escenarios 3 y 4 con mayor tiempo de falla de la máquina es donde la variación de la programación final con respecto a la programación inicial es mayor.

El tiempo de programación total oscila entre 413 y 919 minutos, dependiendo del tamaño del problema y del instante en que se presenta la falla de la máquina.

CONCLUSIONES

Las conclusiones que se pueden generar a partir de los resultados obtenidos en el desarrollo de la presente investigación, se exponen a continuación:

- Con la utilización de la RdPT para modelar y minimizar el Makespan y la tardanza total ponderada, la integración a ella del algoritmo para la obtención de programas activos y el algoritmo genético, en la programación inicial de la producción de un FMS, se obtienen buenos resultados al ser comparados con el algoritmo AFS Petrinet y con el cuello de botella.
- La utilización de la RdPT para el modelaje de los FMS, permite: modelar de forma sencilla las fallas de las máquinas en los FMS.
- El subalgoritmo de programación inicial en PetNMA logra mejorar los resultados teniendo en cuenta los objetivos Makespan y Tardanza Total ponderada, comparándolos con las reglas de despacho.
- Encontrar un buen programa reactivo de producción depende de muchos factores descritos a continuación:
    1. Instante en que se presenta la falla: Cuando la falla se presenta en el último cuartil del horizonte de programación, es más difícil encontrar un buen programa reactivo de producción. Sin embargo los tiempos computacionales son menores que cuando la falla sucede al inicio de la producción debido a que el cromosoma es más extenso.
    2. Tiempo de reparación de la máquina que falla: Cuando el tiempo de reparación es mayor, la variación del objetivo final con respecto al objetivo inicial tiende a ser más grande, esto debido a que una máquina queda deshabilitada por más tiempo.
    3. Cantidad de máquinas en el sistema: Cuando los sistemas son más grandes, existe una mayor probabilidad de encontrar buenos programas reactivos de producción ya que las operaciones se pueden reprogramar en las otras máquinas de la estación en donde se encuentra la máquina que presenta la falla.

- La utilización del Recocido Simulado evita que el algoritmo caiga en óptimos locales. Sin embargo, los cromosomas propuestos pueden ser tan extensos que puede resultar muy costos computacionalmente.

Como continuación de esta investigación se sugiere considerar los siguientes temas:

- Implementar en ambiente real el algoritmo propuesto para la programación reactiva de la producción de un FMS, debido a la falla de máquinas.
- Considerar las situaciones donde los tiempos de alistamiento de las máquinas son independientes al tiempo de procesamiento de la operación y dependientes de la secuencia de las operaciones sobre la misma.

- Determinar la viabilidad de considerar el algoritmo híbrido entre RdPT, Simulación de Eventos Discretos y AG, para la programación reactiva de la producción en un FMS debido a la ocurrencia de otros eventos aleatorios en el piso de la planta, tales como: cancelaciones de órdenes, incorporación de nuevos trabajos con nuevos diseños, cambios de prioridades, entre otros.
- Utilizar como estrategia de decodificación del cromosoma una que explore el espacio de solución de los programas activos en un ambiente de FMS.
- Considerar nuevas estrategias de cruce y mutación para el AG, que permitan la obtención de unos mejores resultados.

REFERENCIAS BIBLIOGRÁFICAS

Acevedo, J. y Mejía, G. (2005). *Programación Reactiva y Robusta de la Producción en un Ambiente Sistema de Manufactura Flexible: Llegada de Nuevas Órdenes y Cambios en la Prioridad de las Órdenes de Trabajo*. Tesis de maestría, Universidad de los Andes, Bogotá.

Acevedo, J. y Mejía, G. (2006). *Reactive Scheduling in FMS: an Integrated Approach based on Petri Nets, Genetic Algorithms and Simulation*. En: Third International Conference on Production Research - Americas'Region 2006 (ICPR-AM06), Brasil.

Al-Hinai, H. y Elmekkawy, T.Y. (2001). An efficient hybridized genetic algorithm architecture for the flexible job shop scheduling problem. *Flexible Servicesand Manufacturing Journal*, 23(1), 64-85.

Aloulou, M. y Portmann, M. (2003). *An Efficient Proactive-Reactive Scheduling Approach to Hedge Against Shop Floor Disturbances*. G. Kendall, E.K. Burke, S. Petrovic, M. Gendreau (Eds.), Multidisciplinary scheduling: theory and applications 1st international conference, MISTA '03 Nottingham, UK, 13–15 August 2003. Selected papers, Elsevier, Amsterdam (2005), pp. 223–246.

Aized, T. (Ed.)(2010). *Advances in Petri Nets. Theory and applications*. Rijeka, Croacia: Sciyo.

Araujo, L. y Cervigón, C. (2009). *Algoritmos Evolutivos. Un enfoque práctico*. México D.F., México: Alfaomega.

Baudin, M. (1990*). Manufacturing Systems Analysis: With Application to Production Scheduling*. Englewood Cliffs, Nueva Jersey: Prentice Hall.

Bobbio, A. (1990). System Modelling with Petri Nets. *Systems Reliability Assessment*, 6, 103-143.

Buitrago, O., Britto, R. y Mejía, G. (2009). Minimización de la tardanza ponderada total en talleres de manufactura aplicando colonia de hormigas. *Colombia Ingeniería,* Grupo Editorial Gaia*14(1), 25-30.*

Castro, A. y Mejía, A. (2006). Combinación de métodos heurísticos con redes de Petri para la programación de sistemas de manufactura flexible. Departamento de Ingeniería Industrial, Universidad de los Andes, Bogotá, Colombia.

Chase, R., Jacobs, F.R. y Aquilano, N. (2005). *Administración de la producción y operaciones para una ventaja competitiva*. (10ª ed.). México D.F., México: McGraw-Hill.

Chen, J. y Chen, F.F. (2003). Adaptive scheduling in random flexible manufacturing systems subject to machine breakdowns. *International Journals of Production Research*, 41, 1927-1951.

Cho, H. (1998). Petri net models for message manipulation and event monitoring in an FMS cell. *International Journal of Production Research*, 36, 231 – 250.

Dorn, J., Kerr, R. y Thalhammer, G. (1995). Reactive scheduling: Improving the Robustness of Schedules and Restricting the Effects of Shop Floor Disturbances by Fuzzy Reasoning. *International Journal on Human–Computer Studies*, 42, 687-704.

Fleming, P., y Purshouse, R.C., (2002). Evolutionary Algorithms in Control Systems Engineering: A Survey. Control Engineering Practice, 10, 1223-1241.

Garey, M.R., Johnson, D.S., Sethi, R., (1976). The complexity of flow shop and job shop scheduling. *Mathematics of Operations Research*, 1(2), 117-129.

Garg, P. (2009). A comparison between Memetic algorithm and Genetic algorithm for the cryptanalysis of Simplified Data Encryption Standard algorithm. *International Journal of Network Security & Its Applications*, 1(1), 34-42.

Gen, M., y Cheng, R., (1997). *Genetic Algorithms and Engineering Design*. Nueva york, Estados Unidos: Jhon Wiley and Sons.

Ghoul, R., Benjelloul, A., Kechida, S. y Tebbikh, H. (2007). A scheduling algorithm based on petri nets and simulated annealing. *American Journal of Applied Sciences,* 4(5), 269-273.

González, M., Vela, C., González, I. y Varela, R., (2006). Scheduling with Memetic Algorithms over the Spaces of Semi-active and Active Schedules. *Artificial Intelligence and Soft Computing – ICAISC 2006, Lecture Notes in Computer Science*, 4029, 370-369. Springer Berlin Heidelberg.

Hatono I., Yamagata, K. y Tamura, H. (1991). Modeling and On-Line Scheduling of **f**lexible Manufacturing Systems Using Stochastic Petri Nets. *International Journal of Production Research*, 34, 313-327.

Hoos, H y Stultze T. (2005). *Stochastic Local Search. Foundations and Applications*. San Francisco, CA: Elsevier.

Kamrul, S.M., Sarker, R. y Essam, D., (2011). Genetic Algorithm for Job-Shop Scheduling with Machine Unavailability and Breakdown. *International Journal of Production Research*, 49, 4999-5015.

Kim, M. y Park, Y. (1998). A systematic procedure for setting parameters in simulated anneling algorithms*. Computers and Operations Research,* 5(3), 207-217.

Kordic, V. (Ed.)(2008). *Petri Net, Theory and Applications*. Viena, Austria: I-Tech.Education and Publishing.

Krasa, A. y Pal, S. (1990). Flexibility in Manufacturing: A survey. *The International Journal of Flexible Manufacturing System*, 2, 289-328.

Kumar, A. y Dhingra, A.K. (2012). Optimization of Schedulin Problems: A genetic algorithm survey. *Int. Journal of Applied Sciences and Engineering Research*, 1(2), 11-25.

Kumar, R., Tyagi, S. y Sharma, M. (2013). Memetic Algorithm: Hybridization of Hill Climbing with Selection Operator. *International Journal of Soft Computing and Engineering*, 3(2), 140-15.

Kumar, V., Murthy, A.N. y Chandrashekara, K. (2011). Scheduling of flexible manufacturing systems using genetic algorithm: A heuristic approach. *Journal of Industrial Engineering*, 7(14), 7-18.

Kutanoglu, E. y Sabuncuoglu, I. (2001). Routing-based reactive scheduling policies for machine failures in dynamic job shops. *International Journal of Production Research,* 39(14), 3141-3158.

Lavington, F. (1921). *The English Capital Market.* Londres:Methuen & Co.

Lawrynowicz, A., (2011). Genetic algorithms for solving scheduling problems in manufacturing systems. *Foundations of Management*, 3(2), 7-26.

Leaver, E.W. y Brown, J.J. (1946). Machines without men. *Fortune*, pp. 165, 192, 194, 196, 200, 203, 204.

Low, C. y Wu, T. H. (2001). Mathematical modelling and heuristic approaches to operation scheduling problems in an FMS environment. *International Journal of Production Research*, 39(4), 689-708.

Marimuthu, S., Ponnambalam, S.G., y Jawahar, N., (2008). Evolutionary algorithms for scheduling m-machine flow shop with lot streaming. *Journal Robotics and Computer-Integrated Manufacturing*, 24, 125-139.

Marshak, T. y Nelson, R. (1962). Flexibility, uncertainty and economic theory. Metroeconómica, 14, 42-48.

Mehta, S.V. y Uzsoy, R., (1999). Predictable scheduling of a single machine subject to breakdown. *Internationals Journal of Computer Integrated Manufacturing*, 12, 15-38.

Mejía G. y Poensgen S., (2007). *Scheduling application using petri nets: A case study: Intergráficas S.A.*. En: 19th INTERNATIONAL CONFERENCE ON PRODUCTION RESEARCH (ICPR). Libro: Sampling Plans For Acceptance In A Supply Chain Model Based On The Bayesian Theory Following Economic Criteria, 1(1), 1 – 6.

Mitchell, M., (1996). An Introduction to Genetic Algorithms. MIT Press.

Moscato, P. y Cotta, C., (2003). Una Introducción a los Algoritmos Meméticos. *Revista*

Iberoamericana de Inteligencia Artificial, 19, 131-148.

Murata, T. (1989). Petri Nets: Properties, analysis and applications. Proceedings of the IEEE, 77(4), 541-580.

Neri, F., Cotta, C. y Moscato, P. (Eds) *Handbook of Memetic Algorithms*, Studies in computational intelligence, 379, (73-94), Springer Berlin Heidelberg.

Ouelhadj D. y Petrovic S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of scheduling*, 12(4), 417-431.

Pawlewski, P. (Ed.)(2012). *Petri Nets – Manufacturing and computer science.* Rijeka, Croacia: Intech.

Pinedo, M. (2009). *Planning and Scheduling in Manufacturing and Services*. (2ª ed.). Nueva York: Springer.

Rodammer, F. A. y White Jr, K. P., (1988). A Recent Survey of Production Scheduling. *IEEE Trans. Systems, Man, and Cybernetics*, 18, 841-851.

Sharifi E., Farahini S. y Fazeli., (2008). *Production Process Optimization in Flexible Manufacturing System Using Petri Nets*. Procceding of the World Congress on Engineering and Computer Science 2008, Octubre 22-24, San Francisco, Estados Unidos.

Shih, H. y Sekiguchi, T. (1991). A timed Petri net and beam search based online FMS scheduling system with routing flexibility. *Proceedings of the IEEE international conference on robotics and automation*, 3, 2548-2553.

Simon, H.A. (1977). *The new science of management decision,* Edición revisada, Englewood Cliffs, Nueva Jersey: PrenticeHall.

Silva, M. y Valette, R. (Reconstruido 2005). Petri Nets and flexible manufacturing. *Advances in Petri Nets, Lecture Notes in Computer Science*, 424, 374-417. Springer Berlin Heidelberg.

Smith S. F. (1994). OPIS: A Methodology and Architecture for Reactive Scheduling. Zweben and Fox (eds) Intelligent Scheduling, Morgan Kaufmann, 29–66.

Suwa, H. y Sandoh, H. (2007). Capability of cumulative delay based reactive scheduling for job shops with machine breakdowns. *Computers & Industrial Engineering*, 53, 63-78.

Tanimizu, Y., Sakaguchi, T., Iwamura, K. y Sugimura, N., (2006). Evolutional reactive scheduling for agile manufacturing systems. *International Journal of Production Research*, 44(18-19), 3727–3742.

Varadharajan, T.K. y Rajendran, C. (2005). A multi-objective simulated annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of Jobs. *European Journal of Operational Research,* 167(3), 772-795.

Viera, G., Herrman, J. y Lin, E. (2003). Rescheduling manufacturing systems: a framework of strategies, policies and methods. *Journal of scheduling*, 6, 39-62.

Woo, Y., Suzuki, T. y Narikiyo, T., (2007). FMS scheduling based on timed Petri Net model and reactive graph search. *Applied Mathematical Modelling*, 31, 955-970.

Yadollahi, M. y Rahmani, A.M., (2009). Solving distributed flexible manufacturing systems scheduling problems subject to maintenance Memetic algorithms approach. *Computer and Infortmation Technology, 2009. CTI '09. Ninth IEEE International Conference on*, 1, 36-41.

Zakaria, Z., Deris, S., Othman, M. y Yatim. S. (2008). *The development of integrated planning and scheduling framework for dynamic and reactive environment of complex manufacturing problem*. Monografía sin publicar, Universiti Teknologi Malaysia, Malasia.

ANEXO A.
PROBLEMAS UTILIZADOS

A continuación se presentan las especificaciones de los problemas utilizados para evaluar el algoritmo PetNMA:

Los trabajos se identifican con el siguiente sistema de codificación:

FMSXX_J_S_M	con:	XX.	Identificación del Número del Problema (01, 02, …)
		J.	Número de Trabajos a Programar
		S.	Número de Estaciones en el FMS
		M.	Número Total de Máquinas en el FMS

Con el fin de representar las características del sistema y de cada uno de los trabajos, se utilizan las siguientes estructuras:

- Vectores de dos posiciones para caracterizar la información de cada estación
  (Identificación de la Estación, Número de Máquinas en la Estación)
- Vectores de cuatro posiciones para caracterizar la información de los trabajos
  (Id, Tiempo Disponibilidad, Fecha de Entrega, Peso)
- Vectores de dos posiciones enfrente de cada trabajo, para caracterizar la información de las operaciones. El orden en que se registran las operaciones enfrente de los trabajos, representan su secuencia por las estaciones.
  (Estación, Tiempo de Procesamiento)

FMS01_6_10_19

Seis (6) trabajos, diez (10) estaciones y un total de diecinueve (19) máquinas.
(Identificación de la Estación, Número de Máquinas en la Estación)

(W001, 2) (W002, 2) (W003, 1) (W004, 2) (W005, 2) (W006, 1) (W007, 3) (W008, 2) (W009, 1) (W0010, 3)

Trabajos: (Id, Tiempo Disponibilidad, Fecha de Entrega, Peso) Operación (Estación, Tiempo de Procesamiento)

(1, 0, 560, 1) (W008, 23) (W002, 44) (W009, 34) (W004, 48) (W001, 43) (W003, 28) (W006, 42)

(W005, 44) (W010, 41) (W007, 50)
(2, 0, 472, 3) (W001, 44) (W008, 39) (W010, 34) (W006, 36) (W005, 21) (W007, 24) (W004, 37)
(W002, 35) (W009, 35) (W003, 20)
(3, 0, 508, 1) (W003, 33) (W006, 35) (W002, 23) (W010, 44) (W009, 31) (W004, 41) (W007, 42)
(W008, 49) (W001, 49) (W005, 26)
(4, 0, 439, 2) (W009, 29) (W008, 39) (W003, 20) (W007, 32) (W005, 25) (W004, 27) (W010, 42)
(W006, 45) (W001, 28) (W002, 26)
(5, 0, 392, 1) (W006, 23) (W001, 47) (W008, 31) (W005, 20) (W010, 30) (W009, 20) (W003, 25)
(W002, 31) (W007, 46) (W004, 21)
(6, 0, 535, 1) (W005, 46) (W002, 41) (W006, 48) (W008, 24) (W001, 44) (W003, 48) (W007, 24)
(W004, 26) (W009, 40) (W010, 49)

FMS02_9_8_20

Nueve (9) trabajos, ocho (8) estaciones y un total de veinte (20) máquinas.
(Identificación de la Estación, Número de Máquinas en la Estación)

(W001, 2) (W002, 3) (W003, 2) (W004, 3) (W005, 2) (W006, 3) (W007, 3) (W008, 2)

Trabajos: (Id, Tiempo Disponibilidad, Fecha de Entrega, Peso) Operación (Estación, Tiempo de Procesamiento)

(1, 0, 419, 3) (W001, 40) (W006, 42) (W005, 31) (W003, 24) (W008, 42) (W007, 43) (W004, 38)
(W002, 50)
(2, 0, 341, 1) (W003, 27) (W001, 31) (W008, 23) (W007, 34) (W006, 28) (W005, 29) (W002, 40)
(W004, 23)
(3, 0, 459, 1) (W003, 42) (W001, 49) (W002, 42) (W008, 40) (W007, 31) (W004, 32) (W005, 28)
(W006, 50)
(4, 0, 345, 1) (W002, 25) (W004, 32) (W006, 21) (W005, 21) (W007, 48) (W008, 33) (W001, 22)
(W003, 48)
(5, 0, 361, 3) (W004, 31) (W002, 40) (W005, 38) (W001, 38) (W006, 28) (W003, 29) (W007,

33)
(W008, 34)
(6, 0, 338, 1) (W003, 29) (W005, 37) (W006, 20) (W008, 36) (W004, 29) (W007, 45) (W001, 24)
(W002, 34)
(7, 0, 398, 3) (W004, 50) (W002, 42) (W003, 35) (W007, 29) (W006, 21) (W008, 25) (W005, 35)
(W001, 28)
(8, 0, 403, 3) (W006, 35) (W004, 32) (W002, 34) (W003, 40) (W001, 46) (W005, 40) (W007, 24)
(W008, 21)
(9, 0, 393, 2) (W007, 38) (W008, 40) (W003, 35) (W004, 31) (W001, 32) (W005, 24) (W006, 25)
(W002, 44)

FMS03_10_5_10

Diez (10) trabajos, cuatro (4) estaciones y un total de diez (10) máquinas.
(Identificación de la Estación, Número de Máquinas en la Estación)

(W001, 2) (W002, 1) (W003, 2) (W004, 3) (W004, 2)

Trabajos: (Id, Tiempo Disponibilidad, Fecha de Entrega, Peso) Operación (Estación, Tiempo de Procesamiento)

(1, 0, 357, 1) (W002, 21) (W001, 53) (W005, 95) (W004, 55) (W003, 34)
(2, 0, 274, 2) (W001, 21) (W004, 52) (W005, 16) (W003, 26) (W002, 71)
(3, 0, 302, 2) (W004, 39) (W005, 98) (W002, 42) (W003, 31) (W001, 12)
(4, 0, 517, 2) (W002, 77) (W001, 55) (W005, 79) (W003, 66) (W004, 77)
(5, 0, 354, 2) (W001, 83) (W004, 34) (W003, 64) (W002, 19) (W005, 37)
(6, 0, 354, 1) (W002, 54) (W003, 43) (W005, 79) (W001, 92) (W004, 62)
(7, 0, 242, 2) (W004, 69) (W005, 77) (W002, 87) (W003, 87) (W001, 93)
(8, 0, 322, 1) (W003, 38) (W001, 60) (W002, 41) (W004, 24) (W005, 83)
(9, 0, 524, 3) (W003, 17) (W002, 49) (W005, 25) (W001, 44) (W003, 98)
(10, 0, 347, 3) (W005, 77) (W004, 79) (W003, 43) (W002, 75) (W001, 96)

FMS04_10_5_12

Diez (10) trabajos, cinco (5) estaciones y un total de doce (12) máquinas.
(Identificación de la Estación, Número de Máquinas en la Estación)

(W001, 3) (W002, 2) (W003, 2) (W004, 3) (W005, 2)

Trabajos: (Id, Tiempo Disponibilidad, Fecha de Entrega, Peso) Operación (Estación, Tiempo de Procesamiento)

(1, 0, 342, 1) (W001, 20) (W004, 87) (W002, 31) (W005, 76) (W003, 17)
(2, 0, 240, 2) (W005, 25) (W003, 32) (W001, 24) (W002, 18) (W004, 81)
(3, 0, 409, 3) (W002, 72) (W003, 23) (W005, 28) (W001, 58) (W004, 99)
(4, 0, 568, 3) (W003, 86) (W002, 76) (W005, 97) (W001, 45) (W004, 90)
(5, 0, 240, 2) (W005, 27) (W001, 42) (W004, 48) (W003, 17) (W002, 46)
(6, 0, 345, 1) (W002, 67) (W001, 98) (W005, 48) (W004, 27) (W003, 62)
(7, 0, 270, 1) (W005, 28) (W002, 12) (W004, 19) (W001, 80) (W003, 50)
(8, 0, 420, 2) (W002, 63) (W001, 94) (W003, 98) (W004, 50) (W005, 80)
(9, 0, 540, 3) (W005, 14) (W001, 75) (W003, 50) (W002, 41) (W004, 55)
(10, 0, 256, 2) (W005, 72) (W003, 18) (W002, 37) (W004, 79) (W001, 61)

FMS05_10_5_12

Diez (10) trabajos, cinco (5) estaciones y un total de doce (12) máquinas.
(Identificación de la Estación, Número de Máquinas en la Estación)

(W001, 2) (W002, 3) (W003, 3) (W004, 2) (W005, 2)

Trabajos: (Id, Tiempo Disponibilidad, Fecha de Entrega, Peso) Operación (Estación, Tiempo de Procesamiento)

(1, 0, 384, 2) (W002, 23) (W003, 45) (W001, 82) (W005, 84) (W004, 38)
(2, 0, 229, 3) (W003, 21) (W002, 29) (W001, 18) (W005, 41) (W004, 50)
(3, 0, 314, 2) (W003, 38) (W004, 54) (W0005, 16) (W001, 52) (W002, 52)
(4, 0, 421, 3) (W005, 37) (W001, 54) (W003, 74) (W002, 62) (W004, 57)
(5, 0, 422, 3) (W005, 57) (W001, 81) (W002, 61) (W004, 68) (W003, 30)
(6, 0, 461, 3) (W005, 81) (W001, 79) (W002, 89) (W003, 89) (W004, 11)
(7, 0, 299, 1) (W004, 33) (W003, 20) (W001, 91) (W005, 20) (W002, 66)
(8, 0, 285, 1) (W005, 24) (W002, 84) (W001, 32) (W003, 55) (W004, 8)
(9, 0, 289, 1) (W005, 56) (W001, 7) (W004, 54) (W003, 64) (W002, 39)
(10, 0, 214, 2) (W005, 40) (W002, 83) (W001, 19) (W003, 8) (W004, 7)

FMS06_10_7_18

Diez (10) trabajos, siete (7) estaciones y un total de dieciocho (18) máquinas.
(Identificación de la Estación, Número de Máquinas en la Estación)

(W001, 2) (W002, 2) (W003, 3) (W004, 2) (W005, 3) (W006, 3) (W007, 3)

Trabajos: (Id, Tiempo Disponibilidad, Fecha de Entrega, Peso) Operación (Estación, Tiempo de Procesamiento)

(1, 0, 376, 1) (W007, 25) (W006, 44) (W005, 45) (W004, 42) (W001, 27) (W002, 49) (W003, 40)
(2, 0, 385, 3) (W005, 40) (W007, 28) (W004, 46) (W006, 46) (W003, 41) (W002, 39) (W001, 47)
(3, 0, 324, 3) (W003, 32) (W001, 49) (W002, 39) (W004, 43) (W005, 20) (W006, 35) (W007, 29)
(4, 0, 353, 3) (W004, 26) (W005, 25) (W003, 35) (W001, 46) (W002, 50) (W007, 35) (W006, 42)
(5, 0, 352, 2) (W004, 26) (W005, 29) (W006, 27) (W001, 43) (W002, 44) (W007, 37) (W003, 38)
(6, 0, 406, 1) (W003, 48) (W007, 36) (W004, 21) (W001, 42) (W005, 36) (W002, 46) (W006, 43)
(7, 0, 317, 2) (W005, 38) (W006, 21) (W004, 20) (W002, 40) (W003, 36) (W001, 36) (W007, 26)
(8, 0, 369, 1) (W005, 39) (W007, 49) (W004, 37) (W006, 36) (W002, 42) (W001, 42) (W003, 26)
(9, 0, 364, 3) (W004, 32) (W003, 39) (W007, 22) (W001, 46) (W005, 42) (W006, 40) (W002, 50)
(10, 0, 357, 2) (W005, 46) (W007, 42) (W003, 50) (W002, 20) (W006, 20) (W004, 39) (W001, 41)

FMS07_10_10_20

Diez (10) trabajos, diez (10) estaciones y un total de veinte (20) máquinas.
(Identificación de la Estación, Número de Máquinas en la Estación)

(W001, 2) (W002, 1) (W003, 2) (W004, 2) (W005, 3) (W006, 2) (W007, 3) (W008, 2) (W009, 2) (W010, 1)

Trabajos: (Id, Tiempo Disponibilidad, Fecha de Entrega, Peso) Operación (Estación, Tiempo de

Procesamiento)

(1, 0, 738, 3) (W001, 72) (W002, 64) (W003, 55) (W004, 31) (W005, 53) (W006, 95) (W007, 11)
(W008, 52) (W009, 6) (W010, 84)
(2, 0, 901, 3) (W001, 61) (W004, 27) (W005, 88) (W003, 78) (W002, 49) (W006, 83) (W009, 91)
(W007, 74) (W008, 29) (W010, 87)
(3, 0, 685, 2) (W001, 86) (W004, 32) (W002, 35) (W003, 37) (W006, 18) (W005, 48) (W007, 91)
(W008, 52) (W010, 60) (W009, 30)
(4, 0, 764, 1) (W001, 8) (W002, 82) (W005, 27) (W004, 99) (W007, 74) (W006, 9) (W003, 33)
(W0010, 20) (W008, 59) (W009, 98)
(5, 0, 1355, 2) (W002, 50) (W001, 94) (W006, 43) (W004, 62) (W005, 55) (W008, 487) (W003, 5)
(W009, 36) (W010, 47) (W007, 36)
(6, 0, 566, 1) (W001, 53) (W007, 30) (W003, 7) (W004, 12) (W002, 68) (W009, 87) (W005, 28)
(W010, 70) (W008, 45) (W006, 7)
(7, 0, 723, 1) (W003, 29) (W004, 96) (W001, 99) (W002, 14) (W005, 34) (W008, 14) (W006, 7)
(W007, 76) (W009, 57) (W010, 76)
(8, 0, 967, 2) (W003, 90) (W001, 19) (W004, 87) (W005, 51) (W002, 84) (W006, 45) (W010, 84)
(W007, 58) (W008, 81) (W009, 96)
(9, 0, 889, 3) (W003, 97) (W002, 99) (W005, 93) (W001, 38) (W008, 13) (W006, 96) (W004, 40)
(W010, 64) (W007, 32) (W009, 45)
(10, 0, 736, 3) (W003, 44) (W001, 60) (W009, 29) (W004, 5) (W007, 74) (W002, 85) (W005, 34)
(W008, 95) (W010, 51) (W006, 47)

FMS08_15_5_12

Quince (15) trabajos, cinco (5) estaciones y un total de doce (12) máquinas.
(Identificación de la Estación, Número de Máquinas en la Estación)

(W001, 3) (W002, 2) (W003, 2) (W004, 2) (W005, 3)

Trabajos: (Id, Tiempo Disponibilidad, Fecha de Entrega, Peso) Operación (Estación, Tiempo de

Procesamiento)

(1, 0, 354, 2) (W002, 21) (W003, 34) (W005, 95) (W001, 53) (W004, 55)
(2, 0, 278, 2) (W004, 52) (W005, 16) (W002, 71) (W003, 26) (W001, 21)
(3, 0, 294, 3) (W003, 31) (W001, 12) (W002, 42) (W004, 39) (W005, 98)
(4, 0, 503, 3) (W004, 77) (W002, 77) (W005, 79) (W001, 55) (W003, 66)
(5, 0, 313, 1) (W005, 37) (W004, 34) (W003, 64) (W002, 19) (W001, 83)
(6, 0, 453, 1) (W003, 43) (W002, 54) (W001, 92) (W004, 62) (W005, 79)
(7, 0, 616, 3) (W001, 93) (W004, 69) (W002, 87) (W005, 77) (W003, 87)
(8, 0, 355, 2) (W001, 60) (W002, 41) (W003, 38) (W005, 83) (W004, 24)
(9, 0, 320, 1) (W003, 98) (W004, 17) (W005, 25) (W001, 44) (W002, 49)
(10, 0, 500, 2) (W001, 96) (W005, 77) (W004, 79) (W002, 75) (W003, 43)
(11, 0, 338, 2) (W005, 28) (W003, 35) (W001, 95) (W004, 76) (W002, 7)
(12, 0, 311, 1) (W001, 61) (W005, 10) (W003, 95) (W002, 9) (W004, 35)
(13, 0, 383, 3) (W005, 59) (W004, 16) (W002, 91) (W003, 59) (W001, 46)
(14, 0, 298, 1) (W005, 43) (W002, 52) (W001, 28) (W003, 27) (W004, 50)
(15, 0, 288, 2) (W001, 87) (W002, 45) (W003, 39) (W005, 9) (W004, 41)

FMS09_15_6_17

Quince (15) trabajos, seis (6) estaciones y un total de diecisiete (17) máquinas.
(Identificación de la Estación, Número de Máquinas en la Estación)

(W001, 3) (W002, 3) (W003, 2) (W004, 2) (W005, 3) (W006, 4)

Trabajos: (Id, Tiempo Disponibilidad, Fecha de Entrega, Peso) Operación (Estación, Tiempo de Procesamiento)

(1, 0, 203, 3) (W001, 24) (W004, 20) (W002, 13) (W003, 27) (W006, 38) (W005, 34)
(2, 0, 236, 3) (W006, 32) (W004, 18) (W002, 37) (W001, 32) (W005, 23) (W003, 39)
(3, 0, 151, 3) (W003, 15) (W004, 11) (W006, 10) (W002, 29) (W005, 13) (W001, 38)
(4, 0, 320, 3) (W005, 27) (W006, 18) (W003, 47) (W002, 28) (W001, 50) (W004, 43)
(5, 0, 254, 2) (W004, 39) (W005, 26) (W003, 14) (W002, 14) (W001, 31) (W006, 45)
(6, 0, 266, 1) (W003, 32) (W001, 50) (W005, 23) (W006, 28) (W004, 19) (W002, 25)
(7, 0, 255, 3) (W002, 49) (W003, 38) (W004, 23) (W005, 14) (W001, 32) (W006, 14)
(8, 0, 273, 2) (W006, 29) (W001, 15) (W003, 48) (W002, 50) (W005, 36) (W004, 32)
(9, 0, 333, 3) (W006, 44) (W005, 28) (W004, 45) (W001, 26) (W002, 47) (W003, 32)
(10, 0, 288, 3) (W005, 23) (W004, 50) (W006, 36) (W003, 25) (W002, 30) (W001, 28)
(11, 0, 193, 1) (W001, 40) (W002, 24) (W004, 18) (W005, 12) (W006, 17) (W003, 37)
(12, 0, 199, 3) (W006, 21) (W002, 31) (W005, 12) (W003, 30) (W001, 27) (W004, 32)

(13, 0, 296, 1) (W005, 35) (W003, 10) (W006, 50) (W001, 49) (W002, 35) (W004, 48)
(14, 0, 302, 3) (W006, 45) (W004, 35) (W001, 30) (W002, 29) (W003, 31) (W005, 31)
(15, 0, 195, 3) (W001, 22) (W003, 25) (W002, 13) (W004, 38) (W006, 17) (W005, 15)

FMS10_20_6_13

Veinte (20) trabajos, seis (6) estaciones y un total de trece (13) máquinas.
(Identificación de la Estación, Número de Máquinas en la Estación)

(W001, 2) (W002, 1) (W003, 3) (W004, 3) (W005, 3) (W006, 1)

Trabajos: (Id, Tiempo Disponibilidad, Fecha de Entrega, Peso) Operación (Estación, Tiempo de Procesamiento)

(1, 0, 246, 2) (W005, 40) (W003, 33) (W006, 26) (W001, 22) (W002, 26) (W004, 31)
(2, 0, 354, 1) (W004, 50) (W002, 34) (W005, 32) (W003, 46) (W001, 38) (W006, 39)
(3, 0, 365, 2) (W005, 31) (W004, 24) (W003, 47) (W001, 49) (W006, 47) (W002, 50)
(4, 0, 317, 1) (W001, 38) (W003, 43) (W004, 37) (W005, 36) (W006, 38) (W002, 26)
(5, 0, 284, 1) (W005, 21) (W002, 50) (W001, 29) (W003, 35) (W006, 47) (W004, 22)
(6, 0, 255, 2) (W003, 31) (W001, 27) (W005, 44) (W002, 21) (W004, 39) (W006, 29)
(7, 0, 281, 3) (W004, 33) (W005, 22) (W001, 28) (W002, 45) (W006, 39) (W003, 39)
(8, 0, 287, 2) (W001, 45) (W002, 21) (W005, 46) (W006, 33) (W004, 30) (W003, 36)
(9, 0, 230, 3) (W001, 36) (W003, 33) (W005, 24) (W002, 21) (W006, 35) (W004, 22)
(10, 0, 291, 1) (W004, 40) (W003, 30) (W002, 32) (W006, 36) (W005, 24) (W001, 34)
(11, 0, 357, 2) (W006, 44) (W002, 42) (W001, 24) (W003, 36) (W005, 46) (W004, 49)
(12, 0, 249, 3) (W004, 23) (W005, 30) (W001, 37) (W003, 30) (W006, 21) (W002, 29)
(13, 0, 297, 1) (W005, 48) (W003, 47) (W004, 31) (W001, 26) (W006, 28) (W002, 43)
(14, 0, 284, 3) (W006, 36) (W003, 33) (W004, 37) (W001, 23) (W005, 46) (W002, 35)
(15, 0, 271, 3) (W003, 27) (W002, 40) (W004, 30) (W006, 42) (W001, 24) (W005, 30)
(16, 0, 348, 3) (W001, 44) (W005, 36) (W006, 42) (W004, 40) (W002, 48) (W003, 35)
(17, 0, 297, 2) (W003, 27) (W004, 32) (W002, 43) (W005, 37) (W001, 50) (W006, 34)
(18, 0, 249, 3) (W003, 20) (W002, 25) (W001, 29) (W005, 26) (W004, 30) (W006, 44)
(19, 0, 270, 3) (W001, 31) (W002, 23) (W004, 22) (W005, 38) (W006, 50) (W003, 40)
(20, 0, 251, 2) (W005, 26) (W004, 22) (W001, 20) (W006, 43) (W002, 29) (W003, 48)

ANEXO B.
RESULTADOS DE LAS CORRIDAS DE LA PROGRAMACIÓN INICIAL

A continuación se muestran los resultados de las corridas de prueba del algoritmo genético del Subalgoritmo de Programación Inicial de PetNMA, en donde:

Alfa	Porcentaje del makespan del objetivo ponderado
Beta	Porcentaje de la tardanza total ponderada del objetivo ponderado
Maxgen	Número de generaciones
P	Tamaño de la población
Tc	Probabilidad de cruce (%)
Tm	Probabilidad de mutación (%)
Im	Intensidad de mutación
Te1	Porcentaje de elitismo
Gen	Generación en la que es encontrada la mejor solución

Tabla 35

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 100% y beta = 0% (Parte 1 de 2)

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
0	100	200	100	60	10	1	0	0	369	0	0
0	100	200	100	60	10	1	25	0	369	0	0
0	100	200	100	60	10	1	50	0	369	0	0
0	100	200	100	60	10	2	0	0	369	0	0
0	100	200	100	60	10	2	25	0	369	0	0
0	100	200	100	60	10	2	50	0	369	0	0
0	100	200	100	60	10	4	0	0	369	0	0
0	100	200	100	60	10	4	25	0	369	0	0
0	100	200	100	60	10	4	50	0	369	0	0
0	100	200	100	60	50	1	0	0	369	0	0
0	100	200	100	60	50	1	25	0	369	0	0
0	100	200	100	60	50	1	50	0	369	0	0
0	100	200	100	60	50	2	0	0	369	0	0
0	100	200	100	60	50	2	25	0	369	0	0
0	100	200	100	60	50	2	50	0	369	0	0
0	100	200	100	60	50	4	0	0	369	0	0
0	100	200	100	60	50	4	25	0	369	0	0

Tabla 36

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 100% y beta = 0% (Parte 2 de 2)

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
0	100	200	100	60	50	4	50	0	369	0	0
0	100	200	100	90	10	1	0	0	369	0	0
0	100	200	100	90	10	1	25	0	369	0	0
0	100	200	100	90	10	1	50	0	369	0	0
0	100	200	100	90	10	2	0	0	369	0	0
0	100	200	100	90	10	2	25	0	369	0	0
0	100	200	100	90	10	2	50	0	369	0	0
0	100	200	100	90	10	4	0	0	369	0	0
0	100	200	100	90	10	4	25	0	369	0	0
0	100	200	100	90	10	4	50	0	369	0	0
0	100	200	100	90	50	1	0	0	369	0	0
0	100	200	100	90	50	1	25	0	369	0	0
0	100	200	100	90	50	1	50	0	369	0	0
0	100	200	100	90	50	2	0	0	369	0	0
0	100	200	100	90	50	2	25	0	369	0	0
0	100	200	100	90	50	2	50	0	369	0	0
0	100	200	100	90	50	4	0	0	369	0	0
0	100	200	100	90	50	4	25	0	369	0	0
0	100	200	100	90	50	4	50	0	369	0	0

Tabla 37

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 0% y beta = 100%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
100	0	200	100	60	10	1	0	324	324	0	0
100	0	200	100	60	10	1	25	324	324	0	0
100	0	200	100	60	10	1	50	324	324	0	0
100	0	200	100	60	10	2	0	324	324	0	0
100	0	200	100	60	10	2	25	324	324	0	0
100	0	200	100	60	10	2	50	324	324	0	0
100	0	200	100	60	10	4	0	324	324	0	0
100	0	200	100	60	10	4	25	324	324	0	0
100	0	200	100	60	10	4	50	324	324	0	0
100	0	200	100	60	50	1	0	324	324	0	0
100	0	200	100	60	50	1	25	324	324	0	0
100	0	200	100	60	50	1	50	324	324	0	0
100	0	200	100	60	50	2	0	324	324	0	0
100	0	200	100	60	50	2	25	324	324	0	0
100	0	200	100	60	50	2	50	324	324	0	0
100	0	200	100	60	50	4	0	324	324	0	0
100	0	200	100	60	50	4	25	324	324	0	0
100	0	200	100	60	50	4	50	324	324	0	0
100	0	200	100	90	10	1	0	324	324	0	0
100	0	200	100	90	10	1	25	324	324	0	0
100	0	200	100	90	10	1	50	324	324	0	0
100	0	200	100	90	10	2	0	324	324	0	0
100	0	200	100	90	10	2	25	324	324	0	0
100	0	200	100	90	10	2	50	324	324	0	0
100	0	200	100	90	10	4	0	324	324	0	0
100	0	200	100	90	10	4	25	324	324	0	0
100	0	200	100	90	10	4	50	324	324	0	0
100	0	200	100	90	50	1	0	324	324	0	0
100	0	200	100	90	50	1	25	324	324	0	0
100	0	200	100	90	50	1	50	324	324	0	0
100	0	200	100	90	50	2	0	324	324	0	0
100	0	200	100	90	50	2	25	324	324	0	0
100	0	200	100	90	50	2	50	324	324	0	0
100	0	200	100	90	50	4	0	324	324	0	0
100	0	200	100	90	50	4	25	324	324	0	0
100	0	200	100	90	50	4	50	324	324	0	0

Tabla 38

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 10% y beta = 90%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
10	90	200	100	60	10	1	0	32,4	324	0	0
10	90	200	100	60	10	1	25	32,4	324	0	0
10	90	200	100	60	10	1	50	32,4	324	0	0
10	90	200	100	60	10	2	0	32,4	324	0	0
10	90	200	100	60	10	2	25	32,4	324	0	0
10	90	200	100	60	10	2	50	32,4	324	0	0
10	90	200	100	60	10	4	0	32,4	324	0	0
10	90	200	100	60	10	4	25	32,4	324	0	0
10	90	200	100	60	10	4	50	32,4	324	0	0
10	90	200	100	60	50	1	0	32,4	324	0	0
10	90	200	100	60	50	1	25	32,4	324	0	0
10	90	200	100	60	50	1	50	32,4	324	0	0
10	90	200	100	60	50	2	0	32,4	324	0	0
10	90	200	100	60	50	2	25	32,4	324	0	0
10	90	200	100	60	50	2	50	32,4	324	0	0
10	90	200	100	60	50	4	0	32,4	324	0	0
10	90	200	100	60	50	4	25	32,4	324	0	0
10	90	200	100	60	50	4	50	32,4	324	0	0
10	90	200	100	90	10	1	0	32,4	324	0	0
10	90	200	100	90	10	1	25	32,4	324	0	0
10	90	200	100	90	10	1	50	32,4	324	0	0
10	90	200	100	90	10	2	0	32,4	324	0	0
10	90	200	100	90	10	2	25	32,4	324	0	0
10	90	200	100	90	10	2	50	32,4	324	0	0
10	90	200	100	90	10	4	0	32,4	324	0	0
10	90	200	100	90	10	4	25	32,4	324	0	0
10	90	200	100	90	10	4	50	32,4	324	0	0
10	90	200	100	90	50	1	0	32,4	324	0	0
10	90	200	100	90	50	1	25	32,4	324	0	0
10	90	200	100	90	50	1	50	32,4	324	0	0
10	90	200	100	90	50	2	0	32,4	324	0	0
10	90	200	100	90	50	2	25	32,4	324	0	0
10	90	200	100	90	50	2	50	32,4	324	0	0
10	90	200	100	90	50	4	0	32,4	324	0	0
10	90	200	100	90	50	4	25	32,4	324	0	0
10	90	200	100	90	50	4	50	32,4	324	0	0

Tabla 39

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 20% y beta =80%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
20	80	200	100	60	10	1	0	64,8	324	0	0
20	80	200	100	60	10	1	25	64,8	324	0	0
20	80	200	100	60	10	1	50	64,8	324	0	0
20	80	200	100	60	10	2	0	64,8	324	0	0
20	80	200	100	60	10	2	25	64,8	324	0	0
20	80	200	100	60	10	2	50	64,8	324	0	0
20	80	200	100	60	10	4	0	64,8	324	0	0
20	80	200	100	60	10	4	25	64,8	324	0	0
20	80	200	100	60	10	4	50	64,8	324	0	0
20	80	200	100	60	50	1	0	64,8	324	0	0
20	80	200	100	60	50	1	25	64,8	324	0	0
20	80	200	100	60	50	1	50	64,8	324	0	0
20	80	200	100	60	50	2	0	64,8	324	0	0
20	80	200	100	60	50	2	25	64,8	324	0	0
20	80	200	100	60	50	2	50	64,8	324	0	0
20	80	200	100	60	50	4	0	64,8	324	0	0
20	80	200	100	60	50	4	25	64,8	324	0	0
20	80	200	100	60	50	4	50	64,8	324	0	0
20	80	200	100	90	10	1	0	64,8	324	0	0
20	80	200	100	90	10	1	25	64,8	324	0	0
20	80	200	100	90	10	1	50	64,8	324	0	0
20	80	200	100	90	10	2	0	64,8	324	0	0
20	80	200	100	90	10	2	25	64,8	324	0	0
20	80	200	100	90	10	2	50	64,8	324	0	0
20	80	200	100	90	10	4	0	64,8	324	0	0
20	80	200	100	90	10	4	25	64,8	324	0	0
20	80	200	100	90	10	4	50	64,8	324	0	0
20	80	200	100	90	50	1	0	64,8	324	0	0
20	80	200	100	90	50	1	25	64,8	324	0	0
20	80	200	100	90	50	1	50	64,8	324	0	0
20	80	200	100	90	50	2	0	64,8	324	0	0
20	80	200	100	90	50	2	25	64,8	324	0	0
20	80	200	100	90	50	2	50	64,8	324	0	0
20	80	200	100	90	50	4	0	64,8	324	0	0
20	80	200	100	90	50	4	25	64,8	324	0	0
20	80	200	100	90	50	4	50	64,8	324	0	0

Tabla 40

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 30% y beta = 70%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
30	70	200	100	60	10	1	0	97,2	324	0	0
30	70	200	100	60	10	1	25	97,2	324	0	0
30	70	200	100	60	10	1	50	97,2	324	0	0
30	70	200	100	60	10	2	0	97,2	324	0	0
30	70	200	100	60	10	2	25	97,2	324	0	0
30	70	200	100	60	10	2	50	97,2	324	0	0
30	70	200	100	60	10	4	0	97,2	324	0	0
30	70	200	100	60	10	4	25	97,2	324	0	0
30	70	200	100	60	10	4	50	97,2	324	0	0
30	70	200	100	60	50	1	0	97,2	324	0	0
30	70	200	100	60	50	1	25	97,2	324	0	0
30	70	200	100	60	50	1	50	97,2	324	0	0
30	70	200	100	60	50	2	0	97,2	324	0	0
30	70	200	100	60	50	2	25	97,2	324	0	0
30	70	200	100	60	50	2	50	97,2	324	0	0
30	70	200	100	60	50	4	0	97,2	324	0	0
30	70	200	100	60	50	4	25	97,2	324	0	0
30	70	200	100	60	50	4	50	97,2	324	0	0
30	70	200	100	90	10	1	0	97,2	324	0	0
30	70	200	100	90	10	1	25	97,2	324	0	0
30	70	200	100	90	10	1	50	97,2	324	0	0
30	70	200	100	90	10	2	0	97,2	324	0	0
30	70	200	100	90	10	2	25	97,2	324	0	0
30	70	200	100	90	10	2	50	97,2	324	0	0
30	70	200	100	90	10	4	0	97,2	324	0	0
30	70	200	100	90	10	4	25	97,2	324	0	0
30	70	200	100	90	10	4	50	97,2	324	0	0
30	70	200	100	90	50	1	0	97,2	324	0	0
30	70	200	100	90	50	1	25	97,2	324	0	0
30	70	200	100	90	50	1	50	97,2	324	0	0
30	70	200	100	90	50	2	0	97,2	324	0	0
30	70	200	100	90	50	2	25	97,2	324	0	0
30	70	200	100	90	50	2	50	97,2	324	0	0
30	70	200	100	90	50	4	0	97,2	324	0	0
30	70	200	100	90	50	4	25	97,2	324	0	0
30	70	200	100	90	50	4	50	97,2	324	0	0

Tabla 41

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 40% y beta = 60%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
40	60	200	100	60	10	1	0	129,6	324	0	0
40	60	200	100	60	10	1	25	129,6	324	0	0
40	60	200	100	60	10	1	50	129,6	324	0	0
40	60	200	100	60	10	2	0	129,6	324	0	0
40	60	200	100	60	10	2	25	129,6	324	0	0
40	60	200	100	60	10	2	50	129,6	324	0	0
40	60	200	100	60	10	4	0	129,6	324	0	0
40	60	200	100	60	10	4	25	129,6	324	0	0
40	60	200	100	60	10	4	50	129,6	324	0	0
40	60	200	100	60	50	1	0	129,6	324	0	0
40	60	200	100	60	50	1	25	129,6	324	0	0
40	60	200	100	60	50	1	50	129,6	324	0	0
40	60	200	100	60	50	2	0	129,6	324	0	0
40	60	200	100	60	50	2	25	129,6	324	0	0
40	60	200	100	60	50	2	50	129,6	324	0	0
40	60	200	100	60	50	4	0	129,6	324	0	0
40	60	200	100	60	50	4	25	129,6	324	0	0
40	60	200	100	60	50	4	50	129,6	324	0	0
40	60	200	100	90	10	1	0	129,6	324	0	0
40	60	200	100	90	10	1	25	129,6	324	0	0
40	60	200	100	90	10	1	50	129,6	324	0	0
40	60	200	100	90	10	2	0	129,6	324	0	0
40	60	200	100	90	10	2	25	129,6	324	0	0
40	60	200	100	90	10	2	50	129,6	324	0	0
40	60	200	100	90	10	4	0	129,6	324	0	0
40	60	200	100	90	10	4	25	129,6	324	0	0
40	60	200	100	90	10	4	50	129,6	324	0	0
40	60	200	100	90	50	1	0	129,6	324	0	0
40	60	200	100	90	50	1	25	129,6	324	0	0
40	60	200	100	90	50	1	50	129,6	324	0	0
40	60	200	100	90	50	2	0	129,6	324	0	0
40	60	200	100	90	50	2	25	129,6	324	0	0
40	60	200	100	90	50	2	50	129,6	324	0	0
40	60	200	100	90	50	4	0	129,6	324	0	0
40	60	200	100	90	50	4	25	129,6	324	0	0
40	60	200	100	90	50	4	50	129,6	324	0	0

Tabla 42

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 50% y beta = 50%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
50	50	200	100	60	10	1	0	162	324	0	0
50	50	200	100	60	10	1	25	162	324	0	0
50	50	200	100	60	10	1	50	162	324	0	0
50	50	200	100	60	10	2	0	162	324	0	0
50	50	200	100	60	10	2	25	162	324	0	0
50	50	200	100	60	10	2	50	162	324	0	0
50	50	200	100	60	10	4	0	162	324	0	0
50	50	200	100	60	10	4	25	162	324	0	0
50	50	200	100	60	10	4	50	162	324	0	0
50	50	200	100	60	50	1	0	162	324	0	0
50	50	200	100	60	50	1	25	162	324	0	0
50	50	200	100	60	50	1	50	162	324	0	0
50	50	200	100	60	50	2	0	162	324	0	0
50	50	200	100	60	50	2	25	162	324	0	0
50	50	200	100	60	50	2	50	162	324	0	0
50	50	200	100	60	50	4	0	162	324	0	0
50	50	200	100	60	50	4	25	162	324	0	0
50	50	200	100	60	50	4	50	162	324	0	0
50	50	200	100	90	10	1	0	162	324	0	0
50	50	200	100	90	10	1	25	162	324	0	0
50	50	200	100	90	10	1	50	162	324	0	0
50	50	200	100	90	10	2	0	162	324	0	0
50	50	200	100	90	10	2	25	162	324	0	0
50	50	200	100	90	10	2	50	162	324	0	0
50	50	200	100	90	10	4	0	162	324	0	0
50	50	200	100	90	10	4	25	162	324	0	0
50	50	200	100	90	10	4	50	162	324	0	0
50	50	200	100	90	50	1	0	162	324	0	0
50	50	200	100	90	50	1	25	162	324	0	0
50	50	200	100	90	50	1	50	162	324	0	0
50	50	200	100	90	50	2	0	162	324	0	0
50	50	200	100	90	50	2	25	162	324	0	0
50	50	200	100	90	50	2	50	162	324	0	0
50	50	200	100	90	50	4	0	162	324	0	0
50	50	200	100	90	50	4	25	162	324	0	0
50	50	200	100	90	50	4	50	162	324	0	0

Tabla 43

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 60% y beta = 40%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
60	40	200	100	60	10	1	0	194,4	324	0	0
60	40	200	100	60	10	1	25	194,4	324	0	0
60	40	200	100	60	10	1	50	194,4	324	0	0
60	40	200	100	60	10	2	0	194,4	324	0	0
60	40	200	100	60	10	2	25	194,4	324	0	0
60	40	200	100	60	10	2	50	194,4	324	0	0
60	40	200	100	60	10	4	0	194,4	324	0	0
60	40	200	100	60	10	4	25	194,4	324	0	0
60	40	200	100	60	10	4	50	194,4	324	0	0
60	40	200	100	60	50	1	0	194,4	324	0	0
60	40	200	100	60	50	1	25	194,4	324	0	0
60	40	200	100	60	50	1	50	194,4	324	0	0
60	40	200	100	60	50	2	0	194,4	324	0	0
60	40	200	100	60	50	2	25	194,4	324	0	0
60	40	200	100	60	50	2	50	194,4	324	0	0
60	40	200	100	60	50	4	0	194,4	324	0	0
60	40	200	100	60	50	4	25	194,4	324	0	0
60	40	200	100	60	50	4	50	194,4	324	0	0
60	40	200	100	90	10	1	0	194,4	324	0	0
60	40	200	100	90	10	1	25	194,4	324	0	0
60	40	200	100	90	10	1	50	194,4	324	0	0
60	40	200	100	90	10	2	0	194,4	324	0	0
60	40	200	100	90	10	2	25	194,4	324	0	0
60	40	200	100	90	10	2	50	194,4	324	0	0
60	40	200	100	90	10	4	0	194,4	324	0	0
60	40	200	100	90	10	4	25	194,4	324	0	0
60	40	200	100	90	10	4	50	194,4	324	0	0
60	40	200	100	90	50	1	0	194,4	324	0	0
60	40	200	100	90	50	1	25	194,4	324	0	0
60	40	200	100	90	50	1	50	194,4	324	0	0
60	40	200	100	90	50	2	0	194,4	324	0	0
60	40	200	100	90	50	2	25	194,4	324	0	0
60	40	200	100	90	50	2	50	194,4	324	0	0
60	40	200	100	90	50	4	0	194,4	324	0	0
60	40	200	100	90	50	4	25	194,4	324	0	0
60	40	200	100	90	50	4	50	194,4	324	0	0

Tabla 44

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 70% y beta = 30%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
70	30	200	100	60	10	1	0	226,8	324	0	0
70	30	200	100	60	10	1	25	226,8	324	0	0
70	30	200	100	60	10	1	50	226,8	324	0	0
70	30	200	100	60	10	2	0	226,8	324	0	0
70	30	200	100	60	10	2	25	226,8	324	0	0
70	30	200	100	60	10	2	50	226,8	324	0	0
70	30	200	100	60	10	4	0	226,8	324	0	0
70	30	200	100	60	10	4	25	226,8	324	0	0
70	30	200	100	60	10	4	50	226,8	324	0	0
70	30	200	100	60	50	1	0	226,8	324	0	0
70	30	200	100	60	50	1	25	226,8	324	0	0
70	30	200	100	60	50	1	50	226,8	324	0	0
70	30	200	100	60	50	2	0	226,8	324	0	0
70	30	200	100	60	50	2	25	226,8	324	0	0
70	30	200	100	60	50	2	50	226,8	324	0	0
70	30	200	100	60	50	4	0	226,8	324	0	0
70	30	200	100	60	50	4	25	226,8	324	0	0
70	30	200	100	60	50	4	50	226,8	324	0	0
70	30	200	100	90	10	1	0	226,8	324	0	0
70	30	200	100	90	10	1	25	226,8	324	0	0
70	30	200	100	90	10	1	50	226,8	324	0	0
70	30	200	100	90	10	2	0	226,8	324	0	0
70	30	200	100	90	10	2	25	226,8	324	0	0
70	30	200	100	90	10	2	50	226,8	324	0	0
70	30	200	100	90	10	4	0	226,8	324	0	0
70	30	200	100	90	10	4	25	226,8	324	0	0
70	30	200	100	90	10	4	50	226,8	324	0	0
70	30	200	100	90	50	1	0	226,8	324	0	0
70	30	200	100	90	50	1	25	226,8	324	0	0
70	30	200	100	90	50	1	50	226,8	324	0	0
70	30	200	100	90	50	2	0	226,8	324	0	0
70	30	200	100	90	50	2	25	226,8	324	0	0
70	30	200	100	90	50	2	50	226,8	324	0	0
70	30	200	100	90	50	4	0	226,8	324	0	0
70	30	200	100	90	50	4	25	226,8	324	0	0
70	30	200	100	90	50	4	50	226,8	324	0	0

Tabla 45

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 80% y beta = 20%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
80	20	200	100	60	10	1	0	259,2	324	0	0
80	20	200	100	60	10	1	25	259,2	324	0	0
80	20	200	100	60	10	1	50	259,2	324	0	0
80	20	200	100	60	10	2	0	259,2	324	0	0
80	20	200	100	60	10	2	25	259,2	324	0	0
80	20	200	100	60	10	2	50	259,2	324	0	0
80	20	200	100	60	10	4	0	259,2	324	0	0
80	20	200	100	60	10	4	25	259,2	324	0	0
80	20	200	100	60	10	4	50	259,2	324	0	0
80	20	200	100	60	50	1	0	259,2	324	0	0
80	20	200	100	60	50	1	25	259,2	324	0	0
80	20	200	100	60	50	1	50	259,2	324	0	0
80	20	200	100	60	50	2	0	259,2	324	0	0
80	20	200	100	60	50	2	25	259,2	324	0	0
80	20	200	100	60	50	2	50	259,2	324	0	0
80	20	200	100	60	50	4	0	259,2	324	0	0
80	20	200	100	60	50	4	25	259,2	324	0	0
80	20	200	100	60	50	4	50	259,2	324	0	0
80	20	200	100	90	10	1	0	259,2	324	0	0
80	20	200	100	90	10	1	25	259,2	324	0	0
80	20	200	100	90	10	1	50	259,2	324	0	0
80	20	200	100	90	10	2	0	259,2	324	0	0
80	20	200	100	90	10	2	25	259,2	324	0	0
80	20	200	100	90	10	2	50	259,2	324	0	0
80	20	200	100	90	10	4	0	259,2	324	0	0
80	20	200	100	90	10	4	25	259,2	324	0	0
80	20	200	100	90	10	4	50	259,2	324	0	0
80	20	200	100	90	50	1	0	259,2	324	0	0
80	20	200	100	90	50	1	25	259,2	324	0	0
80	20	200	100	90	50	1	50	259,2	324	0	0
80	20	200	100	90	50	2	0	259,2	324	0	0
80	20	200	100	90	50	2	25	259,2	324	0	0
80	20	200	100	90	50	2	50	259,2	324	0	0
80	20	200	100	90	50	4	0	259,2	324	0	0
80	20	200	100	90	50	4	25	259,2	324	0	0
80	20	200	100	90	50	4	50	259,2	324	0	0

Tabla 46

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 90% y beta = 10%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
90	10	200	100	60	10	1	0	291,6	324	0	0
90	10	200	100	60	10	1	25	291,6	324	0	0
90	10	200	100	60	10	1	50	291,6	324	0	0
90	10	200	100	60	10	2	0	291,6	324	0	0
90	10	200	100	60	10	2	25	291,6	324	0	0
90	10	200	100	60	10	2	50	291,6	324	0	0
90	10	200	100	60	10	4	0	291,6	324	0	0
90	10	200	100	60	10	4	25	291,6	324	0	0
90	10	200	100	60	10	4	50	291,6	324	0	0
90	10	200	100	60	50	1	0	291,6	324	0	0
90	10	200	100	60	50	1	25	291,6	324	0	0
90	10	200	100	60	50	1	50	291,6	324	0	0
90	10	200	100	60	50	2	0	291,6	324	0	0
90	10	200	100	60	50	2	25	291,6	324	0	0
90	10	200	100	60	50	2	50	291,6	324	0	0
90	10	200	100	60	50	4	0	291,6	324	0	0
90	10	200	100	60	50	4	25	291,6	324	0	0
90	10	200	100	60	50	4	50	291,6	324	0	0
90	10	200	100	90	10	1	0	291,6	324	0	0
90	10	200	100	90	10	1	25	291,6	324	0	0
90	10	200	100	90	10	1	50	291,6	324	0	0
90	10	200	100	90	10	2	0	291,6	324	0	0
90	10	200	100	90	10	2	25	291,6	324	0	0
90	10	200	100	90	10	2	50	291,6	324	0	0
90	10	200	100	90	10	4	0	291,6	324	0	0
90	10	200	100	90	10	4	25	291,6	324	0	0
90	10	200	100	90	10	4	50	291,6	324	0	0
90	10	200	100	90	50	1	0	291,6	324	0	0
90	10	200	100	90	50	1	25	291,6	324	0	0
90	10	200	100	90	50	1	50	291,6	324	0	0
90	10	200	100	90	50	2	0	291,6	324	0	0
90	10	200	100	90	50	2	25	291,6	324	0	0
90	10	200	100	90	50	2	50	291,6	324	0	0
90	10	200	100	90	50	4	0	291,6	324	0	0
90	10	200	100	90	50	4	25	291,6	324	0	0
90	10	200	100	90	50	4	50	291,6	324	0	0

Tabla 47

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 0% y beta = 100%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
0	100	200	100	60	10	1	0	815	1072	815	21
0	100	200	100	60	10	1	25	476	1124	476	190
0	100	200	100	60	10	1	50	709	1115	709	39
0	100	200	100	60	10	2	0	774	1065	774	107
0	100	200	100	60	10	2	25	815	1072	815	12
0	100	200	100	60	10	2	50	623	1072	623	47
0	100	200	100	60	10	4	0	604	1134	604	32
0	100	200	100	60	10	4	25	600	1132	600	33
0	100	200	100	60	10	4	50	647	1075	647	24
0	100	200	100	60	50	1	0	719	1096	719	177
0	100	200	100	60	50	1	25	623	1072	623	72
0	100	200	100	60	50	1	50	565	1087	565	25
0	100	200	100	60	50	2	0	699	1039	699	15
0	100	200	100	60	50	2	25	623	1072	623	63
0	100	200	100	60	50	2	50	476	1124	476	44
0	100	200	100	60	50	4	0	688	1085	688	57
0	100	200	100	60	50	4	25	546	1059	546	83
0	100	200	100	60	50	4	50	476	1124	476	95
0	100	200	100	90	10	1	0	623	1072	623	127
0	100	200	100	90	10	1	25	600	1132	600	86
0	100	200	100	90	10	1	50	413	1139	413	20
0	100	200	100	90	10	2	0	841	1088	841	42
0	100	200	100	90	10	2	25	623	1072	623	12
0	100	200	100	90	10	2	50	613	1093	613	21
0	100	200	100	90	10	4	0	832	1098	832	157
0	100	200	100	90	10	4	25	605	1093	605	97
0	100	200	100	90	10	4	50	476	1124	476	6
0	100	200	100	90	50	1	0	842	1091	842	3
0	100	200	100	90	50	1	25	623	1072	623	18
0	100	200	100	90	50	1	50	619	1093	619	9
0	100	200	100	90	50	2	0	669	1096	669	117
0	100	200	100	90	50	2	25	630	1049	630	6
0	100	200	100	90	50	2	50	623	1072	623	13
0	100	200	100	90	50	4	0	538	1031	538	123
0	100	200	100	90	50	4	25	492	1135	492	178
0	100	200	100	90	50	4	50	546	1059	546	168

Tabla 48

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 100% y beta = 0%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
100	0	200	100	60	10	1	0	1026	1026	1122	87
100	0	200	100	60	10	1	25	1012	1012	1475	14
100	0	200	100	60	10	1	50	1007	1007	1398	30
100	0	200	100	60	10	2	0	1019	1019	1207	23
100	0	200	100	60	10	2	25	1012	1012	1432	189
100	0	200	100	60	10	2	50	1016	1016	1384	14
100	0	200	100	60	10	4	0	1002	1002	909	86
100	0	200	100	60	10	4	25	1002	1002	909	3
100	0	200	100	60	10	4	50	1010	1010	957	22
100	0	200	100	60	50	1	0	1007	1007	1430	137
100	0	200	100	60	50	1	25	1012	1012	1403	5
100	0	200	100	60	50	1	50	1012	1012	1007	101
100	0	200	100	60	50	2	0	1012	1012	1443	82
100	0	200	100	60	50	2	25	994	994	813	33
100	0	200	100	60	50	2	50	994	994	813	11
100	0	200	100	60	50	4	0	994	994	813	199
100	0	200	100	60	50	4	25	994	994	813	10
100	0	200	100	60	50	4	50	1012	1012	1359	41
100	0	200	100	90	10	1	0	1012	1012	1324	42
100	0	200	100	90	10	1	25	1036	1036	1261	11
100	0	200	100	90	10	1	50	1012	1012	1409	21
100	0	200	100	90	10	2	0	1012	1012	1193	10
100	0	200	100	90	10	2	25	1012	1012	1452	23
100	0	200	100	90	10	2	50	994	994	765	18
100	0	200	100	90	10	4	0	1016	1016	1415	130
100	0	200	100	90	10	4	25	1019	1019	1127	130
100	0	200	100	90	10	4	50	1014	1014	1197	26
100	0	200	100	90	50	1	0	994	994	813	100
100	0	200	100	90	50	1	25	1019	1019	1318	100
100	0	200	100	90	50	1	50	1012	1012	1324	57
100	0	200	100	90	50	2	0	1014	1014	1101	186
100	0	200	100	90	50	2	25	994	994	813	3
100	0	200	100	90	50	2	50	1001	1001	1332	26
100	0	200	100	90	50	4	0	1012	1012	1233	23
100	0	200	100	90	50	4	25	1012	1012	1447	16
100	0	200	100	90	50	4	50	1012	1012	1227	52

Tabla 49

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 10% y beta = 90%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
10	90	200	100	60	10	1	0	773,4	1056	742	7
10	90	200	100	60	10	1	25	667,9	1072	623	15
10	90	200	100	60	10	1	50	606,5	1097	552	19
10	90	200	100	60	10	2	0	672,2	1142	620	65
10	90	200	100	60	10	2	25	741,3	1113	700	147
10	90	200	100	60	10	2	50	617,2	1087	565	34
10	90	200	100	60	10	4	0	772,6	1030	744	117
10	90	200	100	60	10	4	25	622	1072	572	53
10	90	200	100	60	10	4	50	563,5	1135	500	25
10	90	200	100	60	50	1	0	810,5	1076	781	150
10	90	200	100	60	50	1	25	610,6	1066	560	172
10	90	200	100	60	50	1	50	753	1122	712	18
10	90	200	100	60	50	2	0	733	1039	699	151
10	90	200	100	60	50	2	25	707,8	1030	672	89
10	90	200	100	60	50	2	50	736,3	1072	699	38
10	90	200	100	60	50	4	0	617,2	1087	565	92
10	90	200	100	60	50	4	25	563,5	1135	500	33
10	90	200	100	60	50	4	50	587,3	1031	538	98
10	90	200	100	90	10	1	0	806,5	1072	777	47
10	90	200	100	90	10	1	25	634,2	1122	580	13
10	90	200	100	90	10	1	50	751,3	1096	713	18
10	90	200	100	90	10	2	0	669	1083	623	26
10	90	200	100	90	10	2	25	667,9	1072	623	20
10	90	200	100	90	10	2	50	697,4	1079	655	66
10	90	200	100	90	10	4	0	690,5	1082	647	33
10	90	200	100	90	10	4	25	661	1093	613	90
10	90	200	100	90	10	4	50	667,9	1072	623	28
10	90	200	100	90	50	1	0	653,8	1093	605	4
10	90	200	100	90	50	1	25	688,2	1077	645	29
10	90	200	100	90	50	1	50	660,4	1078	614	14
10	90	200	100	90	50	2	0	789,6	1119	753	163
10	90	200	100	90	50	2	25	563,5	1135	500	24
10	90	200	100	90	50	2	50	634,2	1122	580	20
10	90	200	100	90	50	4	0	750,5	1079	714	191
10	90	200	100	90	50	4	25	659,7	1116	609	59
10	90	200	100	90	50	4	50	563,5	1135	500	126

Tabla 50

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 20% y beta = 80%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
20	80	200	100	60	10	1	0	925,2	1078	887	65
20	80	200	100	60	10	1	25	736,4	1082	650	12
20	80	200	100	60	10	1	50	808	1072	742	43
20	80	200	100	60	10	2	0	785,4	1083	711	114
20	80	200	100	60	10	2	25	709	1093	613	7
20	80	200	100	60	10	2	50	709	1093	613	7
20	80	200	100	60	10	4	0	820,2	1077	756	94
20	80	200	100	60	10	4	25	712,8	1072	623	12
20	80	200	100	60	10	4	50	649,8	1065	546	48
20	80	200	100	60	50	1	0	768	1088	688	186
20	80	200	100	60	50	1	25	780	1124	694	163
20	80	200	100	60	50	1	50	636,6	1031	538	31
20	80	200	100	60	50	2	0	712,8	1072	623	59
20	80	200	100	60	50	2	25	644,8	1088	534	60
20	80	200	100	60	50	2	50	709	1093	613	9
20	80	200	100	60	50	4	0	655	1143	533	114
20	80	200	100	60	50	4	25	622	1110	500	171
20	80	200	100	60	50	4	50	658	1054	559	19
20	80	200	100	90	10	1	0	716,6	1091	623	1
20	80	200	100	90	10	1	25	648,6	1059	546	99
20	80	200	100	90	10	1	50	712,8	1072	623	17
20	80	200	100	90	10	2	0	848,8	1072	793	153
20	80	200	100	90	10	2	25	712,8	1072	623	65
20	80	200	100	90	10	2	50	688,4	1122	580	17
20	80	200	100	90	10	4	0	651,2	1060	549	68
20	80	200	100	90	10	4	25	669,4	1087	565	19
20	80	200	100	90	10	4	50	702,6	1093	605	93
20	80	200	100	90	50	1	0	728,4	1082	640	142
20	80	200	100	90	50	1	25	743,6	1030	672	142
20	80	200	100	90	50	1	50	804,8	1056	742	12
20	80	200	100	90	50	2	0	714,2	1091	620	155
20	80	200	100	90	50	2	25	710	1082	617	20
20	80	200	100	90	50	2	50	686	1078	588	145
20	80	200	100	90	50	4	0	710	1134	604	71
20	80	200	100	90	50	4	25	636,6	1031	538	67
20	80	200	100	90	50	4	50	652,2	1061	550	35

Tabla 51

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 30% y beta = 70%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
30	70	200	100	60	10	1	0	850,1	1100	743	8
30	70	200	100	60	10	1	25	772,5	1082	640	173
30	70	200	100	60	10	1	50	829,8	1030	744	26
30	70	200	100	60	10	2	0	861,8	1125	749	33
30	70	200	100	60	10	2	25	756,5	1082	617	136
30	70	200	100	60	10	2	50	670,4	1124	476	23
30	70	200	100	60	10	4	0	851,8	1059	763	146
30	70	200	100	60	10	4	25	746	1082	602	59
30	70	200	100	60	10	4	50	785	1093	653	37
30	70	200	100	60	50	1	0	789,3	1082	664	10
30	70	200	100	60	50	1	25	685,9	1031	538	44
30	70	200	100	60	50	1	50	685,9	1031	538	56
30	70	200	100	60	50	2	0	819,3	1077	709	40
30	70	200	100	60	50	2	25	757,6	1072	623	44
30	70	200	100	60	50	2	50	757,6	1072	623	79
30	70	200	100	60	50	4	0	775,3	1091	640	36
30	70	200	100	60	50	4	25	685,9	1031	538	101
30	70	200	100	60	50	4	50	751,4	1093	605	10
30	70	200	100	90	10	1	0	892,1	1072	815	73
30	70	200	100	90	10	1	25	755,7	1049	630	43
30	70	200	100	90	10	1	50	721,5	1087	565	19
30	70	200	100	90	10	2	0	883,8	1119	783	56
30	70	200	100	90	10	2	25	757,6	1072	623	12
30	70	200	100	90	10	2	50	735	1078	588	50
30	70	200	100	90	10	4	0	868,8	1041	795	130
30	70	200	100	90	10	4	25	742,6	1122	580	110
30	70	200	100	90	10	4	50	742,6	1122	580	26
30	70	200	100	90	50	1	0	801	1039	699	134
30	70	200	100	90	50	1	25	757	1093	613	5
30	70	200	100	90	50	1	50	755,7	1049	630	60
30	70	200	100	90	50	2	0	829,8	1051	735	60
30	70	200	100	90	50	2	25	685,9	1031	538	30
30	70	200	100	90	50	2	50	827,9	1096	713	111
30	70	200	100	90	50	4	0	801	1039	699	57
30	70	200	100	90	50	4	25	757,7	1072	623	108
30	70	200	100	90	50	4	50	685,9	1031	538	33

Tabla 52

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 40% y beta = 60%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
40	60	200	100	60	10	1	0	837,4	1075	679	25
40	60	200	100	60	10	1	25	773,8	1087	565	30
40	60	200	100	60	10	1	50	814	1093	628	29
40	60	200	100	60	10	2	0	816,8	1082	640	16
40	60	200	100	60	10	2	25	817,8	1077	645	110
40	60	200	100	60	10	2	50	796,8	1122	580	20
40	60	200	100	60	10	4	0	867,2	1088	720	53
40	60	200	100	60	10	4	25	816	1089	634	9
40	60	200	100	60	10	4	50	867,2	1088	720	44
40	60	200	100	60	50	1	0	897,2	1082	774	3
40	60	200	100	60	50	1	25	807,4	1084	623	115
40	60	200	100	60	50	1	50	802,6	1072	623	10
40	60	200	100	60	50	2	0	840,2	1028	715	130
40	60	200	100	60	50	2	25	773,8	1087	565	25
40	60	200	100	60	50	2	50	751,2	1059	546	124
40	60	200	100	60	50	4	0	800,2	1093	605	171
40	60	200	100	60	50	4	25	773,8	1087	565	34
40	60	200	100	60	50	4	50	754	1135	500	40
40	60	200	100	90	10	1	0	757	1054	559	38
40	60	200	100	90	10	1	25	805	1093	613	55
40	60	200	100	90	10	1	50	823,2	1083	650	21
40	60	200	100	90	10	2	0	773,8	1087	565	140
40	60	200	100	90	10	2	25	802,6	1072	623	9
40	60	200	100	90	10	2	50	738,4	1039	538	34
40	60	200	100	90	10	4	0	835	1039	699	161
40	60	200	100	90	10	4	25	735,2	1031	538	197
40	60	200	100	90	10	4	50	800,2	1093	605	35
40	60	200	100	90	50	1	0	773,8	1087	565	48
40	60	200	100	90	50	1	25	754	1135	500	102
40	60	200	100	90	50	1	50	767,2	1066	568	37
40	60	200	100	90	50	2	0	816	1089	634	107
40	60	200	100	90	50	2	25	802,6	1072	623	13
40	60	200	100	90	50	2	50	755	1061	551	75
40	60	200	100	90	50	4	0	802,6	1072	623	30
40	60	200	100	90	50	4	25	754	1135	500	42
40	60	200	100	90	50	4	50	753,4	1060	549	23

Tabla 53

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 50% y beta = 50%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
50	50	200	100	60	10	1	0	847,5	1072	623	10
50	50	200	100	60	10	1	25	861	1082	640	12
50	50	200	100	60	10	1	50	838	1057	619	75
50	50	200	100	60	10	2	0	899	1056	742	111
50	50	200	100	60	10	2	25	847,5	1072	623	95
50	50	200	100	60	10	2	50	826	1087	565	14
50	50	200	100	60	10	4	0	804	1060	548	106
50	50	200	100	60	10	4	25	851	1030	672	19
50	50	200	100	60	10	4	50	861	1082	640	17
50	50	200	100	60	50	1	0	911	1059	763	54
50	50	200	100	60	50	1	25	826	1087	565	91
50	50	200	100	60	50	1	50	847,5	1072	623	37
50	50	200	100	60	50	2	0	847,5	1072	623	61
50	50	200	100	60	50	2	25	847,5	1072	623	8
50	50	200	100	60	50	2	50	847,5	1072	623	8
50	50	200	100	60	50	4	0	841	1054	628	32
50	50	200	100	60	50	4	25	839,5	1049	630	13
50	50	200	100	60	50	4	50	826	1087	565	18
50	50	200	100	90	10	1	0	847,5	1072	623	183
50	50	200	100	90	10	1	25	881	1077	685	9
50	50	200	100	90	10	1	50	800	1124	476	10
50	50	200	100	90	10	2	0	847,5	1072	623	87
50	50	200	100	90	10	2	25	904,5	1096	713	33
50	50	200	100	90	10	2	50	847,5	1072	623	62
50	50	200	100	90	10	4	0	897,5	1077	718	33
50	50	200	100	90	10	4	25	803	1059,5	548	31
50	50	200	100	90	10	4	50	847,5	1072	623	8
50	50	200	100	90	50	1	0	907	1034	780	61
50	50	200	100	90	50	1	25	847,5	1072	623	93
50	50	200	100	90	50	1	50	804,5	1060	549	78
50	50	200	100	90	50	2	0	929	1122	736	78
50	50	200	100	90	50	2	25	847,5	1072	623	78
50	50	200	100	90	50	2	50	849,5	1054	645	98
50	50	200	100	90	50	4	0	853	1083	623	87
50	50	200	100	90	50	4	25	784,5	1031	538	177
50	50	200	100	90	50	4	50	813,5	1135	492	33

Tabla 54

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 60% y beta = 40%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
60	40	200	100	60	10	1	0	843,4	1047	538	166
60	40	200	100	60	10	1	25	903	1039	699	6
60	40	200	100	60	10	1	50	910,2	1095	633	44
60	40	200	100	60	10	2	0	969,2	1072	815	21
60	40	200	100	60	10	2	25	889,6	1036	670	17
60	40	200	100	60	10	2	50	878,2	1087	565	17
60	40	200	100	60	10	4	0	903	1077	642	46
60	40	200	100	60	10	4	25	913,8	1085	657	147
60	40	200	100	60	10	4	50	856,6	1061	550	86
60	40	200	100	60	50	1	0	967,8	1067	819	5
60	40	200	100	60	50	1	25	878,2	1087	565	22
60	40	200	100	60	50	1	50	895	1057	652	22
60	40	200	100	60	50	2	0	889,6	1036	670	34
60	40	200	100	60	50	2	25	892,4	1072	623	7
60	40	200	100	60	50	2	50	899,8	1033	700	12
60	40	200	100	60	50	4	0	892,4	1072	623	125
60	40	200	100	60	50	4	25	864,8	1124	476	35
60	40	200	100	60	50	4	50	901	1093	613	14
60	40	200	100	90	10	1	0	974	1078	818	26
60	40	200	100	90	10	1	25	928,4	1074	710	21
60	40	200	100	90	10	1	50	888,4	1080	601	52
60	40	200	100	90	10	2	0	983,8	1075	847	23
60	40	200	100	90	10	2	25	892,4	1072	623	5
60	40	200	100	90	10	2	50	865,8	1099	516	24
60	40	200	100	90	10	4	0	833,8	1031	538	148
60	40	200	100	90	10	4	25	864,8	1124	476	52
60	40	200	100	90	10	4	50	886,8	1030	672	82
60	40	200	100	90	50	1	0	978,2	1099	797	18
60	40	200	100	90	50	1	25	886,8	1030	672	26
60	40	200	100	90	50	1	50	878,2	1087	565	63
60	40	200	100	90	50	2	0	889,6	1036	670	7
60	40	200	100	90	50	2	25	889,6	1036	670	16
60	40	200	100	90	50	2	50	914,2	1041	724	94
60	40	200	100	90	50	4	0	886,8	1030	672	83
60	40	200	100	90	50	4	25	856	1054	559	147
60	40	200	100	90	50	4	50	833,8	1031	538	32

Tabla 55

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 70% y beta = 30%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
70	30	200	100	60	10	1	0	926,2	1036	670	28
70	30	200	100	60	10	1	25	929,6	1124	476	9
70	30	200	100	60	10	1	50	915,5	1067	562	19
70	30	200	100	60	10	2	0	970,2	1059	763	11
70	30	200	100	60	10	2	25	923,3	1049	630	17
70	30	200	100	60	10	2	50	923,2	1057	611	26
70	30	200	100	60	10	4	0	926,2	1036	670	59
70	30	200	100	60	10	4	25	926,2	1036	670	23
70	30	200	100	60	10	4	50	936,7	1075	714	6
70	30	200	100	60	50	1	0	960,1	1072	699	23
70	30	200	100	60	50	1	25	922,6	1030	672	54
70	30	200	100	60	50	1	50	926,2	1036	670	11
70	30	200	100	60	50	2	0	926,2	1036	670	40
70	30	200	100	60	50	2	25	922,6	1030	672	22
70	30	200	100	60	50	2	50	905,7	1059	548	115
70	30	200	100	60	50	4	0	910,2	1050	584	106
70	30	200	100	60	50	4	25	905,1	1059	546	51
70	30	200	100	60	50	4	50	905,1	1059	546	85
70	30	200	100	90	10	1	0	937	1039	699	1
70	30	200	100	90	10	1	25	962,9	1083	683	23
70	30	200	100	90	10	1	50	939,5	1079	614	19
70	30	200	100	90	10	2	0	949,4	1082	640	47
70	30	200	100	90	10	2	25	937	1039	699	9
70	30	200	100	90	10	2	50	933,1	1033	700	37
70	30	200	100	90	10	4	0	949,4	1082	640	180
70	30	200	100	90	10	4	25	883,1	1031	538	7
70	30	200	100	90	10	4	50	934,1	1028	715	17
70	30	200	100	90	50	1	0	937	1039	699	198
70	30	200	100	90	50	1	25	956,2	1081	665	40
70	30	200	100	90	50	1	50	938,5	1057	662	30
70	30	200	100	90	50	2	0	999,4	1091	786	188
70	30	200	100	90	50	2	25	952,5	1089	634	14
70	30	200	100	90	50	2	50	956,4	1095	633	46
70	30	200	100	90	50	4	0	934,1	1028	715	63
70	30	200	100	90	50	4	25	922,6	1030	672	34
70	30	200	100	90	50	4	50	926,2	1036	670	22

Tabla 56

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 80% y beta = 20%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
80	20	200	100	60	10	1	0	960,2	1000	801	3
80	20	200	100	60	10	1	25	993,6	1082	640	9
80	20	200	100	60	10	1	50	1008,4	1055	822	16
80	20	200	100	60	10	2	0	1013,6	1122	580	13
80	20	200	100	60	10	2	25	962,8	1036	670	16
80	20	200	100	60	10	2	50	991,8	1067	691	31
80	20	200	100	60	10	4	0	772,6	1030	744	117
80	20	200	100	60	10	4	25	622	1072	572	53
80	20	200	100	60	10	4	50	563,5	1135	500	25
80	20	200	100	60	50	1	0	970,6	1010	813	171
80	20	200	100	60	50	1	25	957	1054	569	38
80	20	200	100	60	50	1	50	962,8	1036	670	14
80	20	200	100	60	50	2	0	932,4	1031	538	161
80	20	200	100	60	50	2	25	962,8	1036	670	196
80	20	200	100	60	50	2	50	1005,4	1048	835	103
80	20	200	100	60	50	4	0	973,4	1042	699	84
80	20	200	100	60	50	4	25	563,5	1135	500	33
80	20	200	100	60	50	4	50	587,3	1031	538	98
80	20	200	100	90	10	1	0	1009	1077	737	15
80	20	200	100	90	10	1	25	987,8	1051	735	25
80	20	200	100	90	10	1	50	982,2	1072	623	20
80	20	200	100	90	10	2	0	970,6	1010	813	23
80	20	200	100	90	10	2	25	991,8	1041	795	80
80	20	200	100	90	10	2	50	1004	1077	712	60
80	20	200	100	90	10	4	0	982,2	1072	623	142
80	20	200	100	90	10	4	25	962,8	1036	670	9
80	20	200	100	90	10	4	50	971	1039	699	8
80	20	200	100	90	50	1	0	977	1046	701	176
80	20	200	100	90	50	1	25	932,4	1031	538	3
80	20	200	100	90	50	1	50	962,8	1036	670	91
80	20	200	100	90	50	2	0	948,2	994	765	68
80	20	200	100	90	50	2	25	932,4	1031	538	68
80	20	200	100	90	50	2	50	938,8	1039	538	26
80	20	200	100	90	50	4	0	948,2	994	765	34
80	20	200	100	90	50	4	25	962,8	1036	670	22
80	20	200	100	90	50	4	50	956,4	1059	546	22

Tabla 57

Resultados de las corridas de prueba del problema mediano (Problema 7) para la combinación de ponderación alfa = 90% y beta = 10%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
90	10	200	100	60	10	1	0	1015,3	1067	550	15
90	10	200	100	60	10	1	25	1022	1051	761	28
90	10	200	100	60	10	1	50	1031	1057	797	10
90	10	200	100	60	10	2	0	980,7	1002	789	3
90	10	200	100	60	10	2	25	979,8	1002	780	12
90	10	200	100	60	10	2	50	993,4	1011	835	30
90	10	200	100	60	10	4	0	996,7	1028	715	9
90	10	200	100	60	10	4	25	993,4	1011	835	13
90	10	200	100	60	10	4	50	1011,2	1046	698	29
90	10	200	100	60	50	1	0	1014,9	1017	996	199
90	10	200	100	60	50	1	25	1005	1039	699	25
90	10	200	100	60	50	1	50	1011,5	1012	1007	25
90	10	200	100	60	50	2	0	1012,5	1046	711	49
90	10	200	100	60	50	2	25	999,9	1028	747	26
90	10	200	100	60	50	2	50	996,7	1028	715	39
90	10	200	100	60	50	4	0	994,2	1005	897	102
90	10	200	100	60	50	4	25	999,4	1036	670	10
90	10	200	100	60	50	4	50	971,1	994	765	81
90	10	200	100	90	10	1	0	981,7	1031	538	20
90	10	200	100	90	10	1	25	971,1	994	765	71
90	10	200	100	90	10	1	50	1021,4	1016	1070	50
90	10	200	100	90	10	2	0	971,1	994	765	198
90	10	200	100	90	10	2	25	1019,4	1051	735	12
90	10	200	100	90	10	2	50	999,9	1028	747	13
90	10	200	100	90	10	4	0	1011	1013	993	16
90	10	200	100	90	10	4	25	971,1	994	765	15
90	10	200	100	90	10	4	50	971,1	994	765	15
90	10	200	100	90	50	1	0	1011,5	1040	755	175
90	10	200	100	90	50	1	25	981,7	1031	538	180
90	10	200	100	90	50	1	50	1022,3	1020	1043	16
90	10	200	100	90	50	2	0	1018,6	1036	862	161
90	10	200	100	90	50	2	25	1004,5	1039	694	12
90	10	200	100	90	50	2	50	971,1	994	765	71
90	10	200	100	90	50	4	0	983,7	1000	837	163
90	10	200	100	90	50	4	25	981,7	1031	538	29
90	10	200	100	90	50	4	50	1005	1039	699	14

Tabla 58

Resultados de las corridas de prueba del problema grande (Problema 9) para la combinación de ponderación alfa = 0% y beta = 100%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
0	100	200	100	60	10	1	0	0	291	0	4
0	100	200	100	60	10	1	25	0	291	0	7
0	100	200	100	60	10	1	50	0	291	0	75
0	100	200	100	60	10	2	0	0	291	0	0
0	100	200	100	60	10	2	25	0	291	0	72
0	100	200	100	60	10	2	50	0	291	0	0
0	100	200	100	60	10	4	0	0	291	0	0
0	100	200	100	60	10	4	25	0	291	0	0
0	100	200	100	60	10	4	50	0	291	0	4
0	100	200	100	60	50	1	0	0	291	0	1
0	100	200	100	60	50	1	25	0	291	0	0
0	100	200	100	60	50	1	50	0	291	0	0
0	100	200	100	60	50	2	0	0	291	0	3
0	100	200	100	60	50	2	25	0	291	0	0
0	100	200	100	60	50	2	50	0	291	0	0
0	100	200	100	60	50	4	0	0	291	0	0
0	100	200	100	60	50	4	25	0	291	0	0
0	100	200	100	60	50	4	50	0	291	0	0
0	100	200	100	90	10	1	0	0	291	0	9
0	100	200	100	90	10	1	25	0	291	0	0
0	100	200	100	90	10	1	50	0	291	0	28
0	100	200	100	90	10	2	0	0	291	0	0
0	100	200	100	90	10	2	25	0	291	0	0
0	100	200	100	90	10	2	50	0	291	0	0
0	100	200	100	90	10	4	0	0	291	0	0
0	100	200	100	90	10	4	25	0	291	0	0
0	100	200	100	90	10	4	50	0	291	0	0
0	100	200	100	90	50	1	0	0	291	0	114
0	100	200	100	90	50	1	25	0	291	0	0
0	100	200	100	90	50	1	50	0	291	0	17
0	100	200	100	90	50	2	0	0	291	0	100
0	100	200	100	90	50	2	25	0	291	0	0
0	100	200	100	90	50	2	50	0	291	0	0
0	100	200	100	90	50	4	0	0	291	0	7
0	100	200	100	90	50	4	25	0	291	0	0
0	100	200	100	90	50	4	50	0	291	0	2

Tabla 59

Resultados de las corridas de prueba del problema grande (Problema 9) para la combinación de ponderación alfa = 100% y beta = 0%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
100	0	200	100	60	10	1	0	254	254	58	57
100	0	200	100	60	10	1	25	254	254	63	11
100	0	200	100	60	10	1	50	254	254	42	0
100	0	200	100	60	10	2	0	254	254	27	49
100	0	200	100	60	10	2	25	253	253	57	107
100	0	200	100	60	10	2	50	250	250	34	74
100	0	200	100	60	10	4	0	250	250	34	57
100	0	200	100	60	10	4	25	254	254	46	20
100	0	200	100	60	10	4	50	254	254	63	29
100	0	200	100	60	50	1	0	253	253	45	73
100	0	200	100	60	50	1	25	251	251	118	15
100	0	200	100	60	50	1	50	254	254	40	92
100	0	200	100	60	50	2	0	250	250	34	5
100	0	200	100	60	50	2	25	249	249	76	28
100	0	200	100	60	50	2	50	249	249	64	121
100	0	200	100	60	50	4	0	250	250	48	122
100	0	200	100	60	50	4	25	250	250	34	45
100	0	200	100	60	50	4	50	249	249	64	126
100	0	200	100	90	10	1	0	254	254	32	13
100	0	200	100	90	10	1	25	254	254	28	11
100	0	200	100	90	10	1	50	250	250	34	132
100	0	200	100	90	10	2	0	254	254	66	24
100	0	200	100	90	10	2	25	251	251	17	15
100	0	200	100	90	10	2	50	254	254	59	19
100	0	200	100	90	10	4	0	253	253	37	42
100	0	200	100	90	10	4	25	254	254	33	9
100	0	200	100	90	10	4	50	254	254	41	24
100	0	200	100	90	50	1	0	250	250	34	31
100	0	200	100	90	50	1	25	254	254	28	2
100	0	200	100	90	50	1	50	255	255	74	0
100	0	200	100	90	50	2	0	255	255	40	7
100	0	200	100	90	50	2	25	251	251	119	27
100	0	200	100	90	50	2	50	253	253	59	34
100	0	200	100	90	50	4	0	250	250	48	82
100	0	200	100	90	50	4	25	254	254	51	15
100	0	200	100	90	50	4	50	254	254	23	13

Tabla 60

Resultados de las corridas de prueba del problema grande (Problema 9) para la combinación de ponderación alfa = 10% y beta = 90%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
10	90	200	100	60	10	1	0	26,7	267	0	9
10	90	200	100	60	10	1	25	26,7	267	0	39
10	90	200	100	60	10	1	50	28,5	285	0	36
10	90	200	100	60	10	2	0	28,1	281	0	0
10	90	200	100	60	10	2	25	25,8	258	0	164
10	90	200	100	60	10	2	50	28,2	282	0	166
10	90	200	100	60	10	4	0	26,5	265	0	36
10	90	200	100	60	10	4	25	26,5	265	0	81
10	90	200	100	60	10	4	50	26,3	263	0	78
10	90	200	100	60	50	1	0	25,7	257	0	89
10	90	200	100	60	50	1	25	26,5	265	0	85
10	90	200	100	60	50	1	50	26,9	269	0	146
10	90	200	100	60	50	2	0	27	270	0	60
10	90	200	100	60	50	2	25	25,7	257	0	153
10	90	200	100	60	50	2	50	26,5	265	0	51
10	90	200	100	60	50	4	0	26	260	0	121
10	90	200	100	60	50	4	25	26,1	261	0	59
10	90	200	100	60	50	4	50	26,5	265	0	72
10	90	200	100	90	10	1	0	28,8	288	0	6
10	90	200	100	90	10	1	25	26,3	263	0	142
10	90	200	100	90	10	1	50	25,8	258	0	117
10	90	200	100	90	10	2	0	28,5	285	0	12
10	90	200	100	90	10	2	25	29,1	291	1	15
10	90	200	100	90	10	2	50	28,8	288	0	0
10	90	200	100	90	10	4	0	27,9	279	0	57
10	90	200	100	90	10	4	25	26,1	261	0	185
10	90	200	100	90	10	4	50	26,5	265	0	129
10	90	200	100	90	50	1	0	27,7	277	0	175
10	90	200	100	90	50	1	25	28,8	288	0	68
10	90	200	100	90	50	1	50	26,1	261	0	71
10	90	200	100	90	50	2	0	26,4	264	0	69
10	90	200	100	90	50	2	25	26,8	268	0	48
10	90	200	100	90	50	2	50	28,1	281	0	67
10	90	200	100	90	50	4	0	26,1	261	0	28
10	90	200	100	90	50	4	25	26,6	266	0	48
10	90	200	100	90	50	4	50	25,8	258	0	186

Tabla 61

Resultados de las corridas de prueba del problema grande (Problema 9) para la combinación de ponderación alfa = 20% y beta = 80%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
20	80	200	100	60	10	1	0	52,2	261	0	2
20	80	200	100	60	10	1	25	51,2	256	0	70
20	80	200	100	60	10	1	50	52,8	264	0	139
20	80	200	100	60	10	2	0	55,8	279	0	0
20	80	200	100	60	10	2	25	52,2	261	0	32
20	80	200	100	60	10	2	50	52,6	263	0	101
20	80	200	100	60	10	4	0	52,2	261	0	119
20	80	200	100	60	10	4	25	51,4	257	0	33
20	80	200	100	60	10	4	50	51,2	256	0	57
20	80	200	100	60	50	1	0	53,2	266	0	136
20	80	200	100	60	50	1	25	51,6	258	0	85
20	80	200	100	60	50	1	50	51,4	257	0	61
20	80	200	100	60	50	2	0	52,2	261	0	181
20	80	200	100	60	50	2	25	52,6	263	0	35
20	80	200	100	60	50	2	50	52,2	261	0	19
20	80	200	100	60	50	4	0	53,4	267	0	184
20	80	200	100	60	50	4	25	53	265	0	62
20	80	200	100	60	50	4	50	52,2	261	0	39
20	80	200	100	90	10	1	0	52,6	263	0	96
20	80	200	100	90	10	1	25	52,2	261	0	4
20	80	200	100	90	10	1	50	53,2	258	2	33
20	80	200	100	90	10	2	0	56,4	282	0	1
20	80	200	100	90	10	2	25	57,8	269	5	7
20	80	200	100	90	10	2	50	54,2	271	0	0
20	80	200	100	90	10	4	0	55,6	270	2	0
20	80	200	100	90	10	4	25	52,8	264	0	17
20	80	200	100	90	10	4	50	51,4	257	0	99
20	80	200	100	90	50	1	0	56,4	282	0	3
20	80	200	100	90	50	1	25	52,8	264	0	198
20	80	200	100	90	50	1	50	53	265	0	70
20	80	200	100	90	50	2	0	52,2	261	0	96
20	80	200	100	90	50	2	25	52	260	0	182
20	80	200	100	90	50	2	50	52	260	0	166
20	80	200	100	90	50	4	0	51,4	257	0	141
20	80	200	100	90	50	4	25	51,2	256	0	194
20	80	200	100	90	50	4	50	51,2	256	0	49

Tabla 62

Resultados de las corridas de prueba del problema grande (Problema 9) para la combinación de ponderación alfa = 30% y beta = 70%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
30	70	200	100	60	10	1	0	80,1	267	0	9
30	70	200	100	60	10	1	25	76,8	256	0	46
30	70	200	100	60	10	1	50	76,8	256	0	67
30	70	200	100	60	10	2	0	77,1	257	0	132
30	70	200	100	60	10	2	25	77,1	257	0	112
30	70	200	100	60	10	2	50	77,1	257	0	89
30	70	200	100	60	10	4	0	82,4	268	3	47
30	70	200	100	60	10	4	25	78,3	261	0	176
30	70	200	100	60	10	4	50	78,3	261	0	47
30	70	200	100	60	50	1	0	79,4	258	3	193
30	70	200	100	60	50	1	25	77,3	258	0	137
30	70	200	100	60	50	1	50	76,8	256	0	52
30	70	200	100	60	50	2	0	78,3	261	0	119
30	70	200	100	60	50	2	25	77,7	259	0	30
30	70	200	100	60	50	2	50	76,8	256	0	118
30	70	200	100	60	50	4	0	78	260	0	45
30	70	200	100	60	50	4	25	76,8	256	0	29
30	70	200	100	60	50	4	50	76,8	256	0	85
30	70	200	100	90	10	1	0	79,8	266	0	1
30	70	200	100	90	10	1	25	80,3	268	0	1
30	70	200	100	90	10	1	50	77,1	257	0	27
30	70	200	100	90	10	2	0	79,2	264	0	17
30	70	200	100	90	10	2	25	77,1	257	0	38
30	70	200	100	90	10	2	50	79,2	264	0	48
30	70	200	100	90	10	4	0	84,6	282	0	2
30	70	200	100	90	10	4	25	79,5	265	0	19
30	70	200	100	90	10	4	50	77,7	259	0	73
30	70	200	100	90	50	1	0	77,1	257	0	23
30	70	200	100	90	50	1	25	78,3	261	0	24
30	70	200	100	90	50	1	50	77,1	257	0	18
30	70	200	100	90	50	2	0	79,5	265	0	130
30	70	200	100	90	50	2	25	78,3	261	0	95
30	70	200	100	90	50	2	50	78,3	261	0	79
30	70	200	100	90	50	4	0	77,3	258	0	60
30	70	200	100	90	50	4	25	78,8	263	0	55
30	70	200	100	90	50	4	50	77,1	257	0	96

Tabla 63

Resultados de las corridas de prueba del problema grande (Problema 9) para la combinación de ponderación alfa = 40% y beta = 60%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
40	60	200	100	60	10	1	0	104,4	261	0	111
40	60	200	100	60	10	1	25	108,4	268	2	119
40	60	200	100	60	10	1	50	111,6	261	12	8
40	60	200	100	60	10	2	0	105,6	258	4	12
40	60	200	100	60	10	2	25	105	258	3	117
40	60	200	100	60	10	2	50	102,8	257	0	38
40	60	200	100	60	10	4	0	104,4	258	2	132
40	60	200	100	60	10	4	25	106,4	257	6	28
40	60	200	100	60	10	4	50	102,4	256	0	23
40	60	200	100	60	50	1	0	106,8	261	4	146
40	60	200	100	60	50	1	25	102,4	256	0	68
40	60	200	100	60	50	1	50	104,4	261	0	12
40	60	200	100	60	50	2	0	106	265	0	89
40	60	200	100	60	50	2	25	103,2	258	0	14
40	60	200	100	60	50	2	50	102,8	257	0	131
40	60	200	100	60	50	4	0	104,4	258	2	76
40	60	200	100	60	50	4	25	103,2	258	0	88
40	60	200	100	60	50	4	50	102,4	256	0	48
40	60	200	100	90	10	1	0	107,2	268	0	21
40	60	200	100	90	10	1	25	107,4	264	3	4
40	60	200	100	90	10	1	50	102,8	257	0	22
40	60	200	100	90	10	2	0	107,4	264	3	3
40	60	200	100	90	10	2	25	104	260	0	43
40	60	200	100	90	10	2	50	102,4	256	0	28
40	60	200	100	90	10	4	0	109	268	3	106
40	60	200	100	90	10	4	25	102,4	256	0	130
40	60	200	100	90	10	4	50	102,4	256	0	28
40	60	200	100	90	50	1	0	102,8	257	0	9
40	60	200	100	90	50	1	25	102,8	257	0	126
40	60	200	100	90	50	1	50	102,8	257	0	39
40	60	200	100	90	50	2	0	104,4	261	0	9
40	60	200	100	90	50	2	25	106	265	0	93
40	60	200	100	90	50	2	50	102,4	256	0	69
40	60	200	100	90	50	4	0	103,6	259	0	133
40	60	200	100	90	50	4	25	102,8	257	0	108
40	60	200	100	90	50	4	50	102,4	256	0	91

Tabla 64

Resultados de las corridas de prueba del problema grande (Problema 9) para la combinación de ponderación alfa = 50% y beta = 50%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
50	50	200	100	60	10	1	0	134,5	266	3	9
50	50	200	100	60	10	1	25	129	258	0	21
50	50	200	100	60	10	1	50	130,5	258	3	39
50	50	200	100	60	10	2	0	135,5	268	3	0
50	50	200	100	60	10	2	25	128,5	257	0	41
50	50	200	100	60	10	2	50	128,5	257	0	29
50	50	200	100	60	10	4	0	134	256	12	8
50	50	200	100	60	10	4	25	129	258	0	47
50	50	200	100	60	10	4	50	128,5	257	0	8
50	50	200	100	60	50	1	0	135	264	6	31
50	50	200	100	60	50	1	25	128,5	257	0	111
50	50	200	100	60	50	1	50	128,5	257	0	28
50	50	200	100	60	50	2	0	130,5	261	0	14
50	50	200	100	60	50	2	25	131,5	263	0	191
50	50	200	100	60	50	2	50	128	256	0	188
50	50	200	100	60	50	4	0	130,5	261	0	36
50	50	200	100	60	50	4	25	128	256	0	58
50	50	200	100	60	50	4	50	130,5	261	0	24
50	50	200	100	90	10	1	0	135	256	14	41
50	50	200	100	90	10	1	25	128	256	0	39
50	50	200	100	90	10	1	50	129	258	0	61
50	50	200	100	90	10	2	0	132,5	265	0	18
50	50	200	100	90	10	2	25	130	258	2	179
50	50	200	100	90	10	2	50	129,5	259	0	99
50	50	200	100	90	10	4	0	128	256	0	97
50	50	200	100	90	10	4	25	128,5	257	0	63
50	50	200	100	90	10	4	50	130,5	261	0	21
50	50	200	100	90	50	1	0	131,5	260	3	67
50	50	200	100	90	50	1	25	136	264	8	12
50	50	200	100	90	50	1	50	128	256	0	164
50	50	200	100	90	50	2	0	131,5	260	3	80
50	50	200	100	90	50	2	25	129	258	0	49
50	50	200	100	90	50	2	50	128,5	257	0	19
50	50	200	100	90	50	4	0	133	266	0	19
50	50	200	100	90	50	4	25	129	258	0	28
50	50	200	100	90	50	4	50	128,5	257	0	38

Tabla 65

Resultados de las corridas de prueba del problema grande (Problema 9) para la combinación de ponderación alfa = 60% y beta = 40%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
60	40	200	100	60	10	1	0	154,2	257	0	5
60	40	200	100	60	10	1	25	154,2	257	0	71
60	40	200	100	60	10	1	50	154,7	256	3	56
60	40	200	100	60	10	2	0	156,6	261	0	43
60	40	200	100	60	10	2	25	153,6	256	0	175
60	40	200	100	60	10	2	50	153,6	256	0	126
60	40	200	100	60	10	4	0	159	265	0	181
60	40	200	100	60	10	4	25	154,2	257	0	20
60	40	200	100	60	10	4	50	155,6	258	2	6
60	40	200	100	60	50	1	0	161,4	261	12	3
60	40	200	100	60	50	1	25	154,7	258	0	99
60	40	200	100	60	50	1	50	156,6	261	0	72
60	40	200	100	60	50	2	0	155,4	259	0	181
60	40	200	100	60	50	2	25	159	265	0	18
60	40	200	100	60	50	2	50	153,6	256	0	46
60	40	200	100	60	50	4	0	153,6	256	0	153
60	40	200	100	60	50	4	25	153,6	256	0	66
60	40	200	100	60	50	4	50	154,2	257	0	59
60	40	200	100	90	10	1	0	156,6	257	6	25
60	40	200	100	90	10	1	25	156,6	261	0	29
60	40	200	100	90	10	1	50	155,9	258	3	12
60	40	200	100	90	10	2	0	160,2	257	15	35
60	40	200	100	90	10	2	25	154,7	258	0	29
60	40	200	100	90	10	2	50	153,6	256	0	98
60	40	200	100	90	10	4	0	161,4	261	12	11
60	40	200	100	90	10	4	25	158,3	258	9	28
60	40	200	100	90	10	4	50	154,2	257	0	0
60	40	200	100	90	50	1	0	154,7	258	0	83
60	40	200	100	90	50	1	25	159	261	6	31
60	40	200	100	90	50	1	50	153,6	256	0	5
60	40	200	100	90	50	2	0	156,6	261	0	33
60	40	200	100	90	50	2	25	153,6	256	0	14
60	40	200	100	90	50	2	50	153,6	256	0	36
60	40	200	100	90	50	4	0	155,4	259	0	142
60	40	200	100	90	50	4	25	154,2	257	0	196
60	40	200	100	90	50	4	50	154,2	257	0	28

Tabla 66

Resultados de las corridas de prueba del problema grande (Problema 9) para la combinación de ponderación alfa = 70% y beta = 30%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
70	30	200	100	60	10	1	0	181,2	258	2	44
70	30	200	100	60	10	1	25	179,8	257	0	27
70	30	200	100	60	10	1	50	181,8	256	9	21
70	30	200	100	60	10	2	0	183,2	258	9	163
70	30	200	100	60	10	2	25	179,2	256	0	29
70	30	200	100	60	10	2	50	179,8	257	0	16
70	30	200	100	60	10	4	0	185,3	261	9	26
70	30	200	100	60	10	4	25	180,6	258	0	84
70	30	200	100	60	10	4	50	179,8	257	0	78
70	30	200	100	60	50	1	0	179,2	256	0	56
70	30	200	100	60	50	1	25	180,6	258	0	16
70	30	200	100	60	50	1	50	179,2	256	0	22
70	30	200	100	60	50	2	0	181,2	258	2	7
70	30	200	100	60	50	2	25	179,2	256	0	92
70	30	200	100	60	50	2	50	179,2	256	0	17
70	30	200	100	60	50	4	0	180,6	258	0	86
70	30	200	100	60	50	4	25	179,2	256	0	25
70	30	200	100	60	50	4	50	180,6	258	0	94
70	30	200	100	90	10	1	0	180,6	258	0	65
70	30	200	100	90	10	1	25	179,8	257	0	11
70	30	200	100	90	10	1	50	184,7	258	14	15
70	30	200	100	90	10	2	0	183,1	259	6	103
70	30	200	100	90	10	2	25	179,8	257	0	18
70	30	200	100	90	10	2	50	181,8	256	9	10
70	30	200	100	90	10	4	0	180,6	258	0	77
70	30	200	100	90	10	4	25	181,8	256	9	88
70	30	200	100	90	10	4	50	179,8	257	0	31
70	30	200	100	90	50	1	0	182,7	261	0	11
70	30	200	100	90	50	1	25	181,2	259	0	43
70	30	200	100	90	50	1	50	180,6	258	0	30
70	30	200	100	90	50	2	0	183,2	258	9	147
70	30	200	100	90	50	2	25	179,8	257	0	28
70	30	200	100	90	50	2	50	179,2	256	0	10
70	30	200	100	90	50	4	0	180,6	258	0	85
70	30	200	100	90	50	4	25	182,7	261	0	20
70	30	200	100	90	50	4	50	179,8	257	0	32

Tabla 67

Resultados de las corridas de prueba del problema grande (Problema 9) para la combinación de ponderación alfa = 80% y beta = 20%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
80	20	200	100	60	10	1	0	206,6	256	9	94
80	20	200	100	60	10	1	25	206,6	256	9	25
80	20	200	100	60	10	1	50	204,8	256	0	16
80	20	200	100	60	10	2	0	207	258	3	70
80	20	200	100	60	10	2	25	204,8	256	0	42
80	20	200	100	60	10	2	50	204,8	256	0	30
80	20	200	100	60	10	4	0	206,8	258	2	19
80	20	200	100	60	10	4	25	207	258	3	26
80	20	200	100	60	10	4	50	207	258	3	12
80	20	200	100	60	50	1	0	206,6	256	9	135
80	20	200	100	60	50	1	25	205,6	257	0	13
80	20	200	100	60	50	1	50	205,6	257	0	19
80	20	200	100	60	50	2	0	207	256	11	173
80	20	200	100	60	50	2	25	205,4	256	3	179
80	20	200	100	60	50	2	50	204,8	256	0	15
80	20	200	100	60	50	4	0	205,6	257	0	90
80	20	200	100	60	50	4	25	205,6	257	0	29
80	20	200	100	60	50	4	50	205,6	257	0	35
80	20	200	100	90	10	1	0	211,6	257	30	72
80	20	200	100	90	10	1	25	207	258	3	23
80	20	200	100	90	10	1	50	210	261	6	20
80	20	200	100	90	10	2	0	208,8	261	0	13
80	20	200	100	90	10	2	25	207,4	257	9	34
80	20	200	100	90	10	2	50	206,6	256	9	9
80	20	200	100	90	10	4	0	205,6	257	0	72
80	20	200	100	90	10	4	25	205,6	257	0	20
80	20	200	100	90	10	4	50	208,8	261	0	15
80	20	200	100	90	50	1	0	204,8	256	0	120
80	20	200	100	90	50	1	25	204,8	256	0	23
80	20	200	100	90	50	1	50	205,6	257	0	36
80	20	200	100	90	50	2	0	204,8	256	0	22
80	20	200	100	90	50	2	25	206,8	258	2	60
80	20	200	100	90	50	2	50	204,8	256	0	41
80	20	200	100	90	50	4	0	207,2	259	0	41
80	20	200	100	90	50	4	25	206,4	258	0	23
80	20	200	100	90	50	4	50	206,8	257	6	41

Tabla 68

Resultados de las corridas de prueba del problema grande (Problema 9) para la combinación de ponderación alfa = 90% y beta = 10%

Alfa	Beta	Maxgen	P	Tc	Tm	Im	Te1	Objetivo Ponderado	Makespan	TTP	Gen
90	10	200	100	60	10	1	0	231,3	257	0	60
90	10	200	100	60	10	1	25	230,4	256	0	48
90	10	200	100	60	10	1	50	230,4	256	0	95
90	10	200	100	60	10	2	0	231,5	256	11	190
90	10	200	100	60	10	2	25	231,3	257	0	1
90	10	200	100	60	10	2	50	231,3	257	0	57
90	10	200	100	60	10	4	0	231,3	256	9	59
90	10	200	100	60	10	4	25	231,3	256	9	10
90	10	200	100	60	10	4	50	233,6	258	14	38
90	10	200	100	60	50	1	0	232,4	258	2	57
90	10	200	100	60	50	1	25	232,5	257	12	3
90	10	200	100	60	50	1	50	230,4	256	0	137
90	10	200	100	60	50	2	0	231,9	256	15	136
90	10	200	100	60	50	2	25	230,7	256	3	45
90	10	200	100	60	50	2	50	230,4	256	0	0
90	10	200	100	60	50	4	0	231,3	257	0	118
90	10	200	100	60	50	4	25	231,3	256	9	44
90	10	200	100	60	50	4	50	232,4	258	2	38
90	10	200	100	90	10	1	0	234,9	261	0	93
90	10	200	100	90	10	1	25	231,3	256	9	42
90	10	200	100	90	10	1	50	231,3	257	0	27
90	10	200	100	90	10	2	0	233,4	258	12	119
90	10	200	100	90	10	2	25	234,3	257	30	158
90	10	200	100	90	10	2	50	235,2	258	30	15
90	10	200	100	90	10	4	0	232,5	258	3	24
90	10	200	100	90	10	4	25	233,1	258	9	22
90	10	200	100	90	10	4	50	231,3	257	0	15
90	10	200	100	90	50	1	0	231,6	256	12	148
90	10	200	100	90	50	1	25	231,3	256	9	28
90	10	200	100	90	50	1	50	231,3	257	0	9
90	10	200	100	90	50	2	0	231,3	257	0	58
90	10	200	100	90	50	2	25	234,3	254	57	109
90	10	200	100	90	50	2	50	234,3	254	57	23
90	10	200	100	90	50	4	0	233,1	259	0	150
90	10	200	100	90	50	4	25	232,2	258	0	160
90	10	200	100	90	50	4	50	232,4	258	2	94

ANEXO C.
DISEÑO DE EXPERIMENTOS DEL SUBALGORITMO DE PROGRAMACIÓN INICIAL

Las pruebas experimentales para la parametrización del subalgoritmo de programación inicial fueron realizadas en 3 problemas previamente seleccionados.

El problema pequeño arrojó los mismos resultados sin importar la combinación de parámetros que se utilizará, es por esto que a continuación se podrán observar los diseños de experimentos del problema mediano y el problema grande en ambos objetivos estudiados.

Los diseños de experimentos fueron realizados en el software Statgraphics Centurion.

Diseño de experimentos para la evaluación del Makespan en el Problema Mediano (Problema 7):

Tabla 69

Análisis de varianza para la evaluación del Makespan en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	334,389	2	167,194	3,09	0,0731
B:Porcentaje de elitismo	29,5556	2	14,7778	0,27	0,7642
C:Probabilidad de cruce	100,0	1	100,0	1,85	0,1926
D:Probabilidad de mutación	484,0	1	484,0	8,96	0,0086
INTERACTIONS					
AB	696,778	4	174,194	3,22	0,0404
AC	360,167	2	180,083	3,33	0,0617
AD	38,1667	2	19,0833	0,35	0,7078
BC	266,0	2	133,0	2,46	0,1169
BD	152,667	2	76,3333	1,41	0,2723
CD	9,0	1	9,0	0,17	0,6886
RESIDUAL	864,5	16	54,0313		
TOTAL (CORRECTED)	3335,22	35			

El factor probabilidad de mutación y la interacción de los factores intensidad de mutación y porcentaje de elitismo tienen efectos significativos sobre el makespan del problema.

Tabla 70

Análisis de varianza mejorado para la evaluación del Makespan en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	334,389	2	167,194	2,03	0,1494
B:Porcentaje de elitismo	29,5556	2	14,7778	0,18	0,8366
C:Probabilidad de cruce	100,0	1	100,0	1,21	0,2795
D:Probabilidad de mutación	484,0	1	484,0	5,88	0,0218
INTERACTIONS					
RESIDUAL	2387,28	29	82,3199		
TOTAL (CORRECTED)	3335,22	35			

Sin embargo, después de mejorar el análisis de varianza el único factor que tiene efectos significativos sobre el objetivo evaluado es la probabilidad de mutación.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 71

Tabla de medias de los resultados obtenidos para evaluación del Makespan en el problema mediano (Problema 7)

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	1009,28			
Intensidad de mutación					
1	12	1013,42	2,61916	1008,06	1018,77
2	12	1006,17	2,61916	1000,81	1011,52
4	12	1008,25	2,61916	1002,89	1013,61
Porcentaje de elitismo					
0	12	1010,0	2,61916	1004,64	1015,36
25	12	1009,83	2,61916	1004,48	1015,19
50	12	1008,0	2,61916	1002,64	1013,36
Probabilidad de cruce					
60	18	1007,61	2,13853	1003,24	1011,98
90	18	1010,94	2,13853	1006,57	1015,32
Probabilidad de mutación					
10	18	1012,94	2,13853	1008,57	1017,32
50	18	1005,61	2,13853	1001,24	1009,98

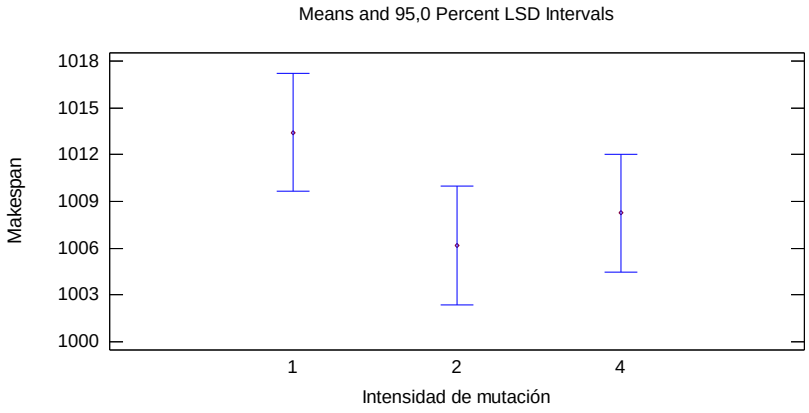


Figura 29. Gráfica de medias para el factor intensidad de mutación para la evaluación del Makespan del problema mediano (Problema 7).

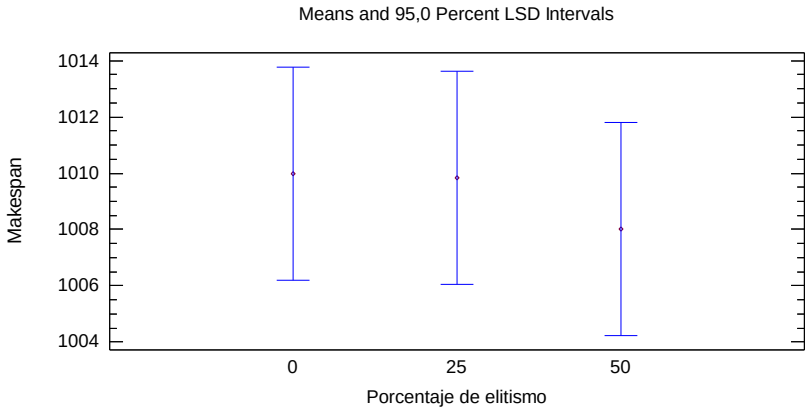


Figura 30. Gráfica de medias para el factor porcentaje de elitismo para la evaluación del Makespan del problema mediano (Problema 7).

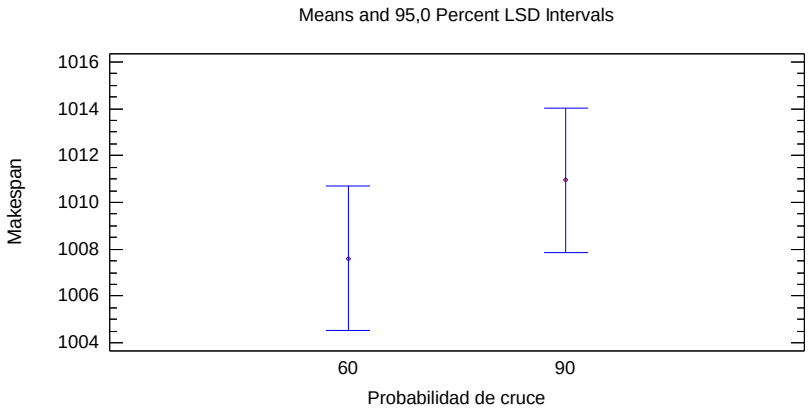


Figura 31. Gráfica de medias para el factor probabilidad de cruce para la evaluación del Makespan del problema mediano (Problema 7).

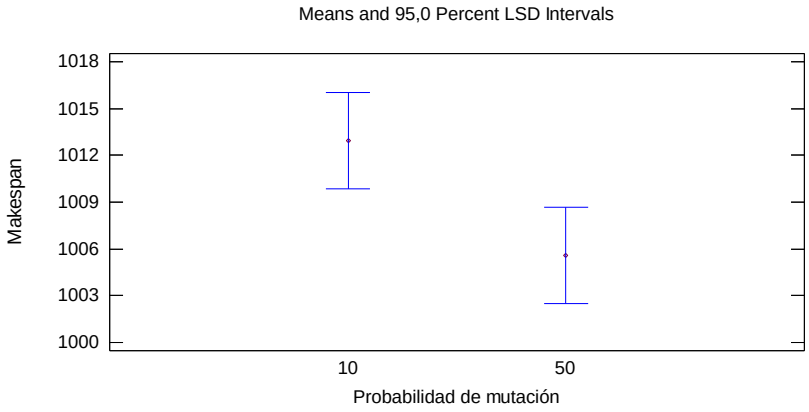


Figura 32. Gráfica de medias para el factor probabilidad de mutación para la evaluación del Makespan del problema mediano (Problema 7).

A continuación se muestra la validación de los supuestos:

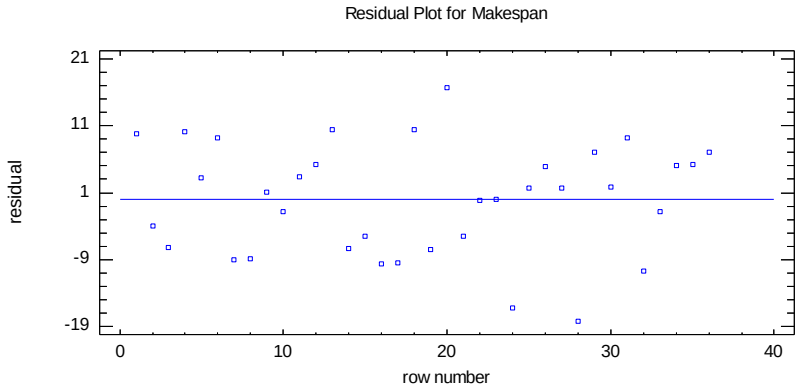


Figura 33. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación del Makespan del problema mediano (Problema 7).

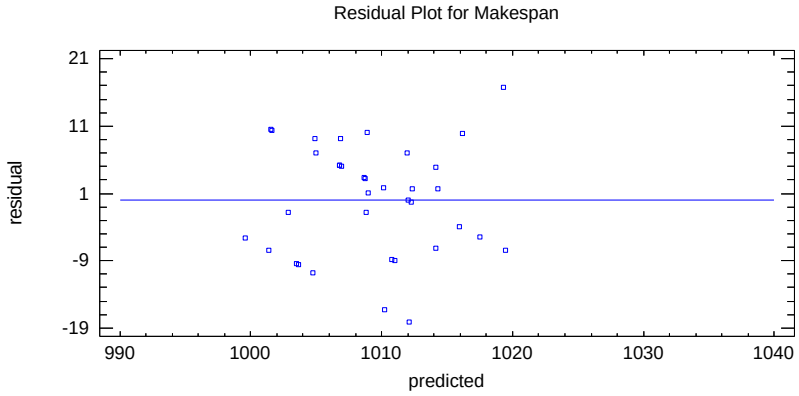


Figura 34. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación del Makespan del problema mediano (Problema 7).

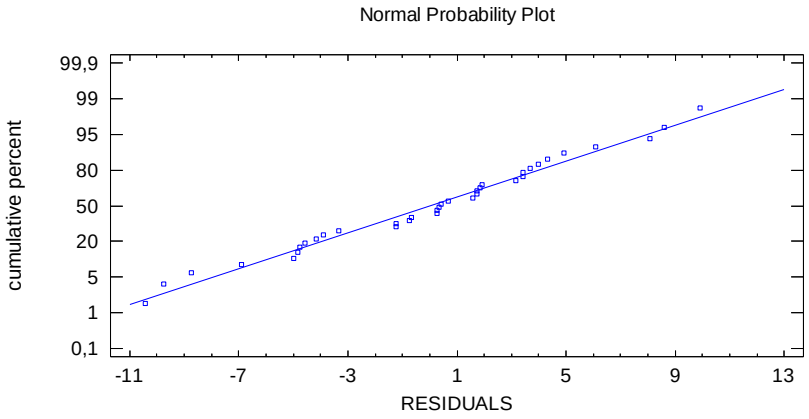


Figura 35. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación del Makespan del problema mediano (Problema 7).

Diseño de experimentos para la evaluación del Makespan en el Problema Grande (Problema 9):

Tabla 72

Análisis de varianza para la evaluación del Makespan en el problema grande (Problema 9)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	9,05556	2	4,52778	1,56	0,2409
B:Porcentaje de elitismo	0,388889	2	0,194444	0,07	0,9355
C:Probabilidad de cruce	13,4444	1	13,4444	4,63	0,0471
D:Probabilidad de mutación	16,0	1	16,0	5,51	0,0322
INTERACTIONS					
AB	23,6111	4	5,90278	2,03	0,1383
AC	13,3889	2	6,69444	2,30	0,1321
AD	3,5	2	1,75	0,60	0,5596
BC	1,05556	2	0,527778	0,18	0,8356
BD	4,5	2	2,25	0,77	0,4776
CD	11,1111	1	11,1111	3,82	0,0682
RESIDUAL	46,5	16	2,90625		
TOTAL (CORRECTED)	142,556	35			

Los factores probabilidad de cruce y probabilidad de mutación tienen efectos significativos sobre el makespan del problema.

Tabla 73

Análisis de varianza mejorado para la evaluación del Makespan en el problema grande (Problema 9)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	9,05556	2	4,52778	1,27	0,2969
B:Porcentaje de elitismo	0,388889	2	0,194444	0,05	0,9472
C:Probabilidad de cruce	13,4444	1	13,4444	3,76	0,0622
D:Probabilidad de mutación	16,0	1	16,0	4,48	0,0431
INTERACTIONS					
RESIDUAL	103,667	29	3,57471		
TOTAL (CORRECTED)	142,556	35			

Sin embargo, después de mejorar el análisis de varianza solo la probabilidad de mutación tienen efectos significativos sobre dicho objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 74

Tabla de medias de los resultados obtenidos para evaluación del Makespan en el problema grande (Problema 9)

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	252,389			
Intensidad de mutación					
1	12	253,083	0,545795	251,967	254,2
2	12	251,917	0,545795	250,8	253,033
4	12	252,167	0,545795	251,05	253,283
Porcentaje de elitismo					
0	12	252,25	0,545795	251,134	253,366
25	12	252,417	0,545795	251,3	253,533
50	12	252,5	0,545795	251,384	253,616
Probabilidad de cruce					
60	18	251,778	0,44564	250,866	252,689
90	18	253,0	0,44564	252,089	253,911
Probabilidad de mutación					
10	18	253,056	0,44564	252,144	253,967
50	18	251,722	0,44564	250,811	252,634

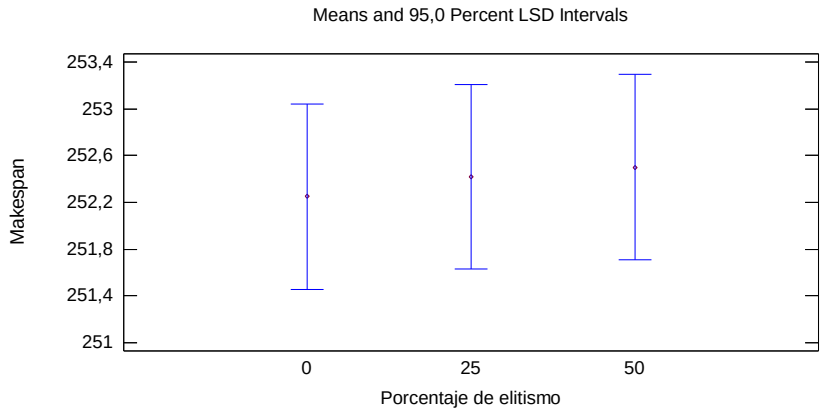


Figura 36. Gráfica de medias para el factor porcentaje de elitismo para la evaluación del Makespan del problema grande (Problema 9).

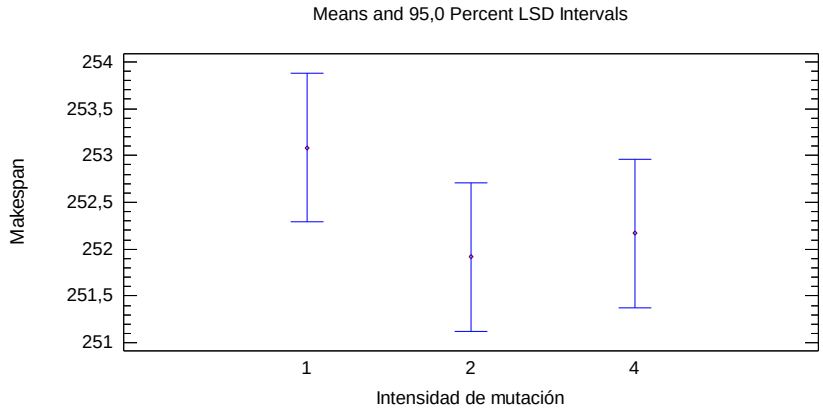


Figura 37. Gráfica de medias para el factor intensidad de mutación para la evaluación del Makespan del problema grande (Problema 9).

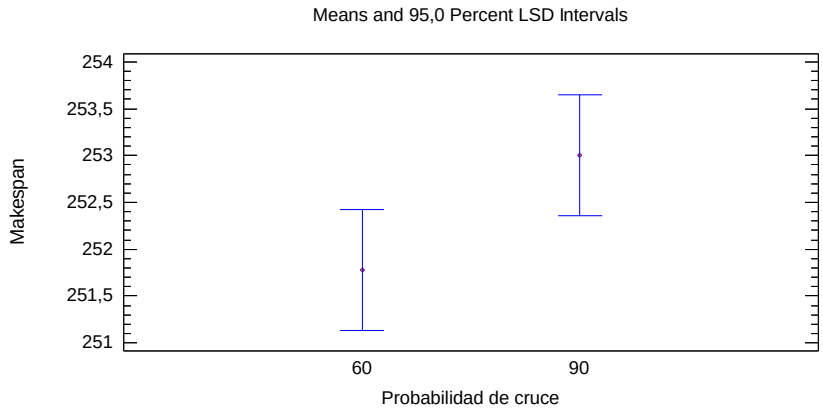


Figura 38. Gráfica de medias para el factor probabilidad de cruce para la evaluación del Makespan del problema grande (Problema 9).

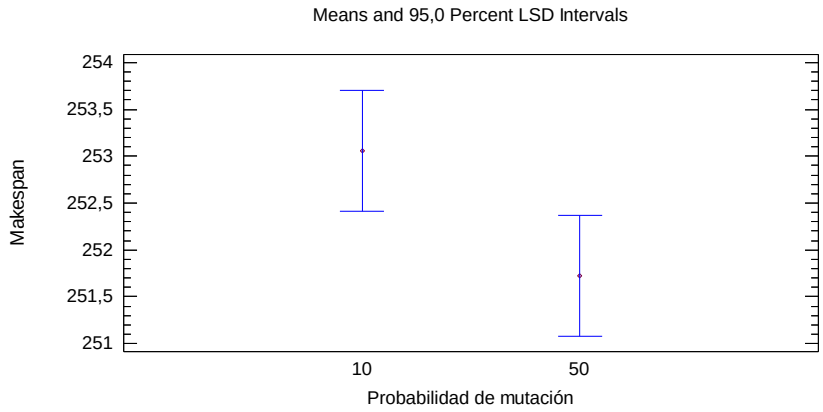


Figura 39. Gráfica de medias para el factor probabilidad de mutación para la evaluación del Makespan del problema grande (Problema 9).

A continuación se muestra la validación de supuestos:

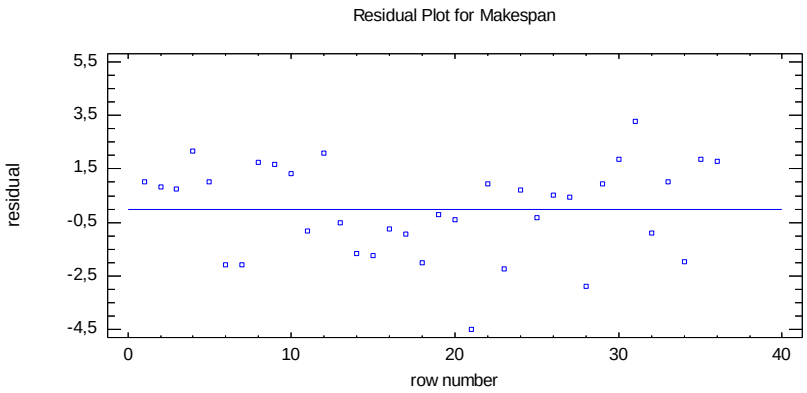


Figura 40. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación del Makespan del problema grande (Problema 9).

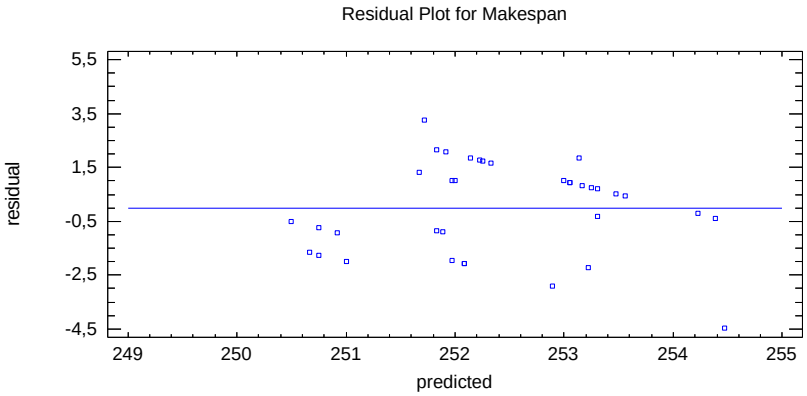


Figura 41. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación del Makespan del problema grande (Problema 9).

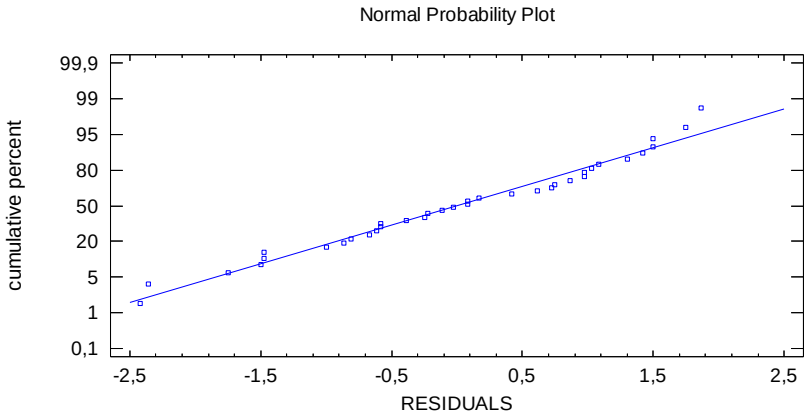


Figura 42. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación del Makespan del problema grande (Problema 9).

Diseño de experimentos para la evaluación de la Tardanza Total Ponderada en el Problema Mediano (Problema 7):

Tabla 75

Análisis de varianza para la evaluación de la Tardanza Total Ponderada en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	38848,2	2	19424,1	2,02	0,1649
B:Porcentaje de elitismo	155545,	2	77772,3	8,10	0,0037
C:Probabilidad de cruce	2025,0	1	2025,0	0,21	0,6523
D:Probabilidad de mutación	13301,8	1	13301,8	1,38	0,2565
INTERACTIONS					
AB	13054,7	4	3263,67	0,34	0,8472
AC	1331,17	2	665,583	0,07	0,9333
AD	43220,7	2	21610,4	2,25	0,1378
BC	2696,0	2	1348,0	0,14	0,8701
BD	1336,22	2	668,111	0,07	0,9331
CD	10133,8	1	10133,8	1,05	0,3197
RESIDUAL	153711,	16	9606,93		
TOTAL (CORRECTED)	435203,	35			

Después de realizar el análisis de varianza se puede notar que el único factor que tiene efectos significativos sobre la tardanza total ponderada de éste problema es el porcentaje de elitismo.

Tabla 76

Análisis de varianza mejorado para la evaluación de la Tardanza Total Ponderada en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	38848,2	2	19424,1	2,50	0,0998
B:Porcentaje de elitismo	155545,	2	77772,3	10,00	0,0005
C:Probabilidad de cruce	2025,0	1	2025,0	0,26	0,6137
D:Probabilidad de mutación	13301,8	1	13301,8	1,71	0,2012
INTERACTIONS					
RESIDUAL	225483,	29	7775,29		
TOTAL (CORRECTED)	435203,	35			

Al mejorar el análisis de varianza se puede observar que el porcentaje de elitismo sigue siendo el único factor con efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 77

Tabla de medias de los resultados obtenidos para evaluación de la Tardanza Total Ponderada en el problema mediano (Problema 7)

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	630,167			
Intensidad de mutación					
1	12	635,583	25,4547	583,523	687,644
2	12	667,417	25,4547	615,356	719,477
4	12	587,5	25,4547	535,439	639,561
Porcentaje de elitismo					
0	12	720,333	25,4547	668,273	772,394
25	12	604,667	25,4547	552,606	656,727
50	12	565,5	25,4547	513,439	617,561
Probabilidad de cruce					
60	18	637,667	20,7837	595,159	680,174
90	18	622,667	20,7837	580,159	665,174
Probabilidad de mutación					
10	18	649,389	20,7837	606,881	691,896
50	18	610,944	20,7837	568,437	653,452

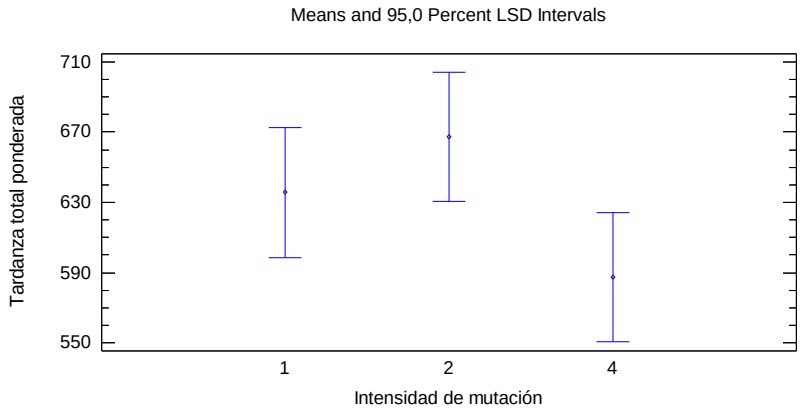


Figura 43. Gráfica de medias para el factor intensidad de mutación para la evaluación de la Tardanza Total Ponderada del problema mediano (Problema 7).

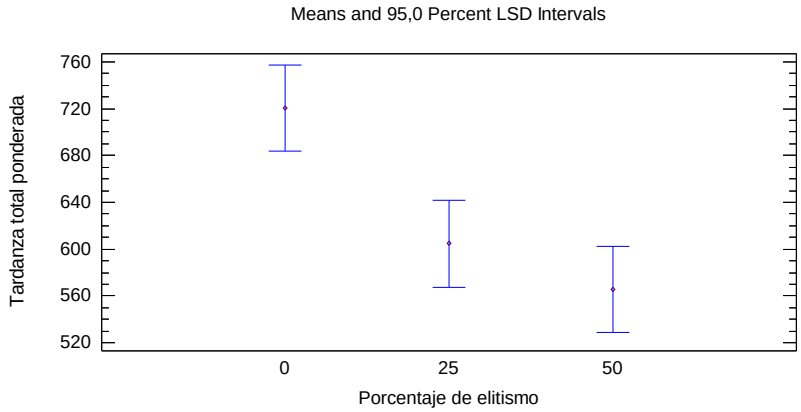


Figura 44. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la Tardanza Total Ponderada del problema mediano (Problema 7).

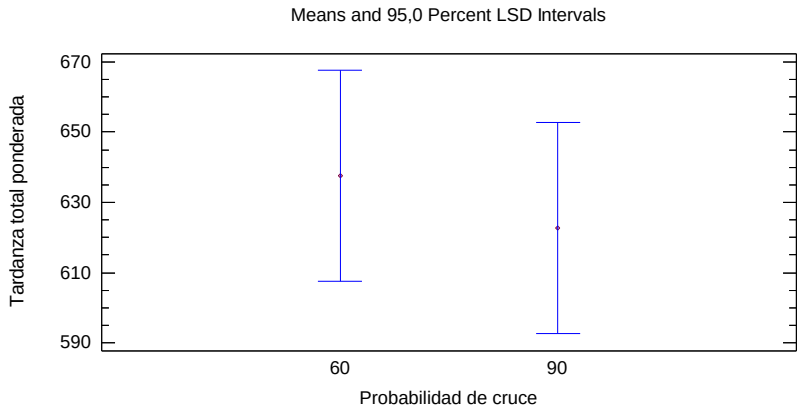


Figura 45. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la Tardanza Total Ponderada del problema mediano (Problema 7).

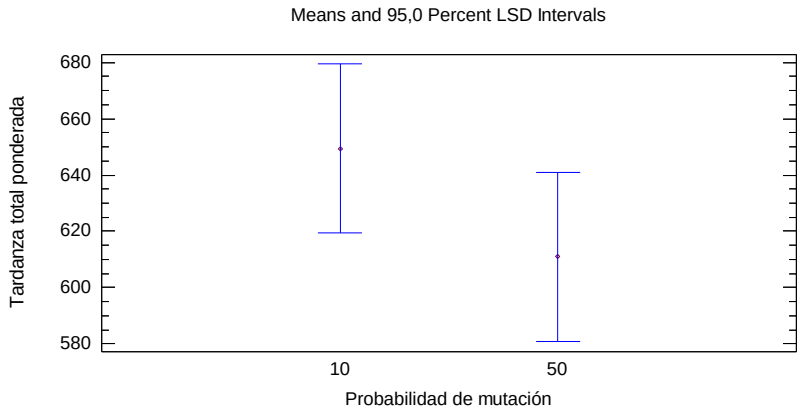


Figura 46. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la Tardanza Total Ponderada del problema mediano (Problema 7).

A continuación se muestra la validación de supuestos:

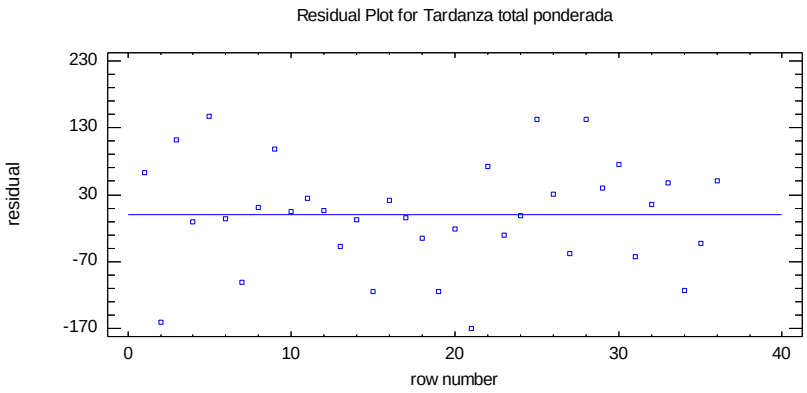


Figura 47. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la Tardanza Total Ponderada del problema mediano (Problema 7).

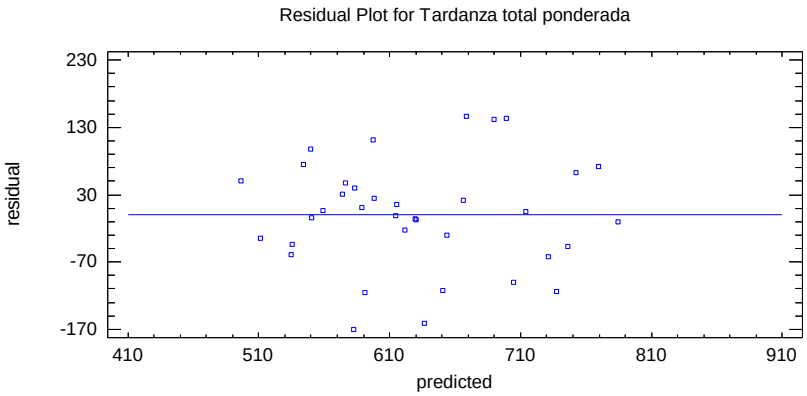


Figura 48. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la Tardanza Total Ponderada del problema mediano (Problema 7).

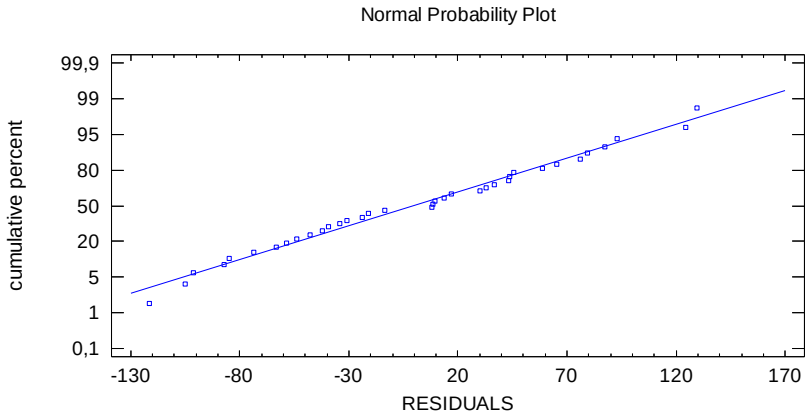


Figura 49. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la Tardanza Total Ponderada del problema mediano (Problema 7).

Diseño de experimentos para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el Problema Mediano (Problema 7):

Tabla 78

Análisis de varianza para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	21699,6	2	10849,8	2,21	0,1418
B:Porcentaje de elitismo	47719,8	2	23859,9	4,87	0,0224
C:Probabilidad de cruce	78,9136	1	78,9136	0,02	0,9006
D:Probabilidad de mutación	1107,78	1	1107,78	0,23	0,6410
INTERACTIONS					
AB	9578,78	4	2394,7	0,49	0,7443
AC	8814,06	2	4407,03	0,90	0,4267
AD	4682,96	2	2341,48	0,48	0,6289
BC	1100,23	2	550,117	0,11	0,8946
BD	2411,55	2	1205,78	0,25	0,7849
CD	3698,67	1	3698,67	0,75	0,3980
RESIDUAL	78467,7	16	4904,23		
TOTAL (CORRECTED)	179360,	35			

En el análisis de varianza se puede notar que el único factor que tiene efectos significativos sobre el objetivo ponderado de éste problema es el porcentaje de elitismo.

Tabla 79

Análisis de varianza mejorado para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	21699,6	2	10849,8	2,89	0,0715
B:Porcentaje de elitismo	47719,8	2	23859,9	6,36	0,0051
C:Probabilidad de cruce	78,9136	1	78,9136	0,02	0,8857
D:Probabilidad de mutación	1107,78	1	1107,78	0,30	0,5909
INTERACTIONS					
RESIDUAL	108754,	29	3750,14		
TOTAL (CORRECTED)	179360,	35			

Al mejorar el análisis de varianza se puede observar que el porcentaje de elitismo sigue siendo el único factor con efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 80

Tabla de medias de los resultados obtenidos para evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	676,803			
Intensidad de mutación					
1	12	701,358	17,678	665,203	737,514
2	12	685,783	17,678	649,628	721,939
4	12	643,267	17,678	607,111	679,422
Porcentaje de elitismo					
0	12	728,233	17,678	692,078	764,389
25	12	648,967	17,678	612,811	685,122
50	12	653,208	17,678	617,053	689,364
Probabilidad de cruce					
60	18	675,322	14,434	645,801	704,843
90	18	678,283	14,434	648,762	707,804
Probabilidad de mutación					
10	18	682,35	14,434	652,829	711,871
50	18	671,256	14,434	641,735	700,777

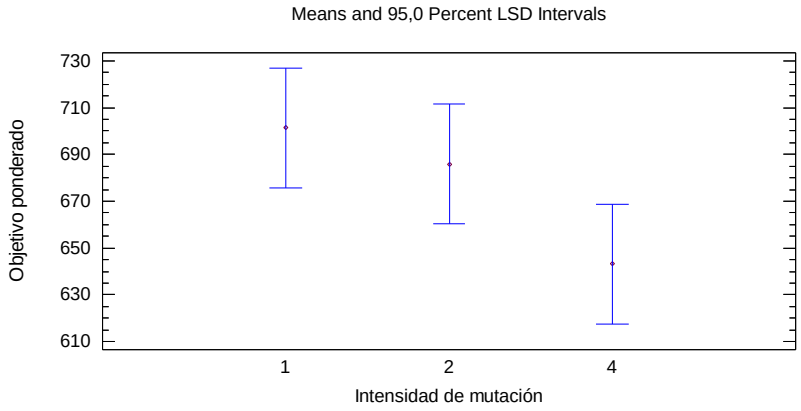


Figura 50. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

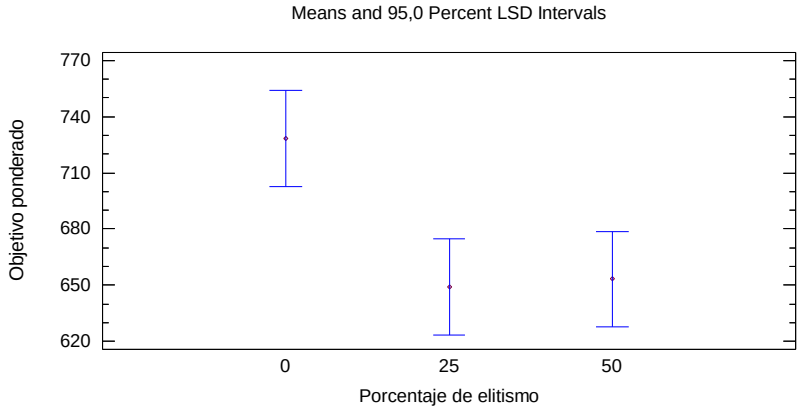


Figura 51. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

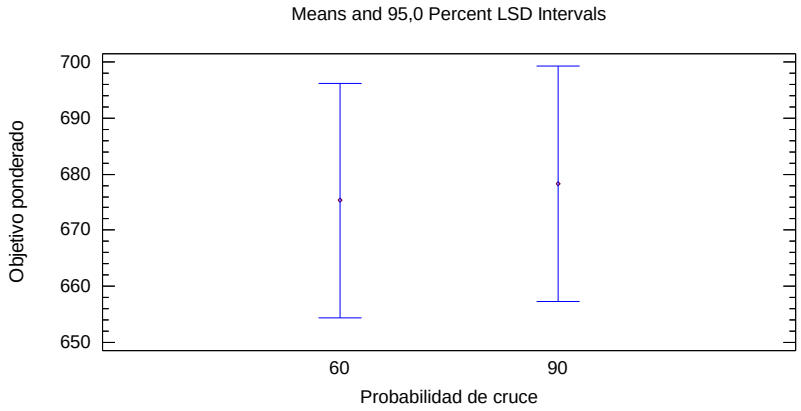


Figura 52. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

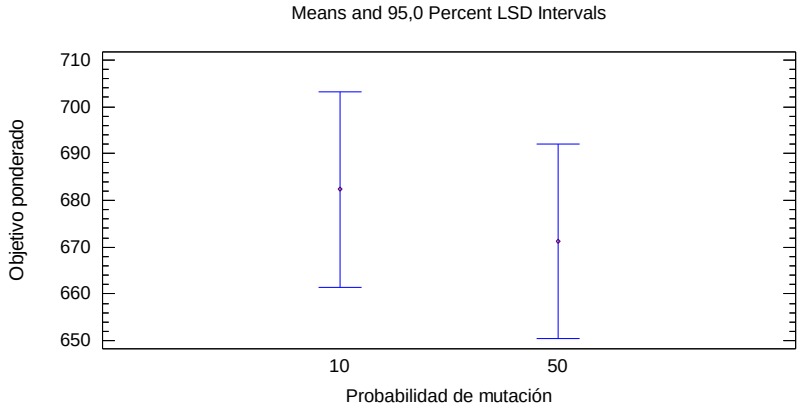


Figura 53. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

A continuación se muestra la validación de supuestos:

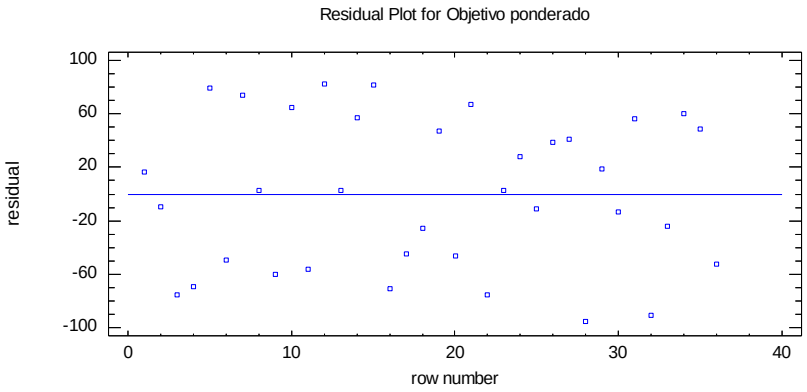


Figura 54. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

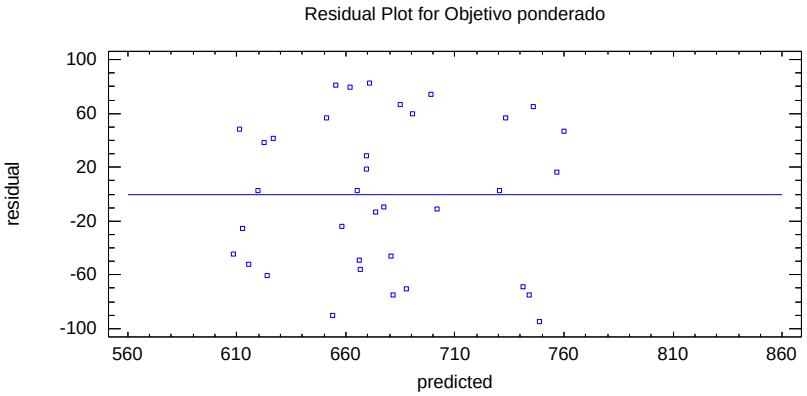


Figura 55. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

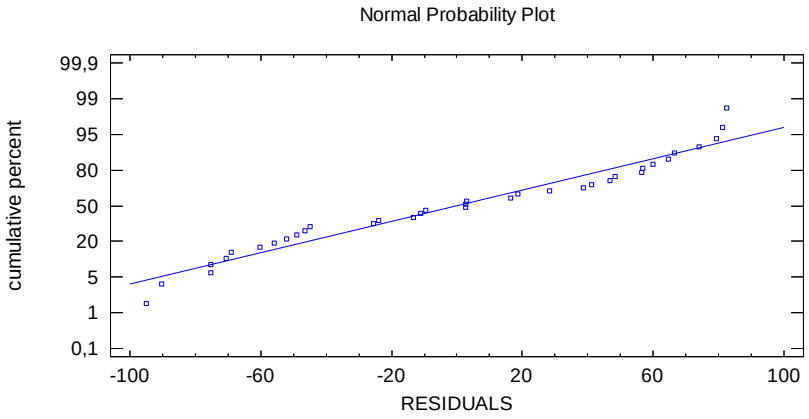


Figura 56. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha\text{Makespan} + \beta\text{TTP})$ en el problema mediano (Problema 7).

Diseño de experimentos para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha\text{Makespan} + \beta\text{TTP})$ en el problema mediano (Problema 7):

Tabla 81

Análisis de varianza para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	31652,5	2	15826,3	5,10	0,0193
B:Porcentaje de elitismo	24855,5	2	12427,8	4,01	0,0389
C:Probabilidad de cruce	2584,03	1	2584,03	0,83	0,3750
D:Probabilidad de mutación	11200,7	1	11200,7	3,61	0,0756
INTERACTIONS					
AB	894,993	4	223,748	0,07	0,9896
AC	6328,81	2	3164,4	1,02	0,3829
AD	1968,54	2	984,271	0,32	0,7326
BC	5860,95	2	2930,47	0,94	0,4095
BD	7862,72	2	3931,36	1,27	0,3084
CD	13774,9	1	13774,9	4,44	0,0512
RESIDUAL	49641,9	16	3102,62		
TOTAL (CORRECTED)	156626,	35			

En el análisis de varianza se puede notar que los factores que tienen efectos significativos sobre el objetivo ponderado de éste problema son la intensidad de mutación y el porcentaje de elitismo.

Tabla 82

Análisis de varianza mejorado para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	31652,5	2	15826,3	5,32	0,0108
B:Porcentaje de elitismo	24855,5	2	12427,8	4,17	0,0255
C:Probabilidad de cruce	2584,03	1	2584,03	0,87	0,3592
D:Probabilidad de mutación	11200,7	1	11200,7	3,76	0,0622
INTERACTIONS					
RESIDUAL	86332,9	29	2977,0		
TOTAL (CORRECTED)	156626,	35			

Después de mejorar el análisis de varianza los factores que siguen teniendo efectos significativos son los mismos, la intensidad de mutación y el porcentaje de elitismo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 83

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	716,083			
Intensidad de mutación					
1	12	750,75	15,7506	718,536	782,964
2	12	719,183	15,7506	686,97	751,397
4	12	678,317	15,7506	646,103	710,53
Porcentaje de elitismo					
0	12	752,983	15,7506	720,77	785,197
25	12	693,833	15,7506	661,62	726,047
50	12	701,433	15,7506	669,22	733,647
Probabilidad de cruce					
60	18	724,556	12,8604	698,253	750,858
90	18	707,611	12,8604	681,309	733,914
Probabilidad de mutación					
10	18	733,722	12,8604	707,42	760,025
50	18	698,444	12,8604	672,142	724,747

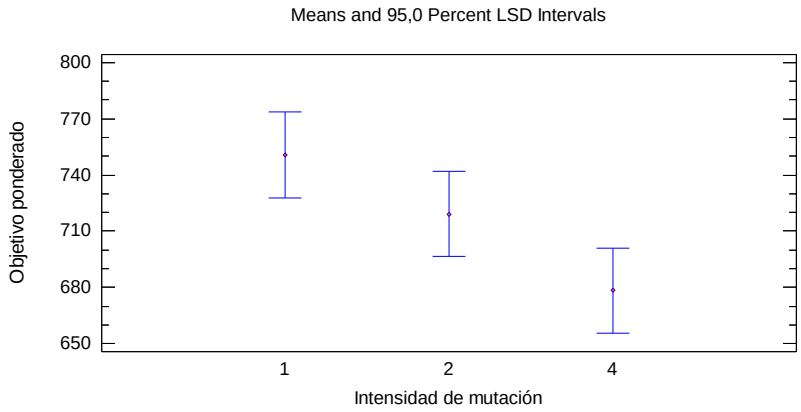


Figura 57. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha \mathrm{Makespan} + \beta \mathrm{TTP})$ en el problema mediano (Problema 7).

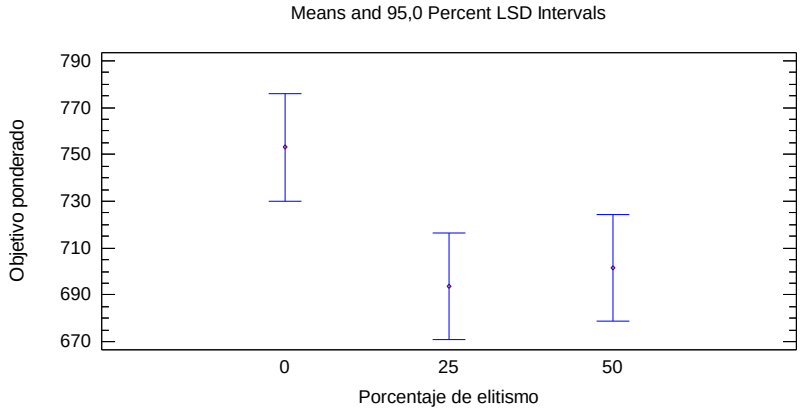


Figura 58. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha \mathrm{Makespan} + \beta \mathrm{TTP})$ en el problema mediano (Problema 7).

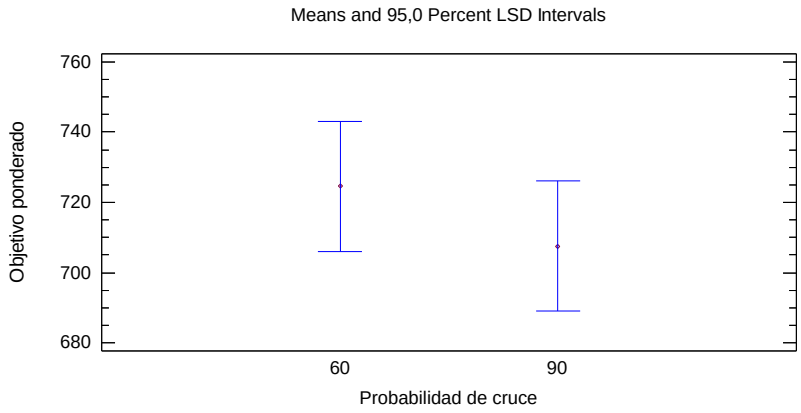


Figura 59. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha \mathrm{Makespan} + \beta \mathrm{TTP})$ en el problema mediano (Problema 7).

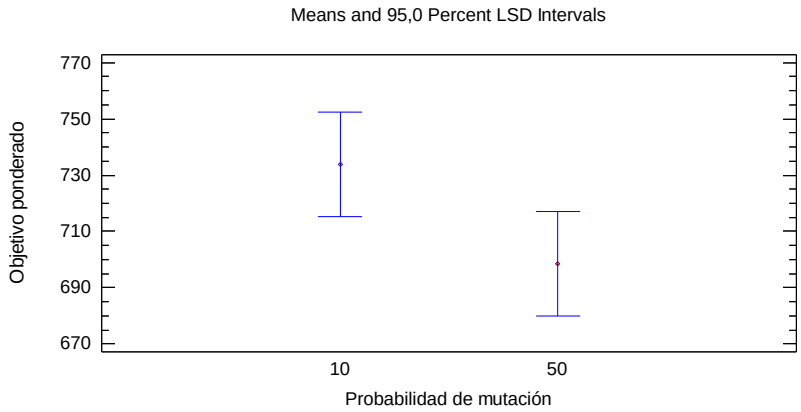


Figura 60. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha \mathrm{Makespan} + \beta \mathrm{TTP})$ en el problema mediano (Problema 7).

A continuación se muestra la validación de supuestos:

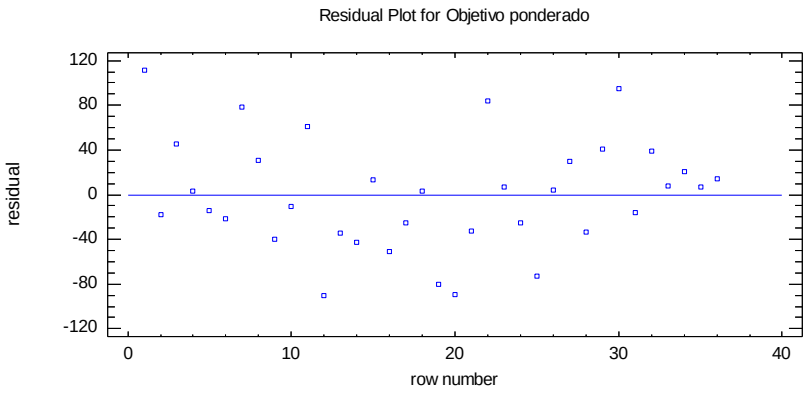


Figura 61. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

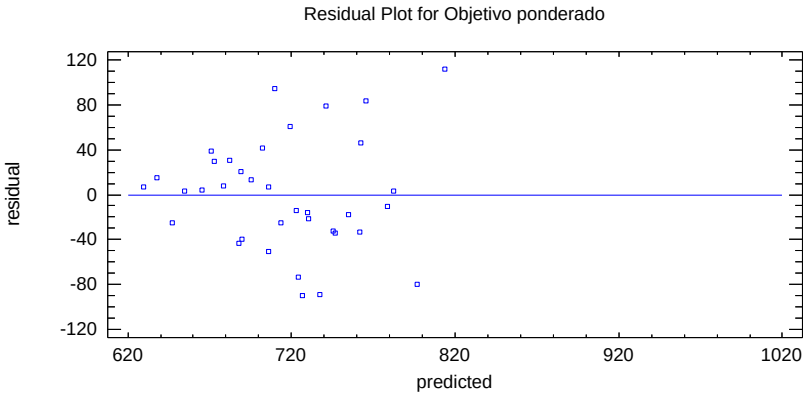


Figura 62. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

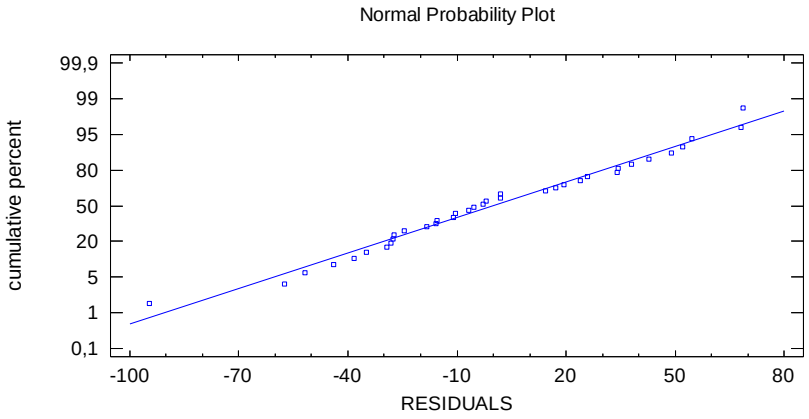


Figura 63. Gráfica de probabilidad normal para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado (αMakespan + βTTP) en el problema mediano (Problema 7).

Diseño de experimentos para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado (αMakespan + βTTP) en el problema mediano (Problema 7):

Tabla 84

Análisis de varianza para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado ($\alpha Makespan + \beta TTP$) en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	970,772	2	485,386	0,26	0,7738
B:Porcentaje de elitismo	69923,0	2	34961,5	18,77	0,0001
C:Probabilidad de cruce	798,063	1	798,063	0,43	0,5220
D:Probabilidad de mutación	10455,1	1	10455,1	5,61	0,0307
INTERACTIONS					
AB	571,367	4	142,842	0,08	0,9883
AC	386,165	2	193,083	0,10	0,9021
AD	6102,0	2	3051,0	1,64	0,2253
BC	823,76	2	411,88	0,22	0,8040
BD	5792,45	2	2896,22	1,55	0,2415
CD	1318,9	1	1318,9	0,71	0,4125
RESIDUAL	29800,7	16	1862,54		
TOTAL (CORRECTED)	126942,	35			

En el análisis de varianza se puede observar que los factores que tienen efectos significativos sobre el objetivo ponderado de éste problema son el porcentaje de elitismo y la probabilidad de mutación.

Tabla 85

Análisis de varianza mejorado para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	970,772	2	485,386	0,31	0,7328
B:Porcentaje de elitismo	69923,0	2	34961,5	22,63	0,0000
C:Probabilidad de cruce	798,062	1	798,062	0,52	0,4780
D:Probabilidad de mutación	10455,1	1	10455,1	6,77	0,0145
INTERACTIONS					
RESIDUAL	44795,3	29	1544,67		
TOTAL (CORRECTED)	126942,	35			

Después de mejorar el análisis de varianza los factores que siguen teniendo efectos significativos son los mismos, el porcentaje de elitismo y la probabilidad de mutación.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 86

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	773,158			
Intensidad de mutación					
1	12	774,708	11,3456	751,504	797,913
2	12	778,6	11,3456	755,396	801,804
4	12	766,167	11,3456	742,962	789,371
Porcentaje de elitismo					
0	12	835,342	11,3456	812,137	858,546
25	12	738,408	11,3456	715,204	761,613
50	12	745,725	11,3456	722,521	768,929
Probabilidad de cruce					
60	18	768,45	9,26363	749,504	787,396
90	18	777,867	9,26363	758,92	796,813
Probabilidad de mutación					
10	18	790,2	9,26363	771,254	809,146

50	18	756,117	9,26363	737,17	775,063

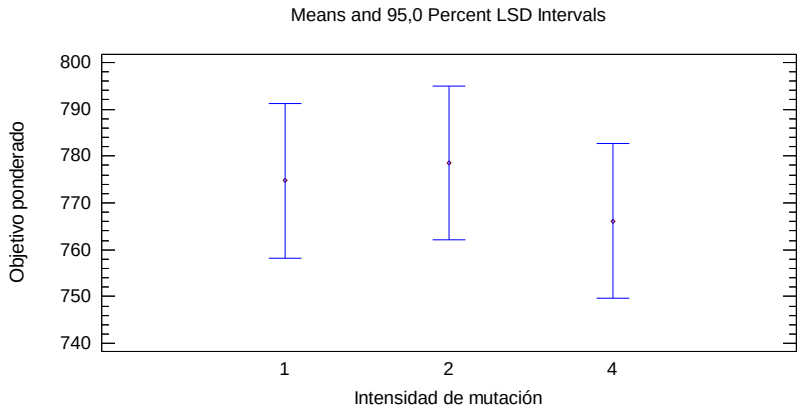


Figura 64. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

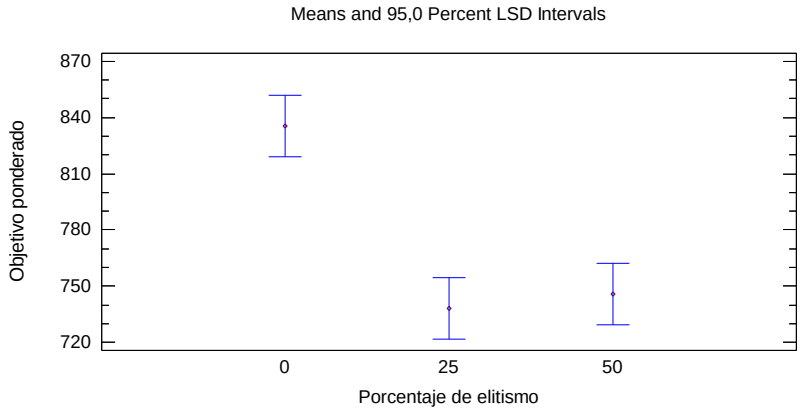


Figura 65. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

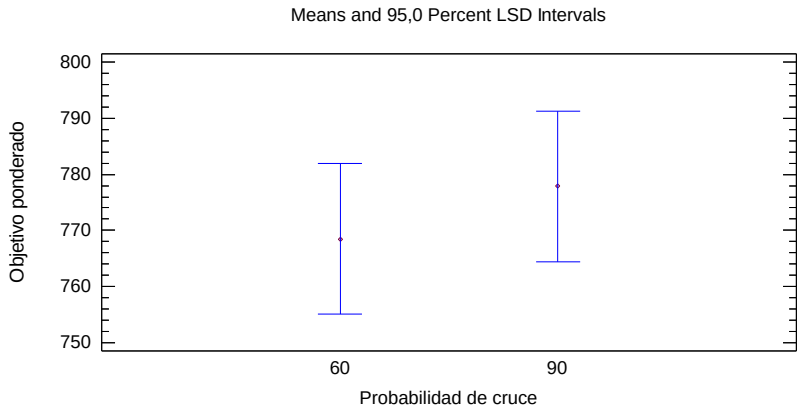


Figura 66. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

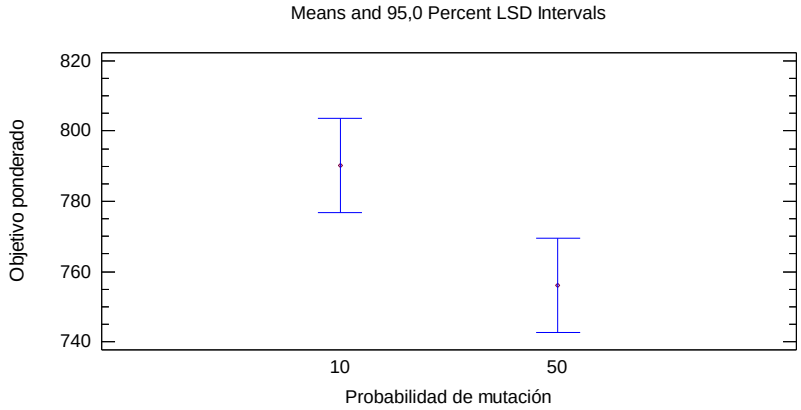


Figura 67. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

A continuación se muestra la validación de supuestos:

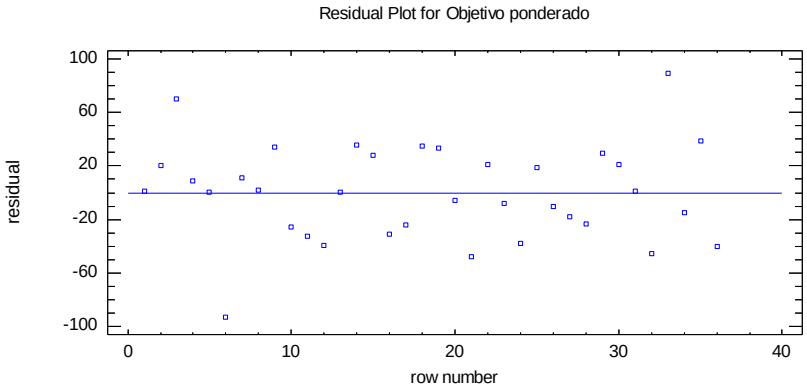


Figura 68. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado (αMakespan + βTTP) en el problema mediano (Problema 7).

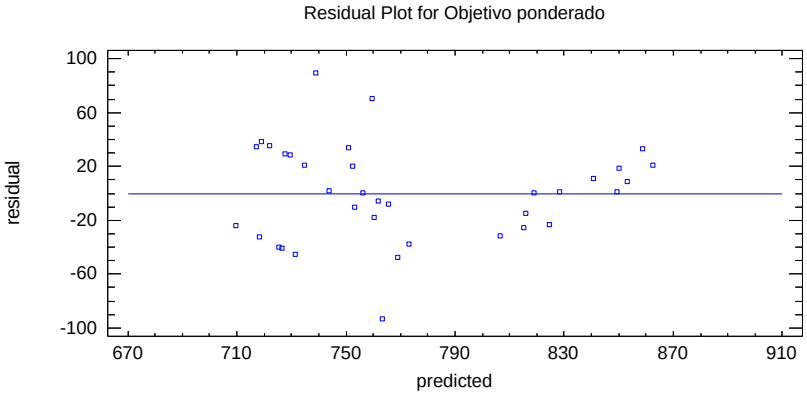


Figura 69. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado (αMakespan + βTTP) en el problema mediano (Problema 7).

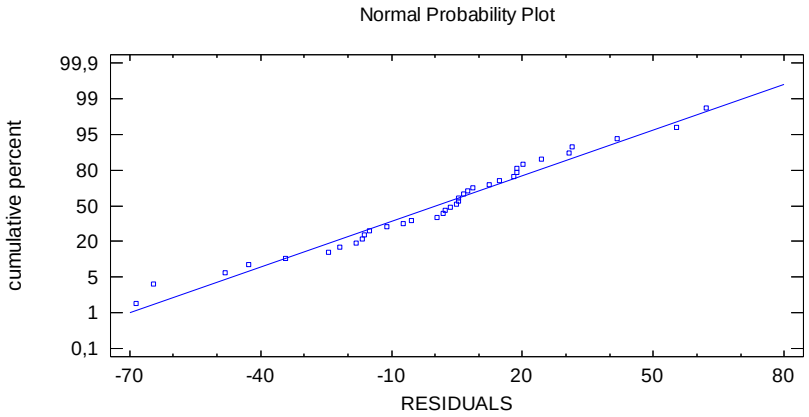


Figura 70. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\alpha\text{Makespan} + \beta\text{TTP})$ en el problema mediano (Problema 7).

Diseño de experimentos para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado $(\alpha\text{Makespan} + \beta\text{TTP})$ en el problema mediano (Problema 7):

Tabla 87

Análisis de varianza para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	683,962	2	341,981	0,38	0,6920
B:Porcentaje de elitismo	8784,7	2	4392,35	4,84	0,0227
C:Probabilidad de cruce	8661,4	1	8661,4	9,54	0,0070
D:Probabilidad de mutación	2473,4	1	2473,4	2,73	0,1183
INTERACTIONS					
AB	5347,95	4	1336,99	1,47	0,2566
AC	881,602	2	440,801	0,49	0,6240
AD	4201,42	2	2100,71	2,31	0,1309
BC	1707,58	2	853,791	0,94	0,4109
BD	3756,92	2	1878,46	2,07	0,1587
CD	366,084	1	366,084	0,40	0,5343
RESIDUAL	14520,6	16	907,536		
TOTAL (CORRECTED)	51385,6	35			

En el análisis de varianza se puede observar que los factores que tienen efectos significativos sobre el objetivo ponderado de éste problema son el porcentaje de elitismo y la probabilidad de cruce.

Tabla 88

Análisis de varianza mejorado para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	683,962	2	341,981	0,32	0,7271
B:Porcentaje de elitismo	8784,7	2	4392,35	4,14	0,0262
C:Probabilidad de cruce	8661,4	1	8661,4	8,16	0,0078
D:Probabilidad de mutación	2473,4	1	2473,4	2,33	0,1377
INTERACTIONS					
RESIDUAL	30782,1	29	1061,45		
TOTAL (CORRECTED)	51385,6	35			

Después de mejorar el análisis de varianza los factores que siguen teniendo efectos significativos son los mismos, el porcentaje de elitismo y la probabilidad de cruce.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 89

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	796,011			
Intensidad de mutación					
1	12	801,05	9,40502	781,815	820,285
2	12	790,417	9,40502	771,181	809,652
4	12	796,567	9,40502	777,331	815,802
Porcentaje de elitismo					
0	12	818,1	9,40502	798,865	837,335
25	12	784,667	9,40502	765,431	803,902
50	12	785,267	9,40502	766,031	804,502
Probabilidad de cruce					
60	18	811,522	7,67917	795,817	827,228
90	18	780,5	7,67917	764,794	796,206
Probabilidad de mutación					
10	18	804,3	7,67917	788,594	820,006
50	18	787,722	7,67917	772,017	803,428

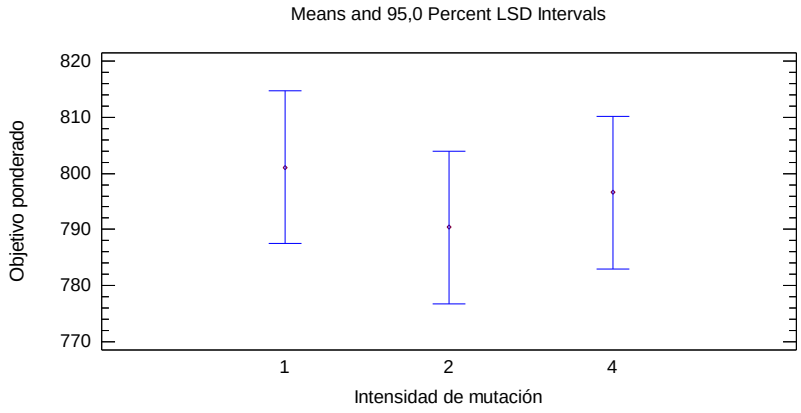


Figura 71. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado ($\alpha \mathrm{Makespan} + \beta \mathrm{TTP}$) en el problema mediano (Problema 7).

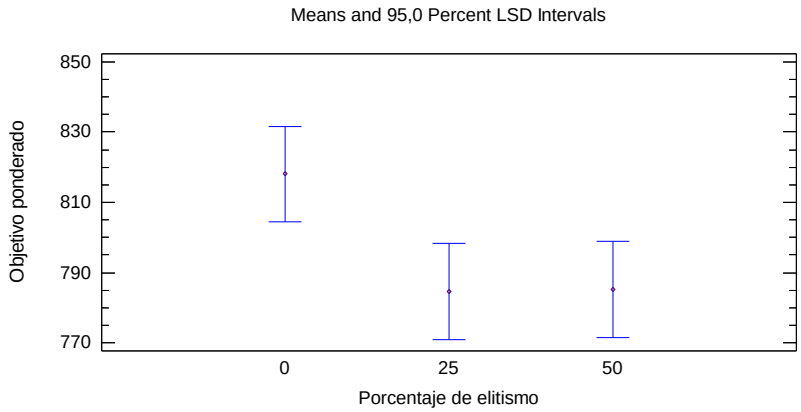


Figura 72. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado ($\alpha \mathrm{Makespan} + \beta \mathrm{TTP}$) en el problema mediano (Problema 7).

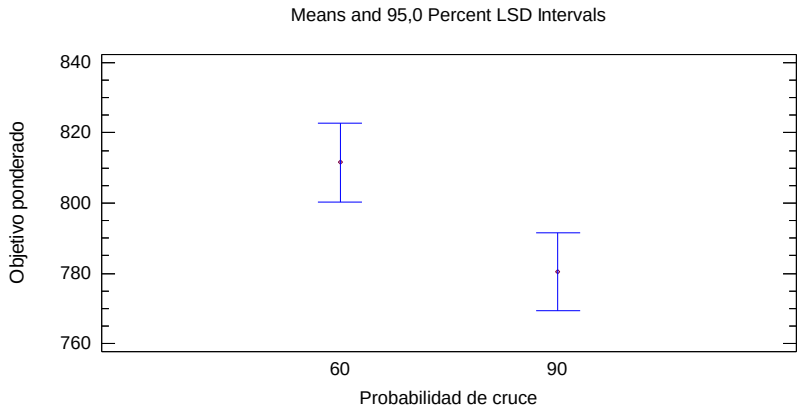


Figura 73. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

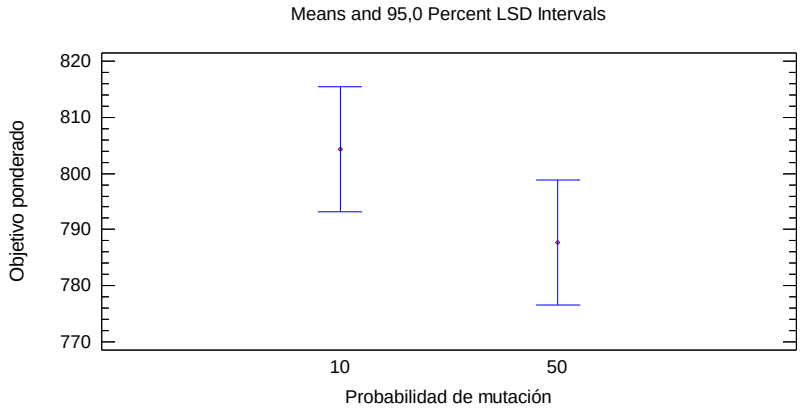


Figura 74. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

A continuación se muestra la validación de supuestos:

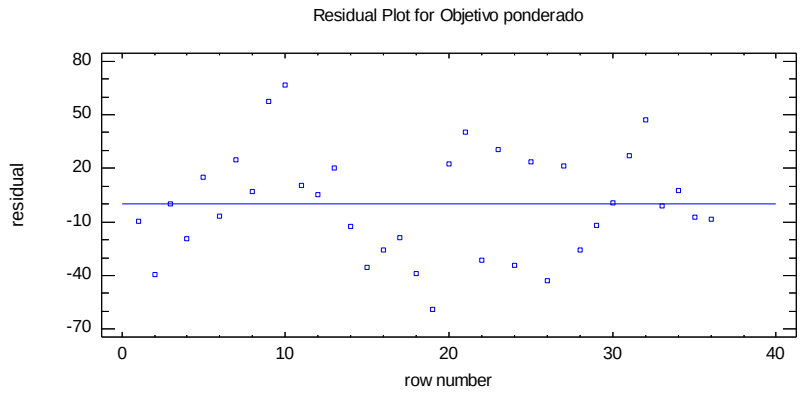


Figura 75. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad mutación para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

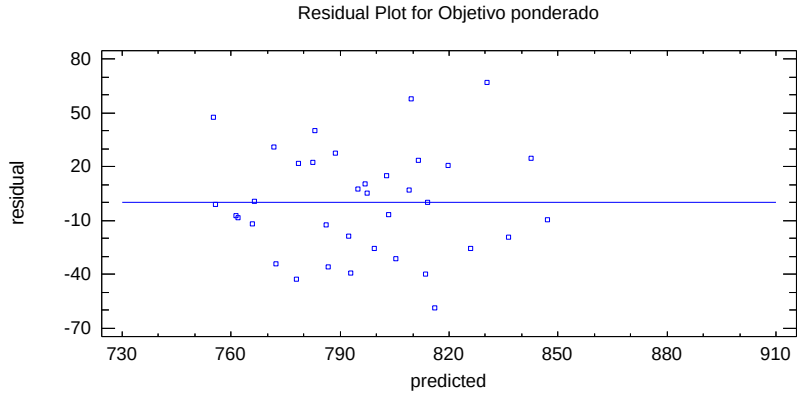


Figura 76. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

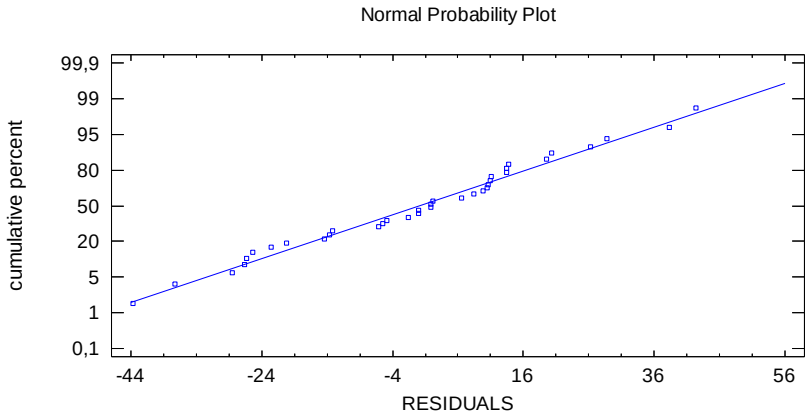


Figura 77. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado $(\alpha\text{Makespan} + \beta\text{TTP})$ en el problema mediano (Problema 7).

Diseño de experimentos para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado $(\alpha\text{Makespan} + \beta\text{TTP})$ en el problema mediano (Problema 7):

Tabla 90

Análisis de varianza para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	4318,17	2	2159,08	2,29	0,1336
B:Porcentaje de elitismo	7806,5	2	3903,25	4,14	0,0356
C:Probabilidad de cruce	52,5625	1	52,5625	0,06	0,8164
D:Probabilidad de mutación	47,8403	1	47,8403	0,05	0,8247
INTERACTIONS					
AB	3096,21	4	774,052	0,82	0,5308
AC	1168,67	2	584,333	0,62	0,5506
AD	1289,39	2	644,694	0,68	0,5190
BC	1971,17	2	985,583	1,04	0,3745
BD	3814,06	2	1907,03	2,02	0,1649
CD	41,1736	1	41,1736	0,04	0,8371
RESIDUAL	15091,5	16	943,216		
TOTAL (CORRECTED)	38697,2	35			

En el análisis de varianza se puede notar que el único factor que tiene efectos significativos sobre el objetivo ponderado de éste problema es el porcentaje de elitismo.

Tabla 91

Análisis de varianza mejorado para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	4318,17	2	2159,08	2,37	0,1118
B:Porcentaje de elitismo	7806,5	2	3903,25	4,28	0,0236
C:Probabilidad de cruce	52,5625	1	52,5625	0,06	0,8120
D:Probabilidad de mutación	47,8403	1	47,8403	0,05	0,8205
INTERACTIONS					
RESIDUAL	26472,1	29	912,832		
TOTAL (CORRECTED)	38697,2	35			

Después de mejorar el análisis de varianza se puede observar que el porcentaje de elitismo sigue siendo el único factor con efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 92

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	849,458			
Intensidad de mutación					
1	12	851,542	8,72177	833,704	869,38
2	12	861,708	8,72177	843,87	879,546
4	12	835,125	8,72177	817,287	852,963
Porcentaje de elitismo					
0	12	869,292	8,72177	851,454	887,13
25	12	845,042	8,72177	827,204	862,88
50	12	834,042	8,72177	816,204	851,88
Probabilidad de cruce					
60	18	848,25	7,1213	833,685	862,815
90	18	850,667	7,1213	836,102	865,231
Probabilidad de mutación					
10	18	850,611	7,1213	836,046	865,176
50	18	848,306	7,1213	833,741	862,87

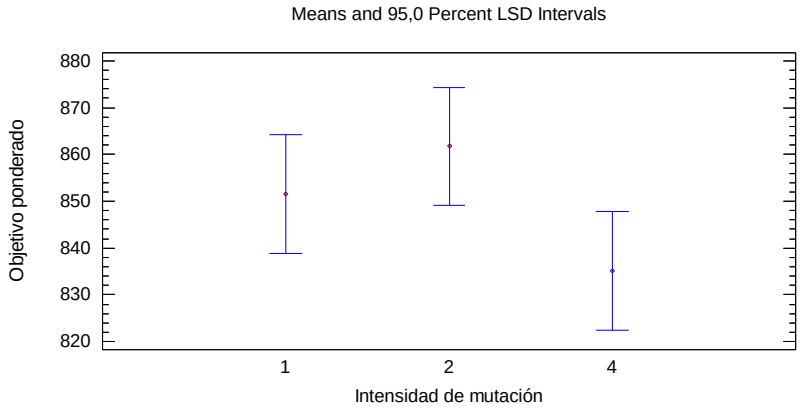


Figura 66. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

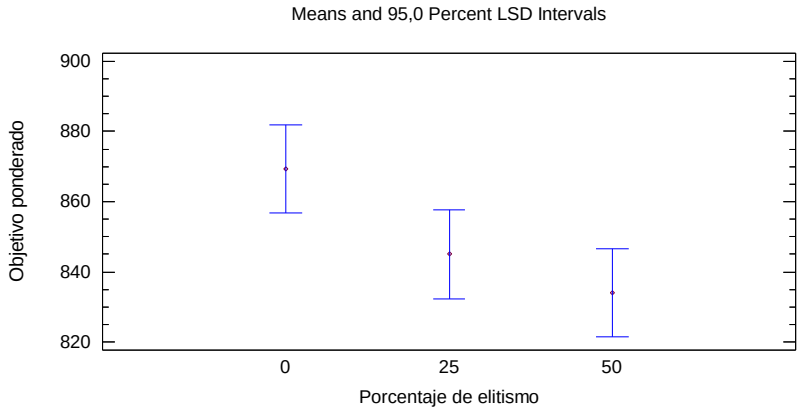


Figura 79. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

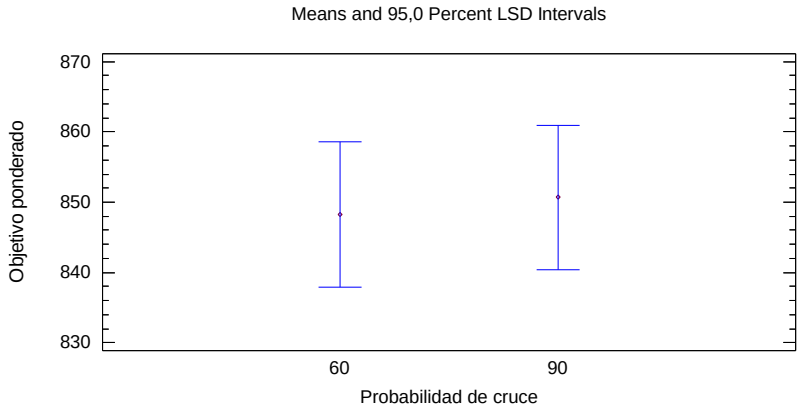


Figura 80. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

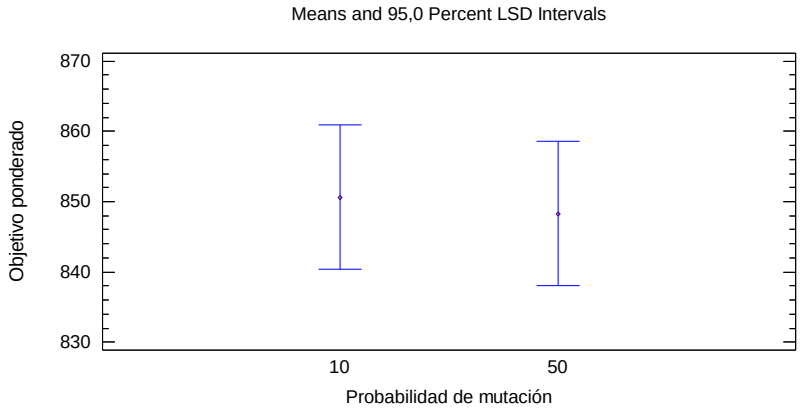


Figura 81. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

A continuación se muestra la validación de supuestos:

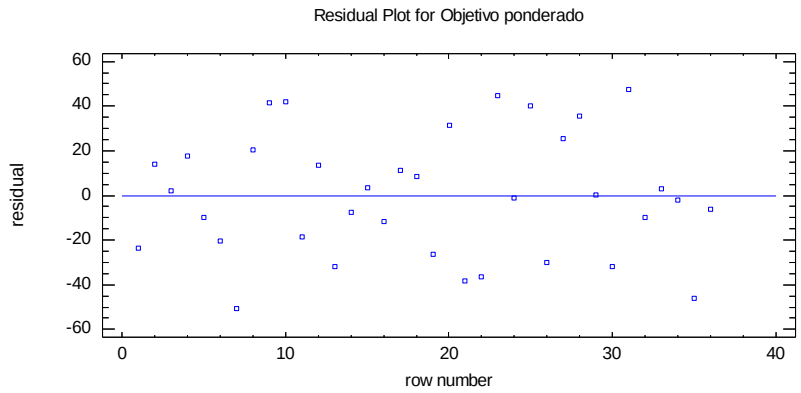


Figura 82. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado (αMakespan + βTTP) en el problema mediano (Problema 7).

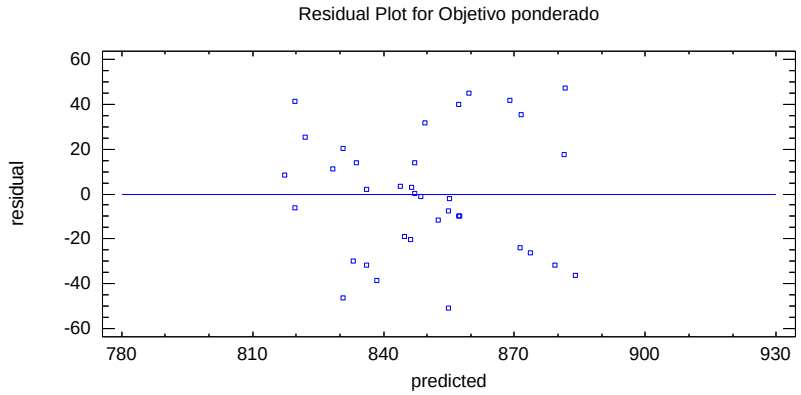


Figura 83. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado (αMakespan + βTTP) en el problema mediano (Problema 7).

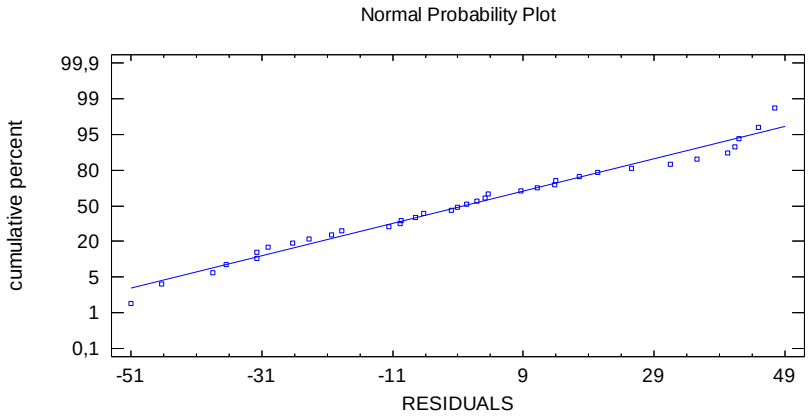


Figura 84. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

Diseño de experimentos para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado ($\boldsymbol{\alpha \text{Makespan} + \beta \text{TTP}}$) en el problema mediano (Problema 7):

Tabla 93

Análisis de varianza para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado ($\alpha Makespan + \beta TTP$) en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	9107,42	2	4553,71	3,06	0,0751
B:Porcentaje de elitismo	8037,21	2	4018,6	2,70	0,0978
C:Probabilidad de cruce	7,65444	1	7,65444	0,01	0,9437
D:Probabilidad de mutación	230,028	1	230,028	0,15	0,6995
INTERACTIONS					
AB	2465,41	4	616,353	0,41	0,7962
AC	3962,74	2	1981,37	1,33	0,2922
AD	828,696	2	414,348	0,28	0,7608
BC	1035,02	2	517,508	0,35	0,7117
BD	1164,1	2	582,048	0,39	0,6829
CD	393,361	1	393,361	0,26	0,6144
RESIDUAL	23834,9	16	1489,68		
TOTAL (CORRECTED)	51066,5	35			

De acuerdo con el análisis de varianza ninguno de los factores tiene efectos significativos sobre el objetivo ponderado de éste problema.

Tabla 94

Análisis de varianza mejorado para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	9107,42	2	4553,71	3,92	0,0311
B:Porcentaje de elitismo	8037,21	2	4018,6	3,46	0,0449
C:Probabilidad de cruce	7,65444	1	7,65444	0,01	0,9359
D:Probabilidad de mutación	230,028	1	230,028	0,20	0,6596
INTERACTIONS					
RESIDUAL	33684,2	29	1161,52		
TOTAL (CORRECTED)	51066,5	35			

Después de mejorar el análisis de varianza se puede observar que los factores intensidad de mutación y el porcentaje de elitismo tienen efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 95

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	896,65			
Intensidad de mutación					
1	12	910,967	9,83838	890,845	931,088
2	12	904,517	9,83838	884,395	924,638
4	12	874,467	9,83838	854,345	894,588
Porcentaje de elitismo					
0	12	917,633	9,83838	897,512	937,755
25	12	888,317	9,83838	868,195	908,438
50	12	884,0	9,83838	863,878	904,122
Probabilidad de cruce					
60	18	897,111	8,033	880,682	913,54
90	18	896,189	8,033	879,76	912,618
Probabilidad de mutación					
10	18	899,178	8,033	882,748	915,607
50	18	894,122	8,033	877,693	910,552

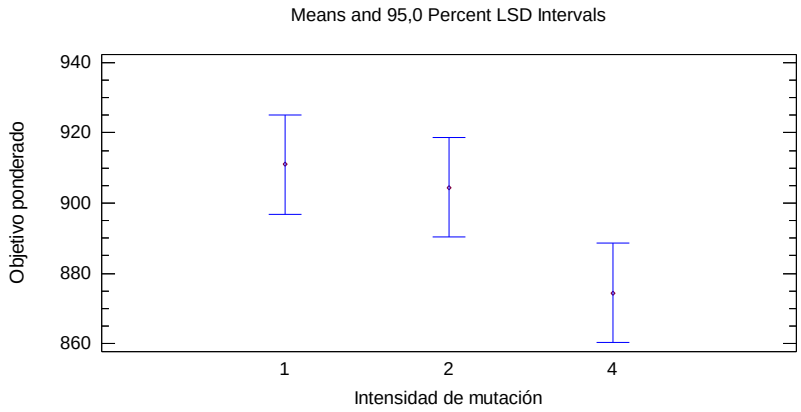


Figura 85. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

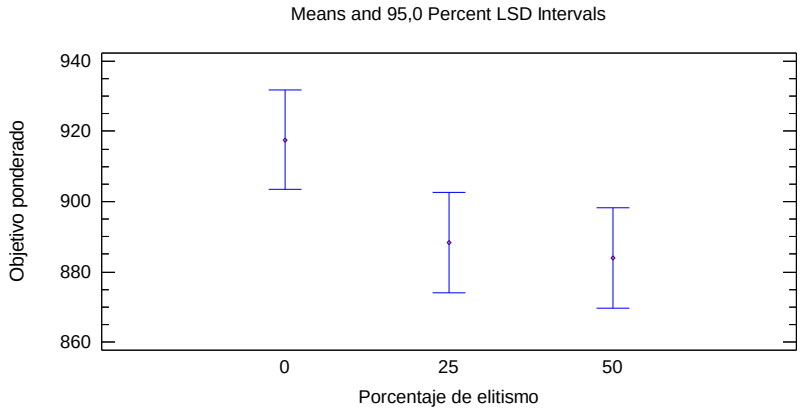


Figura 86. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

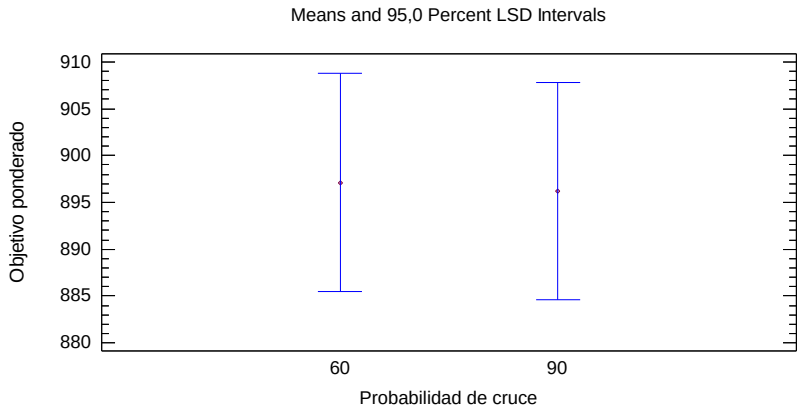


Figura 87. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

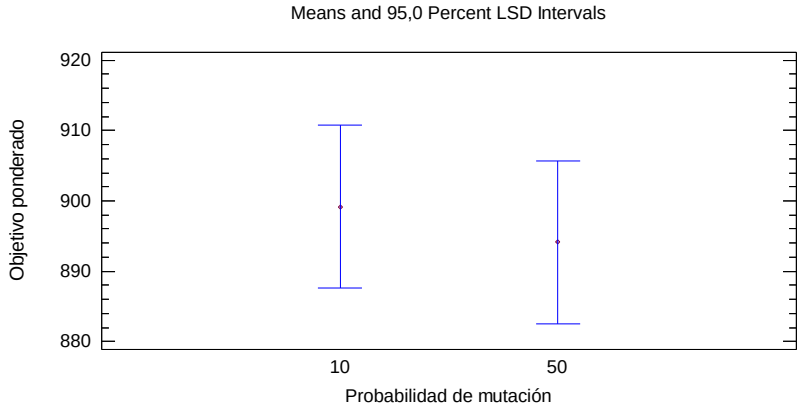


Figura 88. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

A continuación se muestra la validación de supuestos:

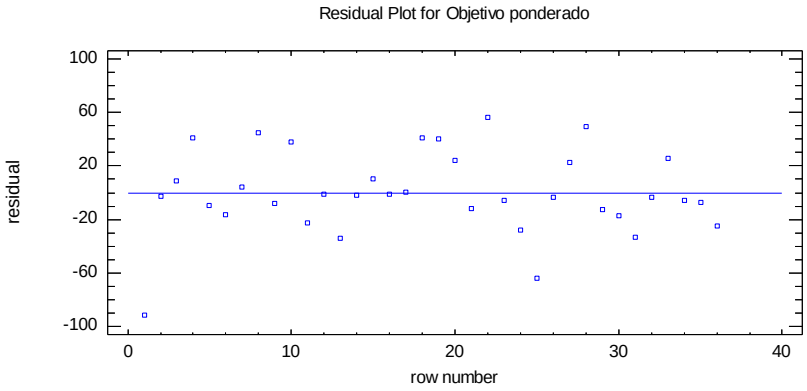


Figura 89. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

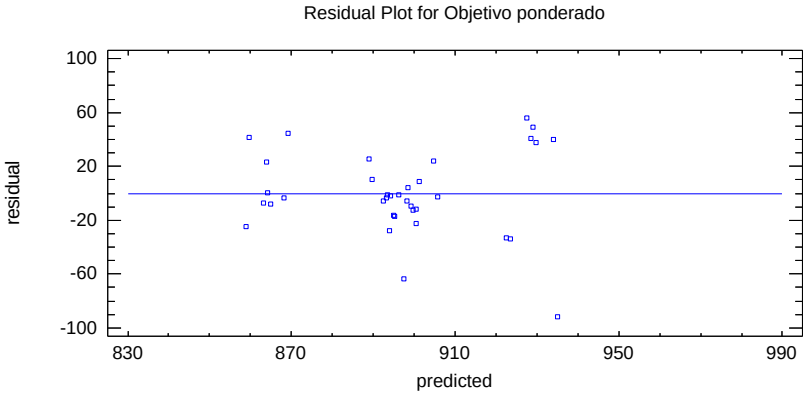


Figura 90. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

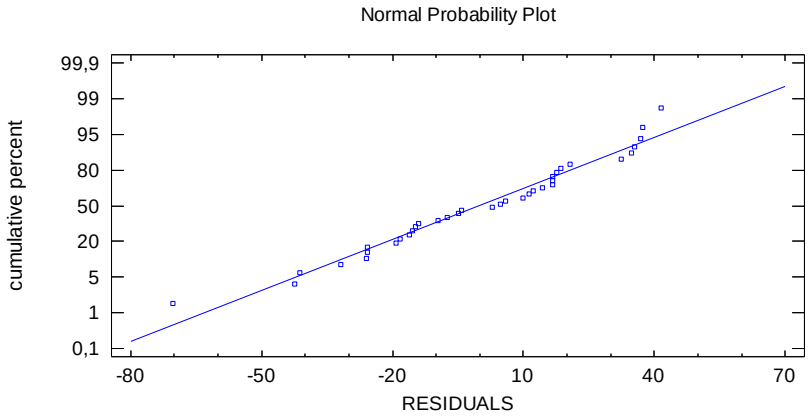


Figura 91. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha\text{Makespan} + \beta\text{TTP})$ en el problema mediano (Problema 7).

Diseño de experimentos para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado $(\boldsymbol{\alpha}\mathbf{Makespan} + \boldsymbol{\beta}\mathbf{TTP})$ en el problema mediano (Problema 7):

Tabla 96

Análisis de varianza para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	2690,4	2	1345,2	4,12	0,0359
B:Porcentaje de elitismo	1870,17	2	935,086	2,87	0,0863
C:Probabilidad de cruce	2296,01	1	2296,01	7,04	0,0174
D:Probabilidad de mutación	0,466944	1	0,466944	0,00	0,9703
INTERACTIONS					
AB	1820,03	4	455,008	1,39	0,2802
AC	569,524	2	284,762	0,87	0,4367
AD	361,811	2	180,905	0,55	0,5849
BC	48,0772	2	24,0386	0,07	0,9293
BD	85,3839	2	42,6919	0,13	0,8783
CD	1010,18	1	1010,18	3,10	0,0975
RESIDUAL	5218,97	16	326,185		
TOTAL (CORRECTED)	15971,0	35			

De acuerdo con el análisis de varianza los factores intensidad de mutación y probabilidad de cruce tienen efectos significativos sobre el objetivo ponderado de éste problema.

Tabla 97

Análisis de varianza mejorado para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	2690,4	2	1345,2	4,28	0,0235
B:Porcentaje de elitismo	1870,17	2	935,086	2,98	0,0668
C:Probabilidad de cruce	2296,01	1	2296,01	7,31	0,0114
D:Probabilidad de mutación	0,466944	1	0,466944	0,00	0,9695
INTERACTIONS					
RESIDUAL	9113,98	29	314,275		
TOTAL (CORRECTED)	15971,0	35			

Después de mejorar el análisis de varianza se puede observar que los mismos factores, intensidad de mutación y probabilidad de cruce, siguen teniendo efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 98

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	933,592			
Intensidad de mutación					
1	12	937,608	5,11758	927,142	948,075
2	12	941,583	5,11758	931,117	952,05
4	12	921,583	5,11758	911,117	932,05
Porcentaje de elitismo					
0	12	943,783	5,11758	933,317	954,25
25	12	928,642	5,11758	918,175	939,108
50	12	928,35	5,11758	917,883	938,817
Probabilidad de cruce					
60	18	925,606	4,17848	917,06	934,152
90	18	941,578	4,17848	933,032	950,124
Probabilidad de mutación					
10	18	933,478	4,17848	924,932	942,024
50	18	933,706	4,17848	925,16	942,252

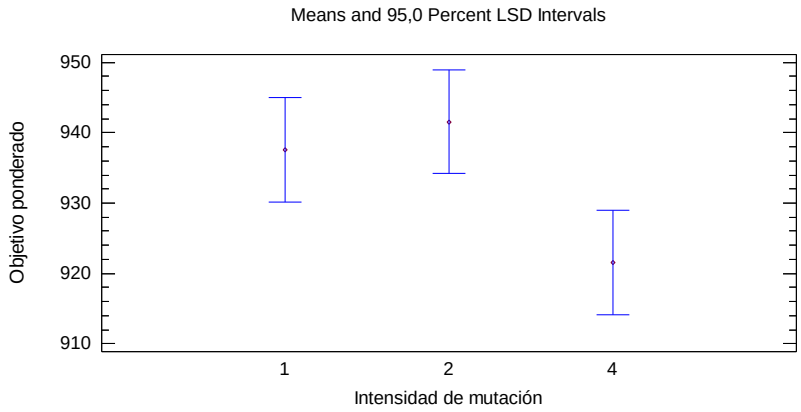


Figura 92. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

|

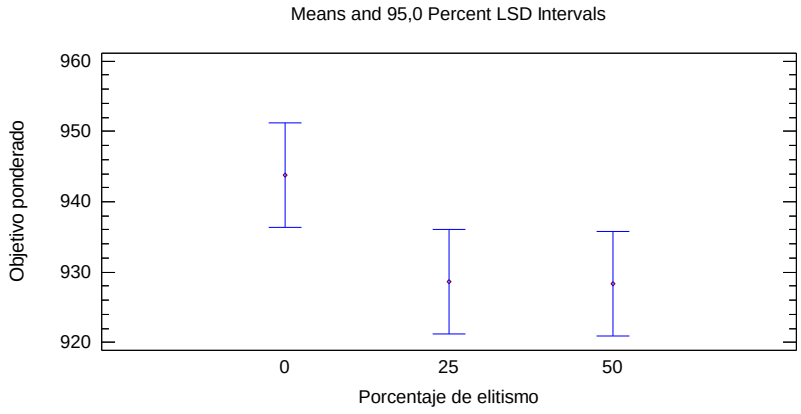


Figura 93. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

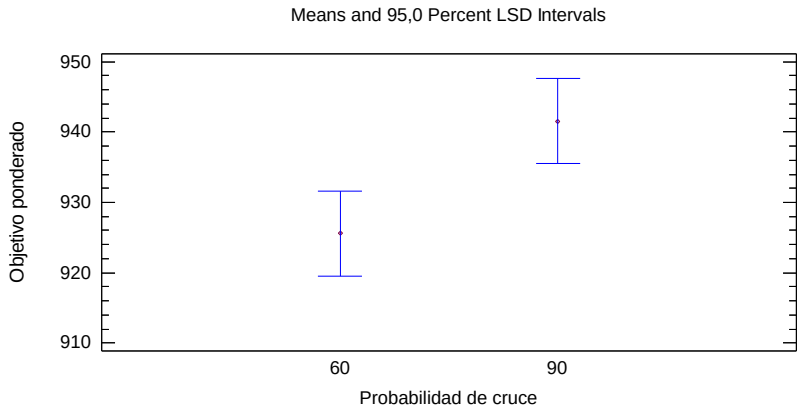


Figura 94. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

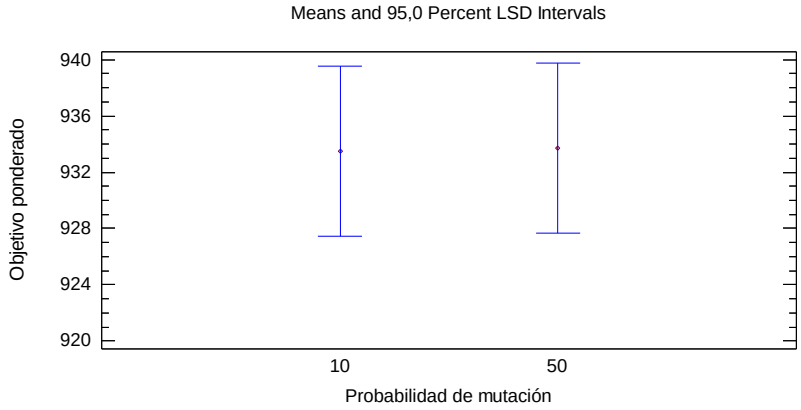


Figura 95. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

A continuación se muestra la validación de supuestos:

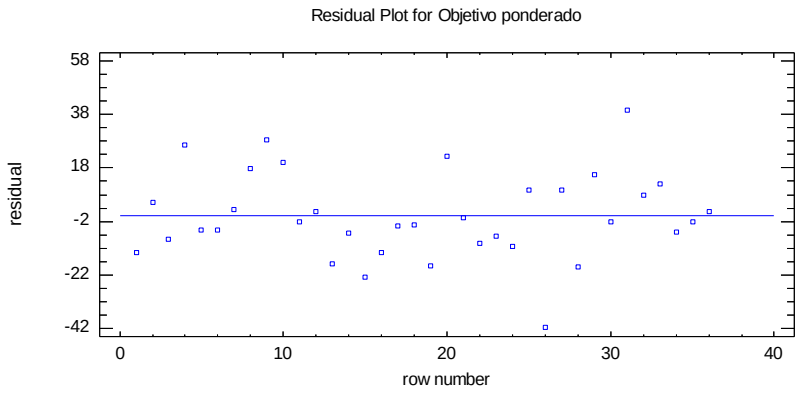


Figura 96. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

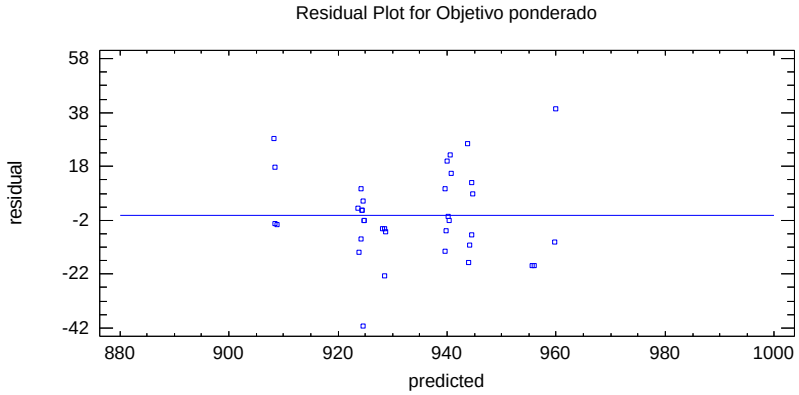


Figura 97. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

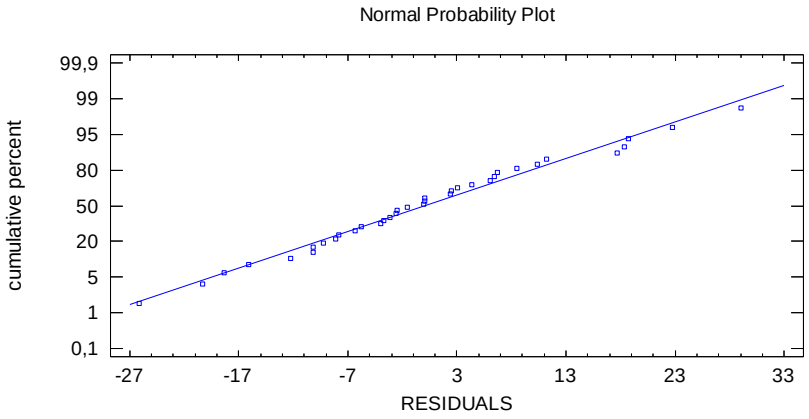


Figura 98. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado $(\alpha\text{Makespan} + \beta\text{TTP})$ en el problema mediano (Problema 7).

Diseño de experimentos para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado $(\boldsymbol{\alpha}\mathbf{Makespan} + \boldsymbol{\beta}\mathbf{TTP})$ en el problema mediano (Problema 7):

Tabla 99

Análisis de varianza para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	182811,	2	91405,5	22,74	0,0000
B:Porcentaje de elitismo	18804,3	2	9402,17	2,34	0,1284
C:Probabilidad de cruce	72603,3	1	72603,3	18,07	0,0006
D:Probabilidad de mutación	2111,4	1	2111,4	0,53	0,4790
INTERACTIONS					
AB	39252,6	4	9813,15	2,44	0,0891
AC	169116,	2	84558,0	21,04	0,0000
AD	5535,05	2	2767,52	0,69	0,5165
BC	13331,6	2	6665,79	1,66	0,2215
BD	2665,54	2	1332,77	0,33	0,7226
CD	3008,52	1	3008,52	0,75	0,3997
RESIDUAL	64300,3	16	4018,77		
TOTAL (CORRECTED)	573540,	35			

De acuerdo con el análisis de varianza los factores intensidad de mutación y probabilidad de cruce tienen efectos significativos sobre el objetivo ponderado de éste problema.

Tabla 100

Análisis de varianza mejorado para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	182811,	2	91405,5	19,27	0,0000
B:Porcentaje de elitismo	18804,3	2	9402,17	1,98	0,1574
C:Probabilidad de cruce	72603,3	1	72603,3	15,30	0,0006
D:Probabilidad de mutación	2111,4	1	2111,4	0,45	0,5104
INTERACTIONS					
AC	169116,	2	84558,0	17,82	0,0000
RESIDUAL	128094,	27	4744,21		
TOTAL (CORRECTED)	573540,	35			

Después de mejorar el análisis de varianza se puede observar que los mismos factores, intensidad de mutación y probabilidad de cruce, siguen teniendo efectos significativos sobre el objetivo, además de la interacción entre ellos.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 101

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7) (Parte 1 de 2)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	922,892			
Intensidad de mutación					
1	12	975,317	19,8834	934,519	1016,11
2	12	971,217	19,8834	930,419	1012,01
4	12	822,142	19,8834	781,344	862,939
Porcentaje de elitismo					
0	12	954,833	19,8834	914,036	995,631
25	12	902,642	19,8834	861,844	943,439
50	12	911,2	19,8834	870,402	951,998
Probabilidad de cruce					
60	18	877,983	16,2348	844,672	911,294
90	18	967,8	16,2348	934,489	1001,11

Tabla 102

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado ($\alpha Makespan + \beta TTP$) en el problema mediano (Problema 7) (Parte 2 de 2)

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	922,892			
Probabilidad de mutación					
10	18	930,55	16,2348	897,239	963,861
50	18	915,233	16,2348	881,922	948,544
Intensidad de mutación by Probabilidad de cruce					
1,60	6	975,433	28,1194	917,737	1033,13
1,90	6	975,2	28,1194	917,504	1032,9
2,60	6	978,133	28,1194	920,437	1035,83
2,90	6	964,3	28,1194	906,604	1022,0
4,60	6	680,383	28,1194	622,687	738,08
4,90	6	963,9	28,1194	906,204	1021,6

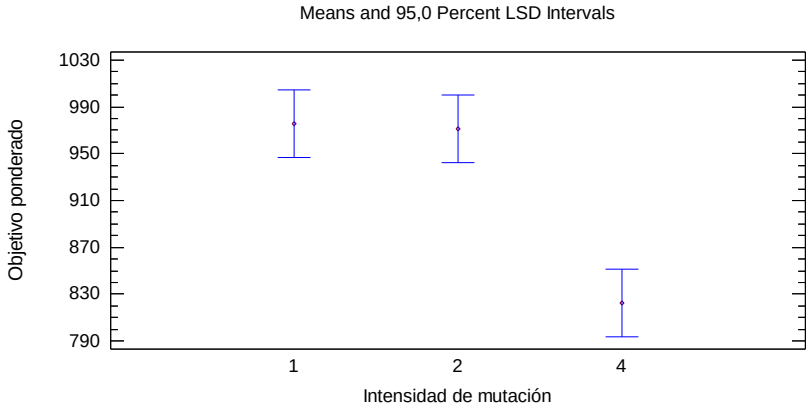


Figura 99. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado ($\alpha Makespan + \beta TTP$) en el problema mediano (Problema 7).

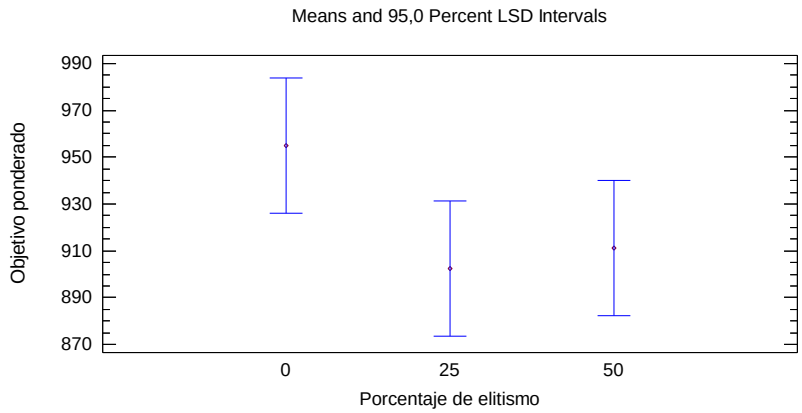


Figura 100. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

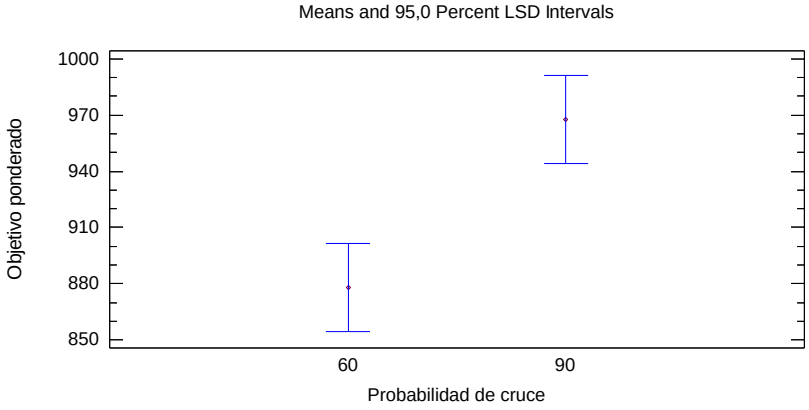


Figura 101. Gráfica de medias para el factor probablidad de cruce para la evaluación de la

combinación α = 80% y β = 20% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

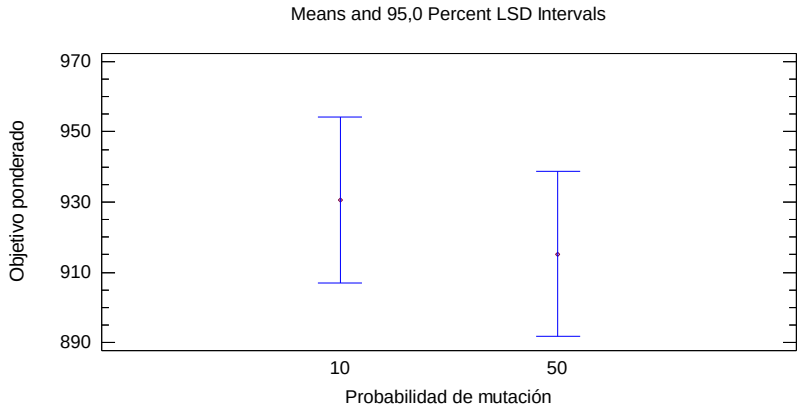


Figura 102. Gráfica de medias para el factor probablidad de mutación para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

La gráfica de interacción de los factores que tienen relación se encuentra a continuación:

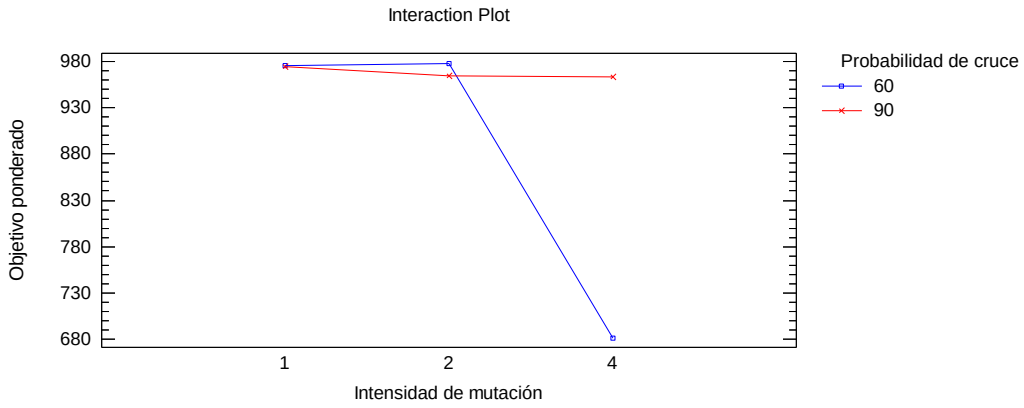


Figura 103. Gráfica de interacción entre los factores intensidad de mutación y probabilidad de cruce para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).interacción

A continuación se muestra la validación de supuestos:

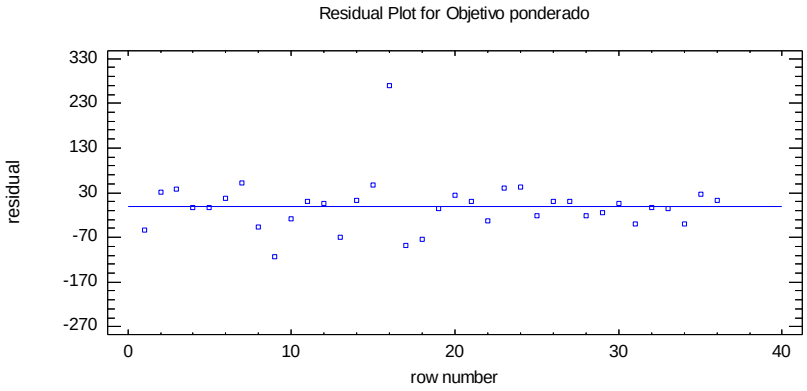


Figura 104. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

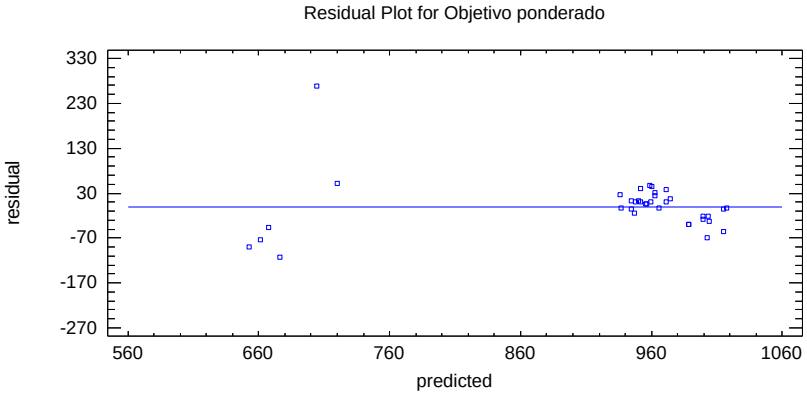


Figura 105. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

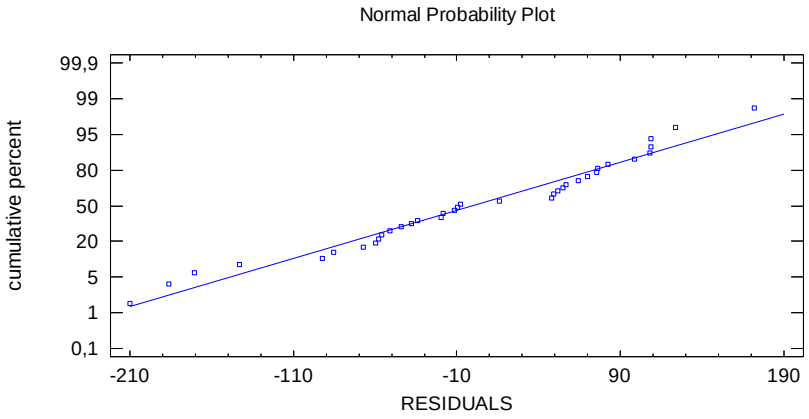


Figura 106. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

Diseño de experimentos para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7):

Tabla 103

Análisis de varianza para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema mediano (Problema 7)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	1760,87	2	880,434	2,68	0,0991
B:Porcentaje de elitismo	278,604	2	139,302	0,42	0,6615
C:Probabilidad de cruce	475,24	1	475,24	1,45	0,2465
D:Probabilidad de mutación	53,7778	1	53,7778	0,16	0,6911
INTERACTIONS					
AB	1586,35	4	396,587	1,21	0,3461
AC	721,787	2	360,893	1,10	0,3572
AD	269,282	2	134,641	0,41	0,6705
BC	93,735	2	46,8675	0,14	0,8681
BD	695,594	2	347,797	1,06	0,3700
CD	180,454	1	180,454	0,55	0,4693
RESIDUAL	5255,83	16	328,49		
TOTAL (CORRECTED)	11371,5	35			

De acuerdo con el análisis de varianza ninguno de los factores tiene efectos significativos sobre el objetivo ponderado de éste problema.

Tabla 104

Análisis de varianza mejorado para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	1760,87	2	880,434	2,90	0,0711
B:Porcentaje de elitismo	278,604	2	139,302	0,46	0,6365
C:Probabilidad de cruce	475,24	1	475,24	1,57	0,2208
D:Probabilidad de mutación	53,7778	1	53,7778	0,18	0,6769
INTERACTIONS					
RESIDUAL	8803,03	29	303,553		
TOTAL (CORRECTED)	11371,5	35			

Aunque se realizó una mejora del análisis de varianza se puede observar que ninguno de los factores tiene efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 105

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema mediano (Problema 7)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	997,961			
Intensidad de mutación					
1	12	1007,45	5,02952	997,163	1017,74
2	12	995,633	5,02952	985,347	1005,92
4	12	990,8	5,02952	980,513	1001,09
Porcentaje de elitismo					
0	12	999,325	5,02952	989,038	1009,61
25	12	994,083	5,02952	983,797	1004,37
50	12	1000,48	5,02952	990,188	1010,76
Probabilidad de cruce					
60	18	1001,59	4,10659	993,196	1009,99
90	18	994,328	4,10659	985,929	1002,73
Probabilidad de mutación					
10	18	996,739	4,10659	988,34	1005,14
50	18	999,183	4,10659	990,784	1007,58

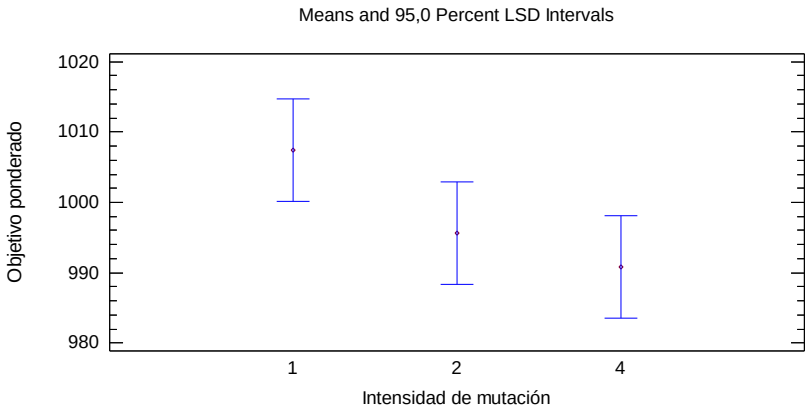


Figura 107. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

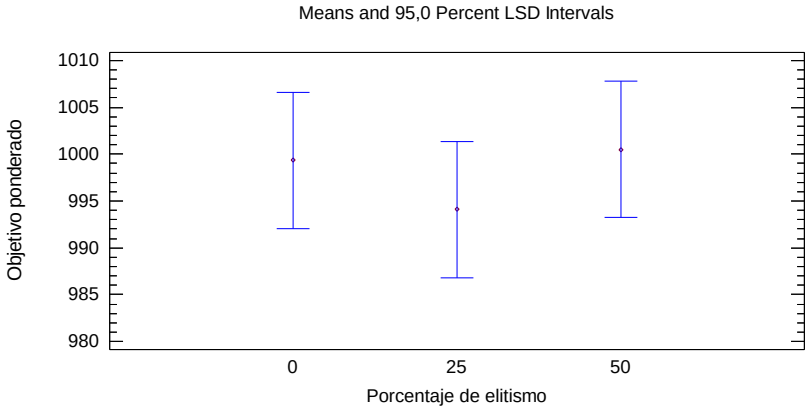


Figura 108. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

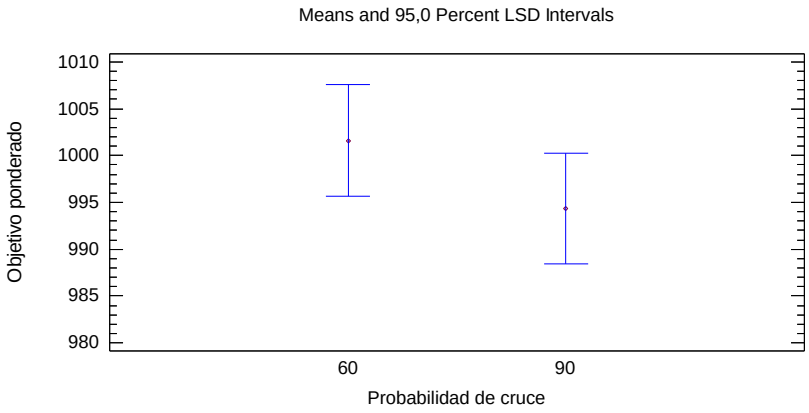


Figura 109. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

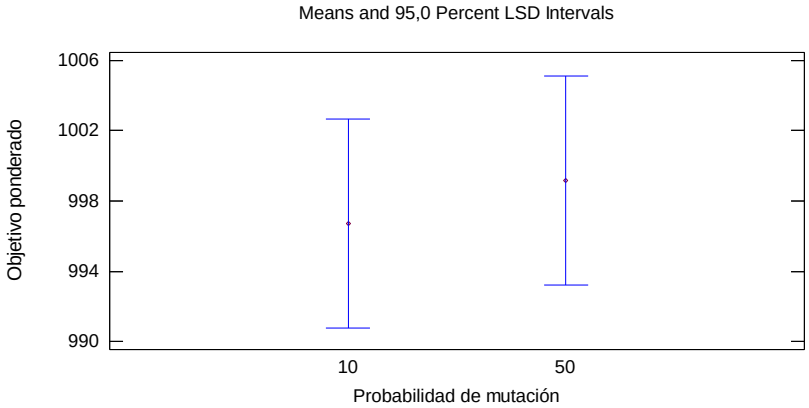


Figura 110. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

A continuación se muestra la validación de supuestos:

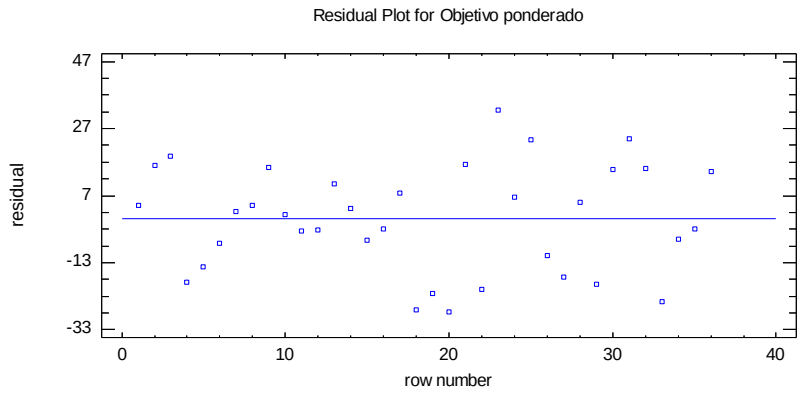


Figura 111. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

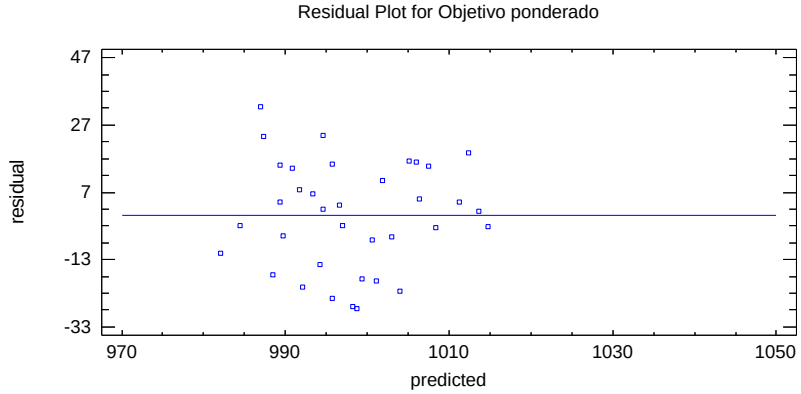


Figura 112. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema mediano (Problema 7).

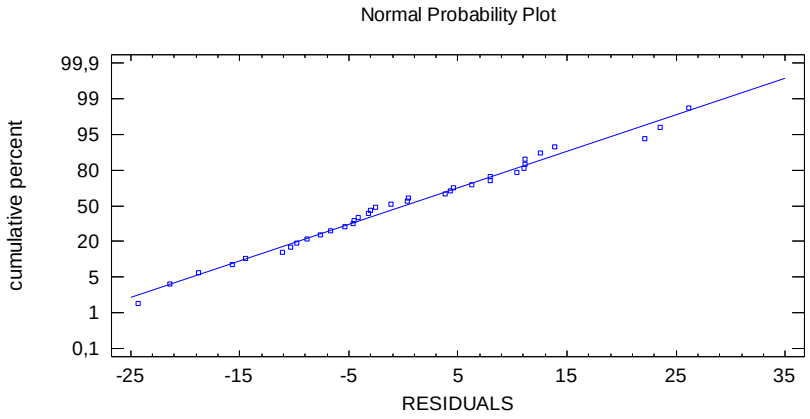


Figura 113. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema mediano (Problema 7).

Diseño de experimentos para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\boldsymbol{\alpha \text{Makespan} + \beta \text{TTP}})$ en el problema grande (Problema 9):

Tabla 106

Análisis de varianza para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema grande (Problema 9)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	6,23389	2	3,11694	3,58	0,0520
B:Porcentaje de elitismo	0,842222	2	0,421111	0,48	0,6254
C:Probabilidad de cruce	2,77778	1	2,77778	3,19	0,0931
D:Probabilidad de mutación	3,86778	1	3,86778	4,44	0,0512
INTERACTIONS					
AB	2,01778	4	0,504444	0,58	0,6821
AC	1,25722	2	0,628611	0,72	0,5012
AD	2,17389	2	1,08694	1,25	0,3137
BC	3,33556	2	1,66778	1,91	0,1797
BD	2,41556	2	1,20778	1,39	0,2785
CD	0,0277778	1	0,0277778	0,03	0,8605
RESIDUAL	13,9394	16	0,871215		
TOTAL (CORRECTED)	38,8889	35			

De acuerdo con el análisis de varianza ningún factor tiene efectos significativos sobre el objetivo ponderado de éste problema.

Tabla 107

Análisis de varianza mejorado para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	6,23389	2	3,11694	3,59	0,0404
B:Porcentaje de elitismo	0,842222	2	0,421111	0,49	0,6205
C:Probabilidad de cruce	2,77778	1	2,77778	3,20	0,0841
D:Probabilidad de mutación	3,86778	1	3,86778	4,46	0,0435
INTERACTIONS					
RESIDUAL	25,1672	29	0,867835		
TOTAL (CORRECTED)	38,8889	35			

Después de mejorar el análisis de varianza se puede observar que los factores intensidad de mutación y probabilidad de mutación tienen efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 108

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	26,9556			
Intensidad de mutación					
1	12	27,0417	0,268923	26,4917	27,5917
2	12	27,4167	0,268923	26,8667	27,9667
4	12	26,4083	0,268923	25,8583	26,9583
Porcentaje de elitismo					
0	12	27,1167	0,268923	26,5667	27,6667
25	12	26,75	0,268923	26,2	27,3
50	12	27,0	0,268923	26,45	27,55
Probabilidad de cruce					
60	18	26,6778	0,219575	26,2287	27,1269
90	18	27,2333	0,219575	26,7843	27,6824
Probabilidad de mutación					
10	18	27,2833	0,219575	26,8343	27,7324
50	18	26,6278	0,219575	26,1787	27,0769

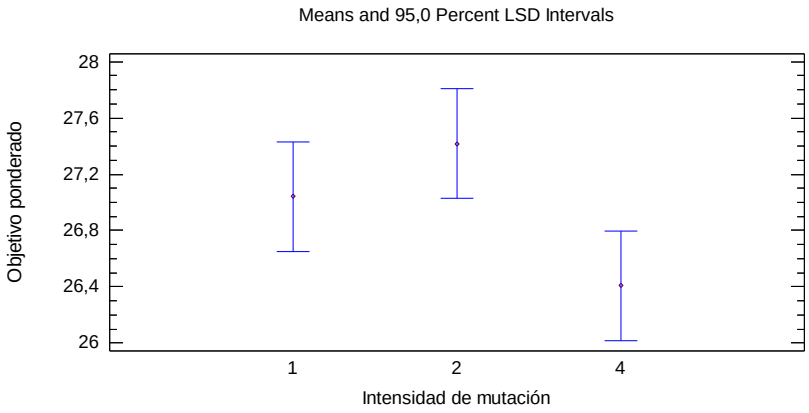


Figura 114. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

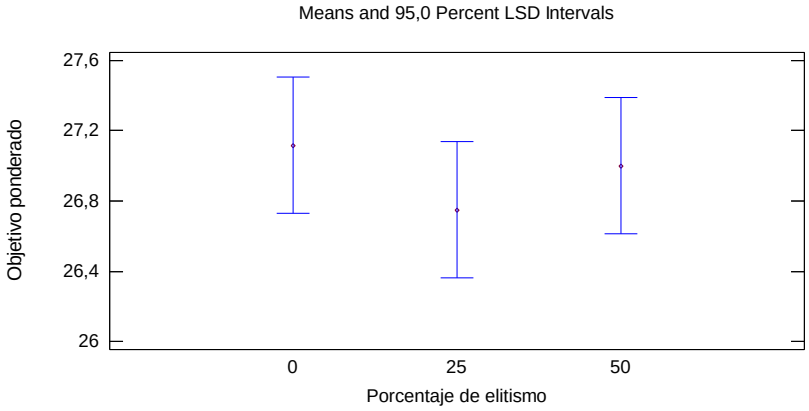


Figura 115. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

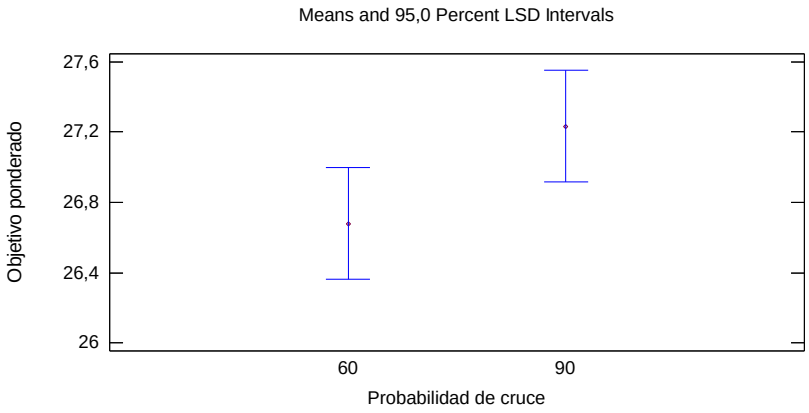


Figura 116. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

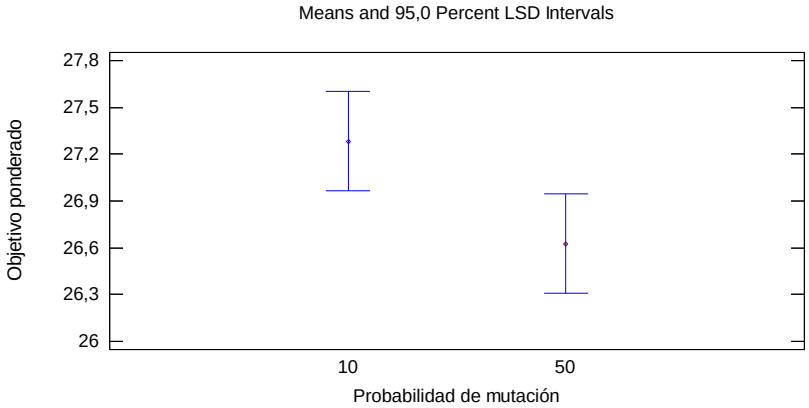


Figura 117. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

A continuación se muestra la validación de supuestos:

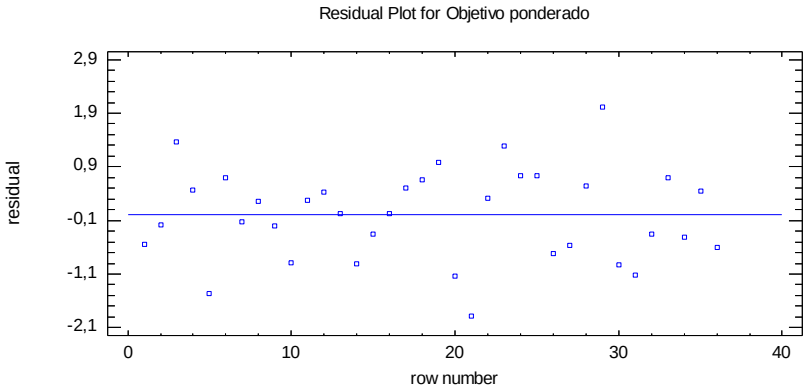


Figura 118. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

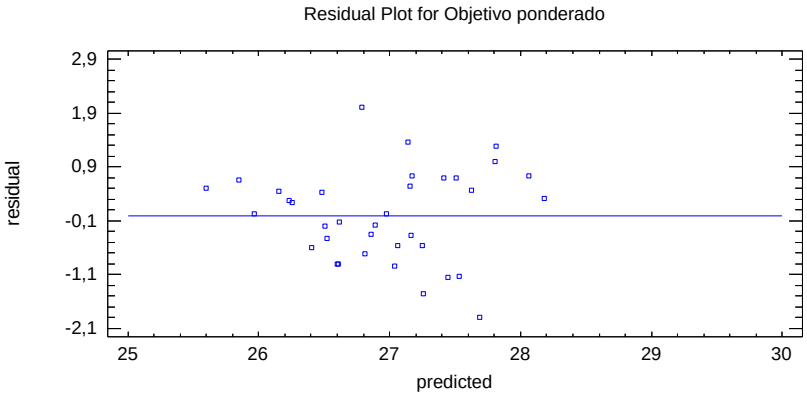


Figura 119. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

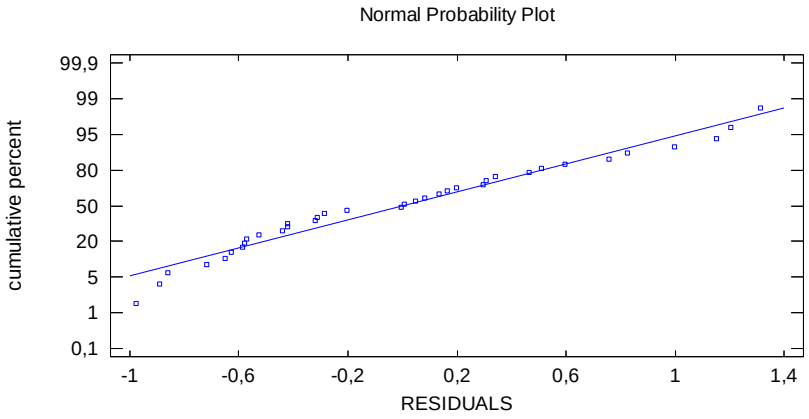


Figura 120. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 10% y β = 90% del Objetivo Ponderado $(\alpha\text{Makespan} + \beta\text{TTP})$ en el problema grande (Problema 9).

Diseño de experimentos para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\boldsymbol{\alpha}\mathbf{Makespan} + \boldsymbol{\beta}\mathbf{TTP})$ en el problema grande (Problema 9):

Tabla 109

Análisis de varianza para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema grande (Problema 9)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	9,84889	2	4,92444	2,56	0,1084
B:Porcentaje de elitismo	12,1622	2	6,08111	3,16	0,0696
C:Probabilidad de cruce	6,25	1	6,25	3,25	0,0903
D:Probabilidad de mutación	5,29	1	5,29	2,75	0,1167
INTERACTIONS					
AB	2,97111	4	0,742778	0,39	0,8153
AC	2,90667	2	1,45333	0,76	0,4857
AD	17,3867	2	8,69333	4,52	0,0278
BC	0,78	2	0,39	0,20	0,8185
BD	0,286667	2	0,143333	0,07	0,9285
CD	5,60111	1	5,60111	2,91	0,1072
RESIDUAL	30,7689	16	1,92306		
TOTAL (CORRECTED)	94,2522	35			

De acuerdo con el análisis de varianza ninguno de los factores por si solos tiene efectos significativos sobre el objetivo ponderado de éste problema. Sin embargo, la interacción entre la intensidad de mutación y la probabilidad de mutación si lo tienen.

Tabla 110

Análisis de varianza mejorado para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	9,84889	2	4,92444	3,07	0,0629
B:Porcentaje de elitismo	12,1622	2	6,08111	3,79	0,0354
C:Probabilidad de cruce	6,25	1	6,25	3,90	0,0587
D:Probabilidad de mutación	5,29	1	5,29	3,30	0,0805
INTERACTIONS					
AD	17,3867	2	8,69333	5,42	0,0105
RESIDUAL	43,3144	27	1,60424		
TOTAL (CORRECTED)	94,2522	35			

Después de realizar una mejora del análisis de varianza se puede observar que además de la interacción entre la intensidad de mutación y la probabilidad de mutación, también el porcentaje de elitismo tiene efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 111

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9) (Parte 1 de 2)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	52,8278			
Intensidad de mutación					
1	12	52,7167	0,365632	51,9665	53,4669
2	12	53,5167	0,365632	52,7665	54,2669
4	12	52,25	0,365632	51,4998	53,0002
Porcentaje de elitismo					
0	12	53,6333	0,365632	52,8831	54,3835
25	12	52,5667	0,365632	51,8165	53,3169
50	12	52,2833	0,365632	51,5331	53,0335
Probabilidad de cruce					
60	18	52,4111	0,298537	51,7986	53,0237

90	18	53,2444	0,298537	52,6319	53,857

Tabla 112

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9) (Parte 2 de 2)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	52,8278			
Probabilidad de mutación					
10	18	53,2111	0,298537	52,5986	53,8237
50	18	52,4444	0,298537	51,8319	53,057
Intensidad de mutación by Probabilidad de mutación					
1,10	6	52,3667	0,517081	51,3057	53,4276
1,50	6	53,0667	0,517081	52,0057	54,1276
2,10	6	54,8333	0,517081	53,7724	55,8943
2,50	6	52,2	0,517081	51,139	53,261
4,10	6	52,4333	0,517081	51,3724	53,4943
4,50	6	52,0667	0,517081	51,0057	53,1276

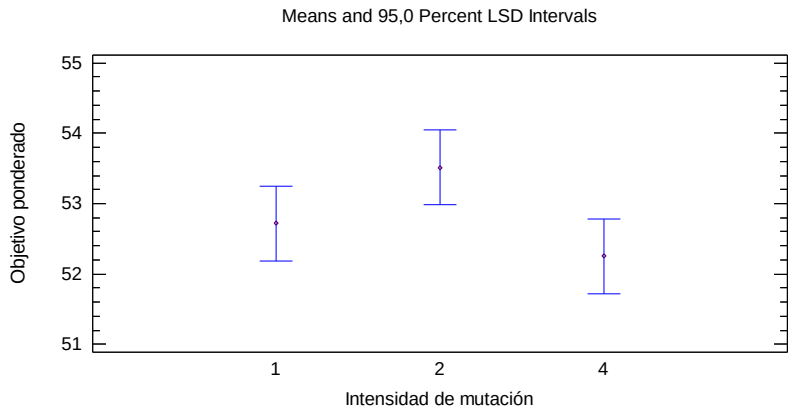


Figura 121. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema grande (Problema 9).

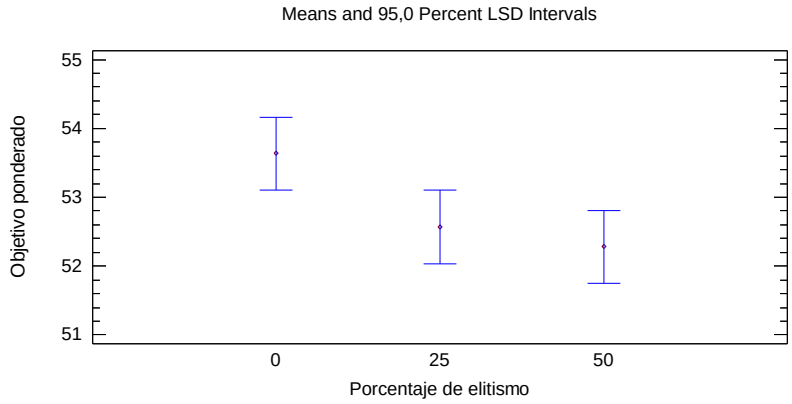


Figura 122. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9).

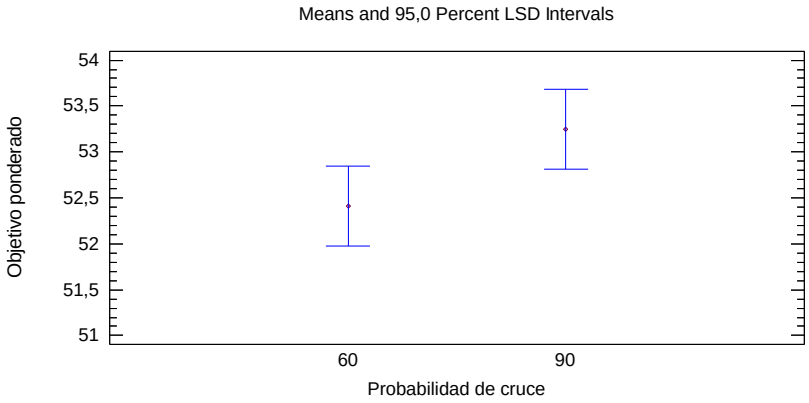


Figura 123. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9).

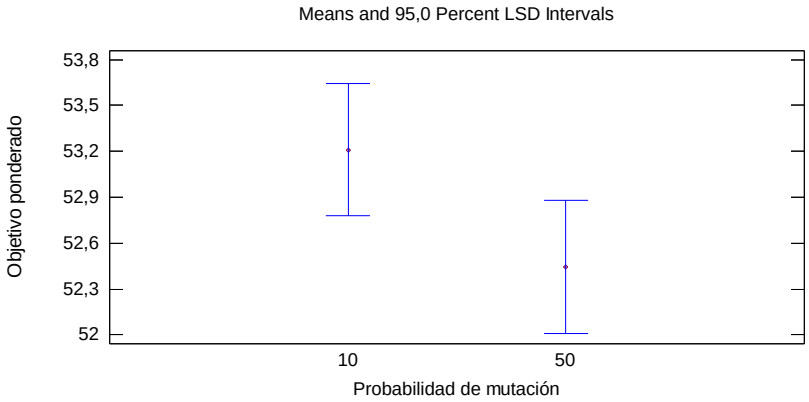


Figura 124. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9).

La gráfica de interacción de los factores que tienen relación se encuentra a continuación:

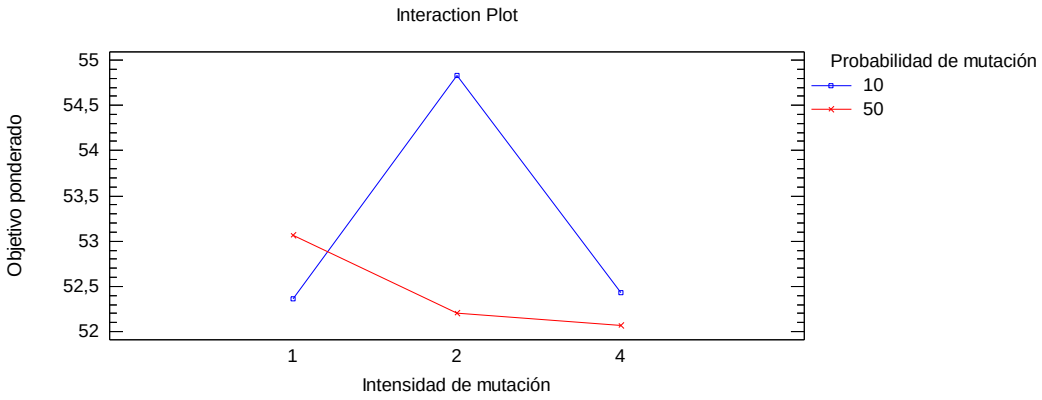


Figura 125. Gráfica de interacción entre los factores intensidad de mutación y probabilidad de mutación para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9).

A continuación se muestra la validación de supuestos:

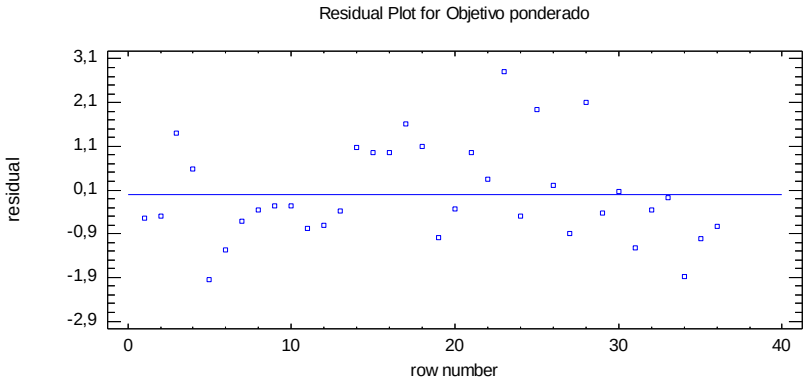


Figura 126. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

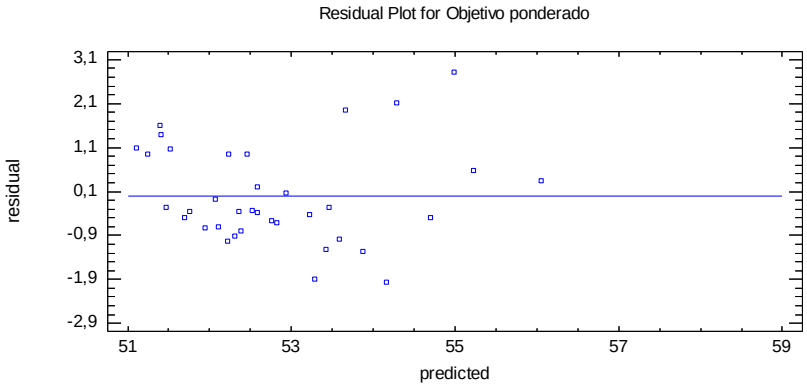


Figura 127. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

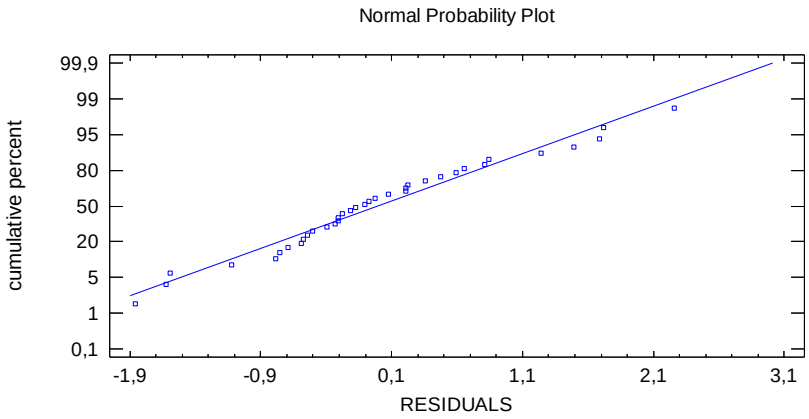


Figura 128. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 20% y β = 80% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9).

Diseño de experimentos para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\boldsymbol{\alpha \text{Makespan} + \beta \text{TTP}})$ en el problema grande (Problema 9):

Tabla 113

Análisis de varianza para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema grande (Problema 9)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	4,865	2	2,4325	1,69	0,2152
B:Porcentaje de elitismo	24,605	2	12,3025	8,57	0,0030
C:Probabilidad de cruce	5,76	1	5,76	4,01	0,0625
D:Probabilidad de mutación	9,81778	1	9,81778	6,84	0,0188
INTERACTIONS					
AB	7,13	4	1,7825	1,24	0,3332
AC	1,06167	2	0,530833	0,37	0,6968
AD	13,8839	2	6,94194	4,83	0,0228
BC	1,65167	2	0,825833	0,57	0,5739
BD	6,80389	2	3,40194	2,37	0,1256
CD	1,21	1	1,21	0,84	0,3723
RESIDUAL	22,9811	16	1,43632		
TOTAL (CORRECTED)	99,77	35			

Al realizar el análisis de varianza los factores porcentaje de elitismo y probabilidad de mutación tienen efectos significativos sobre el objetivo ponderado de éste problema.

Tabla 114

Análisis de varianza mejorado para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	4,865	2	2,4325	1,61	0,2188
B:Porcentaje de elitismo	24,605	2	12,3025	8,13	0,0017
C:Probabilidad de cruce	5,76	1	5,76	3,81	0,0614
D:Probabilidad de mutación	9,81778	1	9,81778	6,49	0,0168
INTERACTIONS					
AD	13,8839	2	6,94194	4,59	0,0192
RESIDUAL	40,8383	27	1,51253		
TOTAL (CORRECTED)	99,77	35			

Después de realizar una mejora del análisis de varianza los mismos factores y la misma interacción siguen teniendo efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 115

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9) (Parte 1 de 2)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	78,2833			
Intensidad de mutación					
1	12	78,075	0,355027	77,3465	78,8035
2	12	77,975	0,355027	77,2465	78,7035
4	12	78,8	0,355027	78,0715	79,5285
Porcentaje de elitismo					
0	12	79,4	0,355027	78,6715	80,1285
25	12	78,025	0,355027	77,2965	78,7535
50	12	77,425	0,355027	76,6965	78,1535
Probabilidad de cruce					
60	18	77,8833	0,289878	77,2886	78,4781
90	18	78,6833	0,289878	78,0886	79,2781
Probabilidad de mutación					
10	18	78,8056	0,289878	78,2108	79,4003

50	18	77,7611	0,289878	77,1663	78,3559

Tabla 116

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado ($\alpha Makespan + \beta TTP$) en el problema grande (Problema 9) (Parte 2 de 2)

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	78,2833			
Intensidad de mutación by Probabilidad de mutación					
1,10	6	78,4833	0,502084	77,4531	79,5135
1,50	6	77,6667	0,502084	76,6365	78,6969
2,10	6	77,8	0,502084	76,7698	78,8302
2,50	6	78,15	0,502084	77,1198	79,1802
4,10	6	80,1333	0,502084	79,1031	81,1635
4,50	6	77,4667	0,502084	76,4365	78,4969

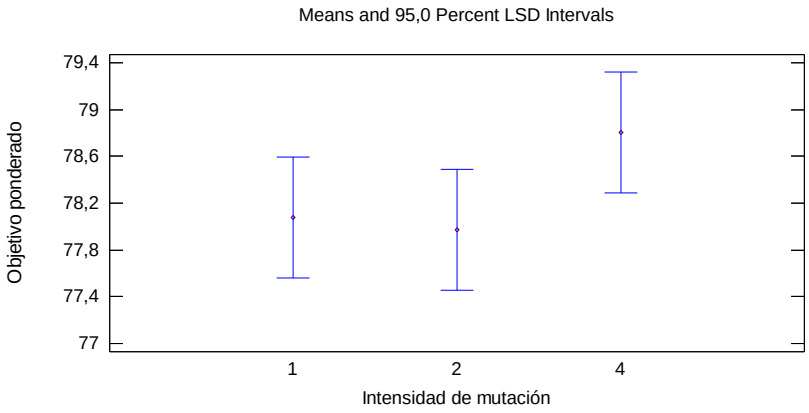


Figura 129. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado ($\alpha Makespan + \beta TTP$) en el problema grande (Problema 9).

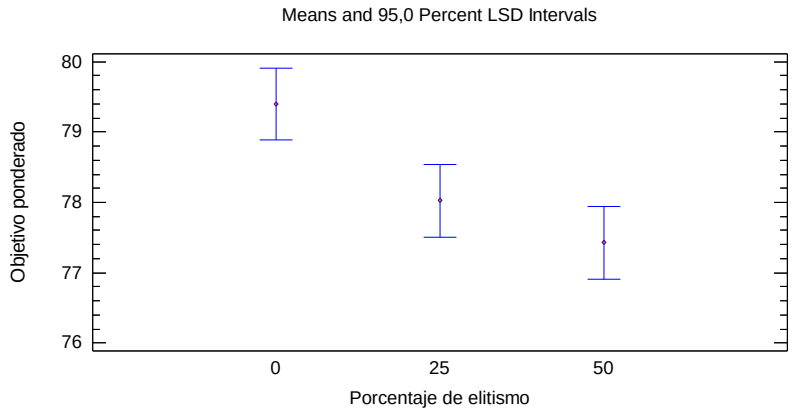


Figura 130. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

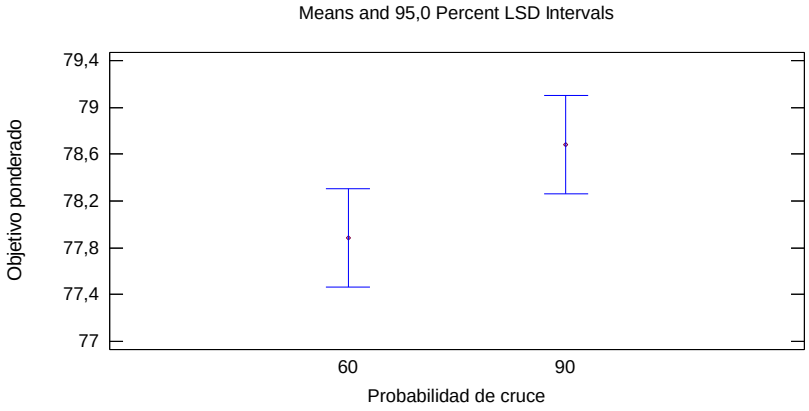


Figura 131. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

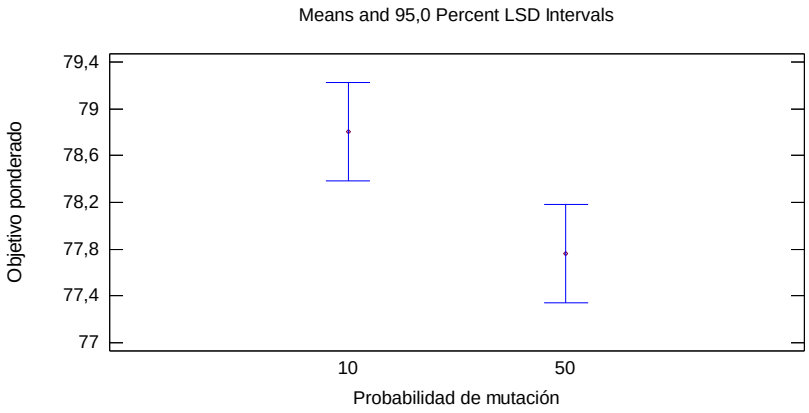


Figura 132. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9).

La gráfica de interacción de los factores que tienen relación se encuentra a continuación:

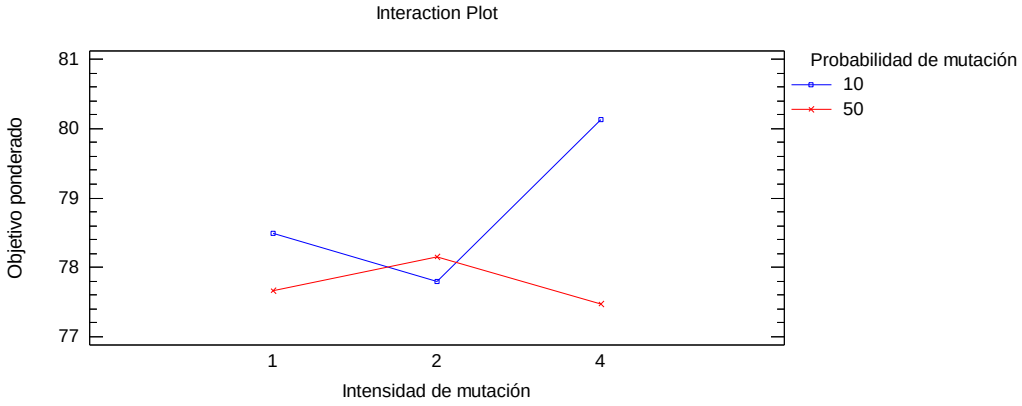


Figura 133. Gráfica de interacción entre los factores intensidad de mutación y probabilidad de mutación para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9).

A continuación se muestra la validación de supuestos:

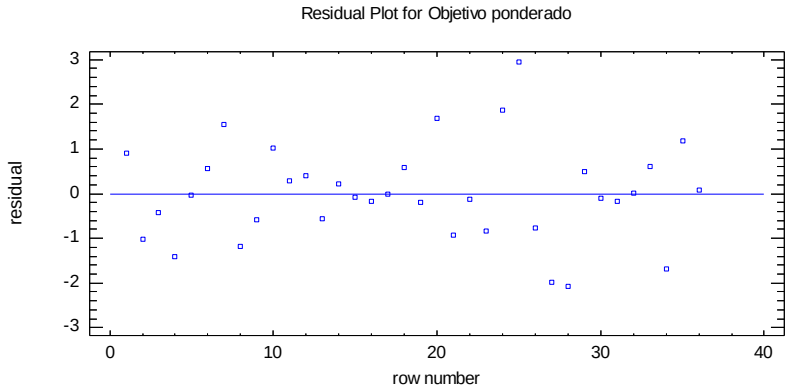


Figura 134. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9).

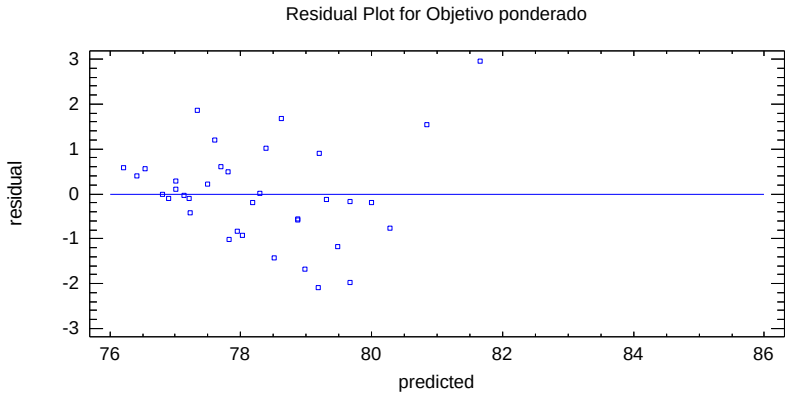


Figura 135. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9).

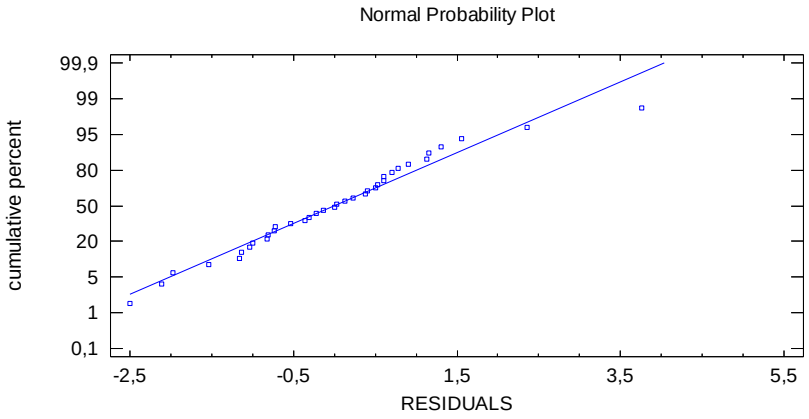


Figura 136. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 30% y β = 70% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9).

Diseño de experimentos para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9):

Tabla 117

Análisis de varianza para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado ($\alpha Makespan + \beta TTP$) en el problema grande (Problema 9)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	13,9356	2	6,96778	1,57	0,2381
B:Porcentaje de elitismo	24,8089	2	12,4044	2,80	0,0908
C:Probabilidad de cruce	3,73778	1	3,73778	0,84	0,3721
D:Probabilidad de mutación	25,6711	1	25,6711	5,79	0,0285
INTERACTIONS					
AB	14,1311	4	3,53278	0,80	0,5444
AC	8,81556	2	4,40778	0,99	0,3917
AD	13,0822	2	6,54111	1,48	0,2581
BC	8,22222	2	4,11111	0,93	0,4158
BD	1,50222	2	0,751111	0,17	0,8456
CD	0,00444444	1	0,00444444	0,00	0,9751
RESIDUAL	70,9244	16	4,43278		
TOTAL (CORRECTED)	184,836	35			

De acuerdo con el análisis de varianza el único factor que tiene efectos significativos sobre el objetivo ponderado de éste problema es la probabilidad de mutación.

Tabla 118

Análisis de varianza mejorado para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	13,9356	2	6,96778	1,73	0,1948
B:Porcentaje de elitismo	24,8089	2	12,4044	3,08	0,0611
C:Probabilidad de cruce	3,73778	1	3,73778	0,93	0,3431
D:Probabilidad de mutación	25,6711	1	25,6711	6,38	0,0173
INTERACTIONS					
RESIDUAL	116,682	29	4,02352		
TOTAL (CORRECTED)	184,836	35			

Después de realizar una mejora del análisis de varianza el mismo factor sigue teniendo efectos significativos sobre el objetivo.

Tabla 119

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	104,489			
Intensidad de mutación					
1	12	105,317	0,579046	104,132	106,501
2	12	104,333	0,579046	103,149	105,518
4	12	103,817	0,579046	102,632	105,001
Porcentaje de elitismo					
0	12	105,5	0,579046	104,316	106,684
25	12	104,5	0,579046	103,316	105,684
50	12	103,467	0,579046	102,282	104,651
Probabilidad de cruce					
60	18	104,811	0,472789	103,844	105,778
90	18	104,167	0,472789	103,2	105,134
Probabilidad de mutación					
10	18	105,333	0,472789	104,366	106,3
50	18	103,644	0,472789	102,677	104,611

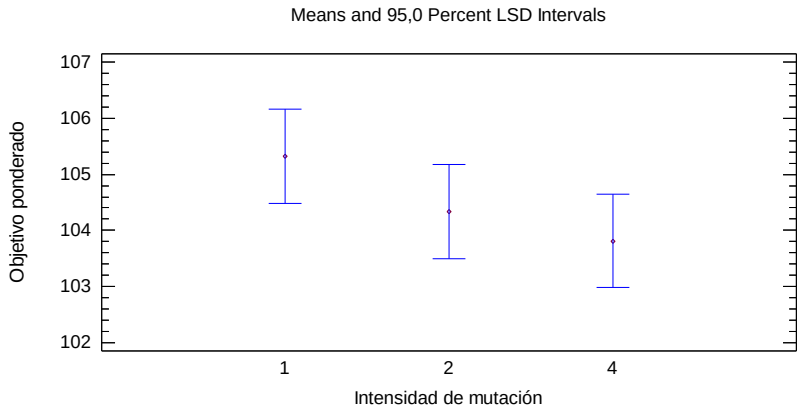


Figura 137. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

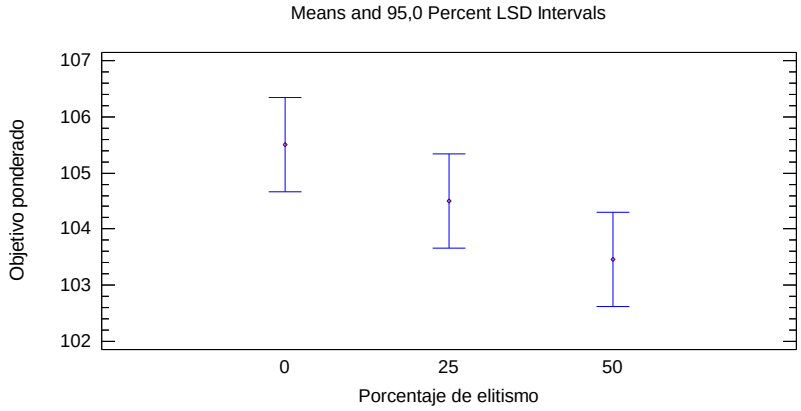


Figura 138. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

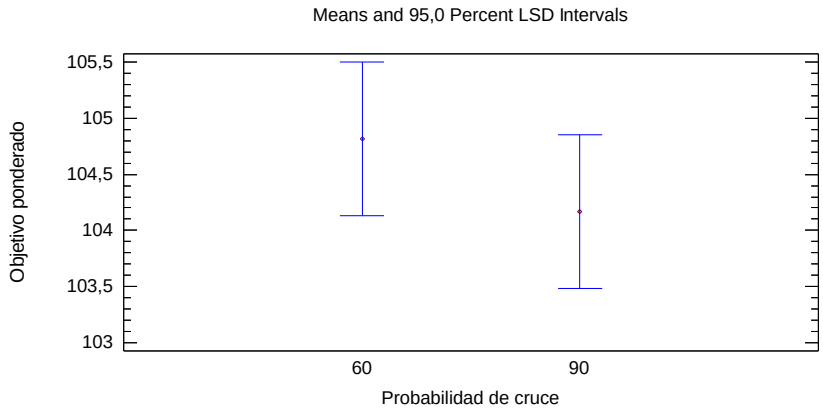


Figura 139. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado ($\alpha \mathrm{Makespan} + \beta \mathrm{TTP}$) en el problema grande (Problema 9).

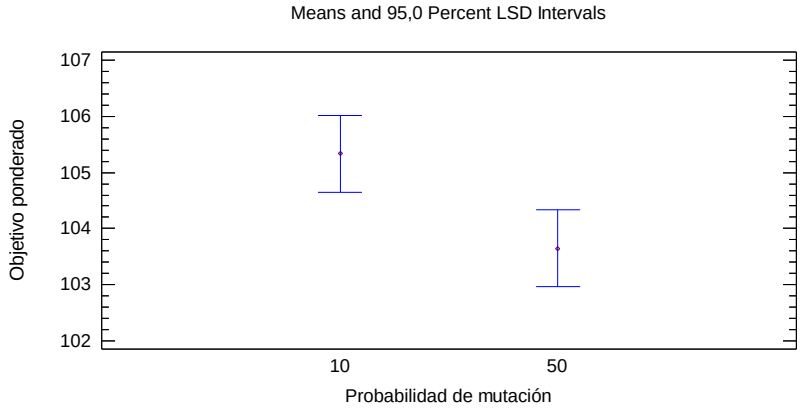


Figura 140. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado ($\alpha \mathrm{Makespan} + \beta \mathrm{TTP}$) en el problema grande (Problema 9).

A continuación se muestra la validación de supuestos:

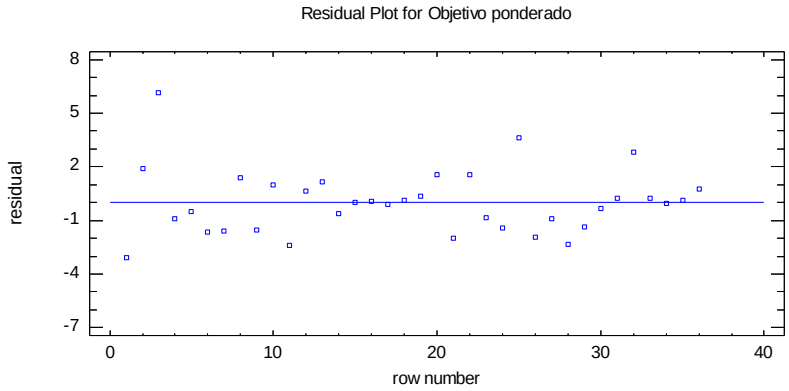


Figura 141. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9).

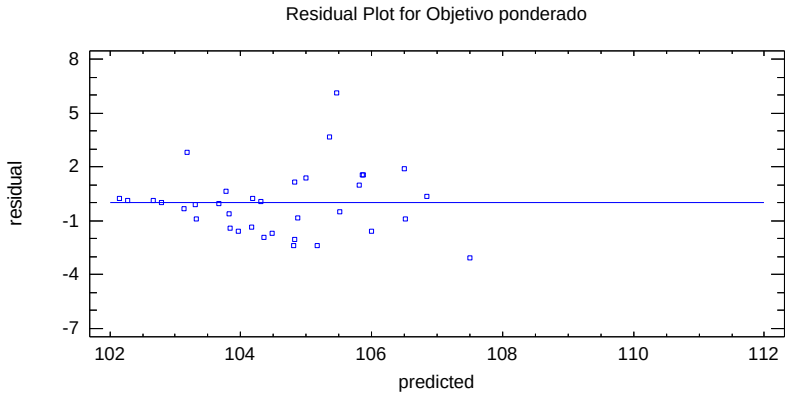


Figura 142. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9).

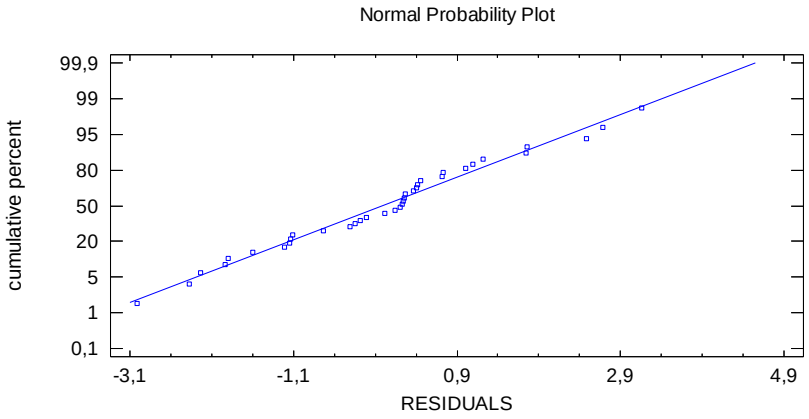


Figura 143. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 40% y β = 60% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9).

Diseño de experimentos para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9):

Tabla 120

Análisis de varianza para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado ($\alpha Makespan + \beta TTP$) en el problema grande (Problema 9)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	10,2917	2	5,14583	1,13	0,3461
B:Porcentaje de elitismo	89,5417	2	44,7708	9,87	0,0016
C:Probabilidad de cruce	0,25	1	0,25	0,06	0,8173
D:Probabilidad de mutación	0,25	1	0,25	0,06	0,8173
INTERACTIONS					
AB	11,4167	4	2,85417	0,63	0,6485
AC	0,875	2	0,4375	0,10	0,9086
AD	2,54167	2	1,27083	0,28	0,7592
BC	8,79167	2	4,39583	0,97	0,4005
BD	12,875	2	6,4375	1,42	0,2707
CD	3,36111	1	3,36111	0,74	0,4020
RESIDUAL	72,5556	16	4,53472		
TOTAL (CORRECTED)	212,75	35			

Al realizar el análisis de varianza el porcentaje de elitismo tiene efectos significativos sobre el objetivo ponderado de éste problema.

Tabla 121

Análisis de varianza mejorado para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	10,2917	2	5,14583	1,33	0,2808
B:Porcentaje de elitismo	89,5417	2	44,7708	11,55	0,0002
C:Probabilidad de cruce	0,25	1	0,25	0,06	0,8013
D:Probabilidad de mutación	0,25	1	0,25	0,06	0,8013
INTERACTIONS					
RESIDUAL	112,417	29	3,87644		
TOTAL (CORRECTED)	212,75	35			

Después de realizar una mejora del análisis de varianza el mismo factor sigue teniendo efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 122

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	130,417			
Intensidad de mutación					
1	12	131,125	0,568363	129,963	132,287
2	12	130,292	0,568363	129,129	131,454
4	12	129,833	0,568363	128,671	130,996
Porcentaje de elitismo					
0	12	132,625	0,568363	131,463	133,787
25	12	129,583	0,568363	128,421	130,746
50	12	129,042	0,568363	127,879	130,204
Probabilidad de cruce					
60	18	130,5	0,464066	129,551	131,449
90	18	130,333	0,464066	129,384	131,282
Probabilidad de mutación					
10	18	130,5	0,464066	129,551	131,449
50	18	130,333	0,464066	129,384	131,282

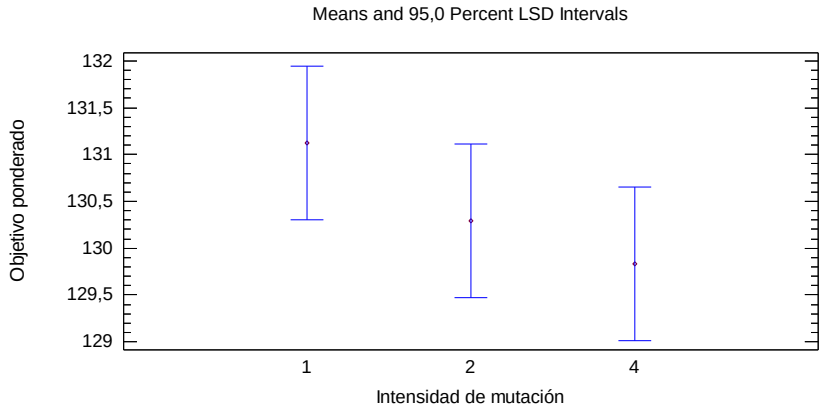


Figura 144. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

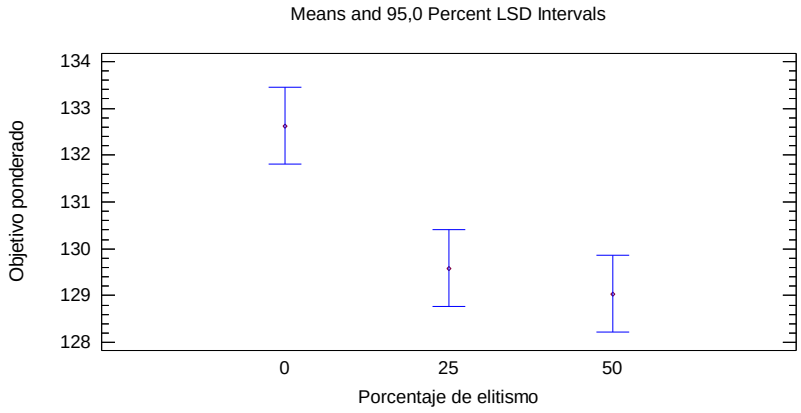


Figura 145. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

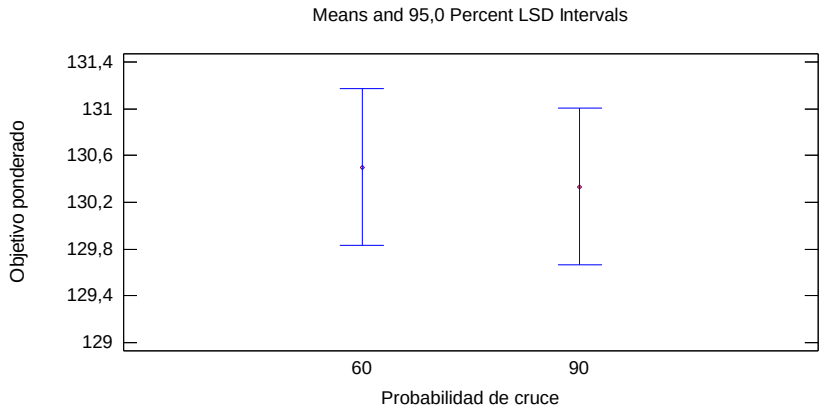


Figura 146. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

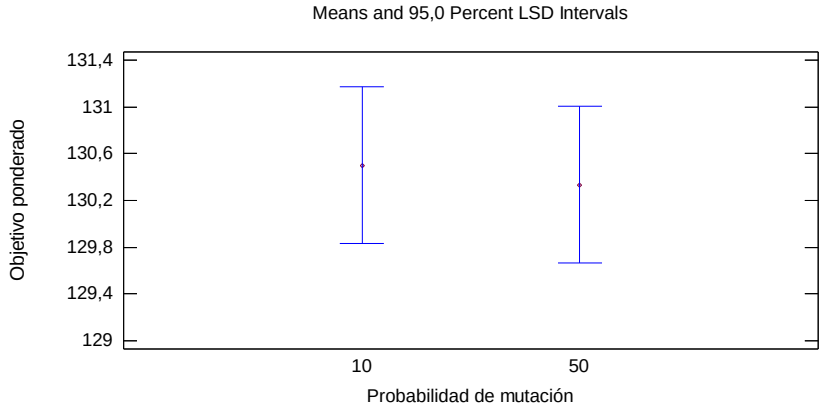


Figura 147. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

A continuación se muestra la validación de supuestos:

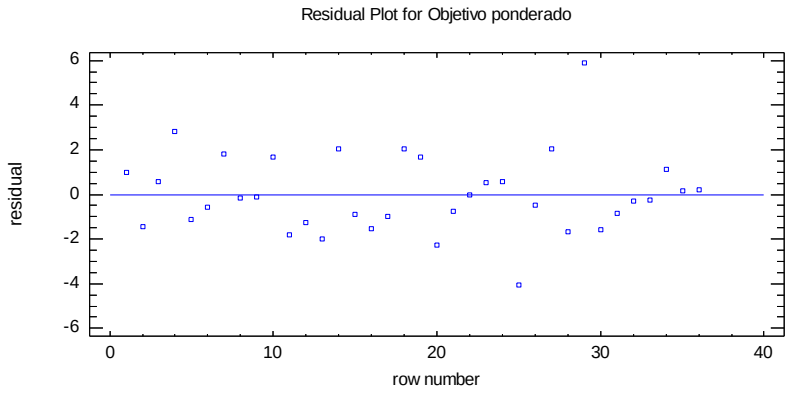


Figura 148. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9).

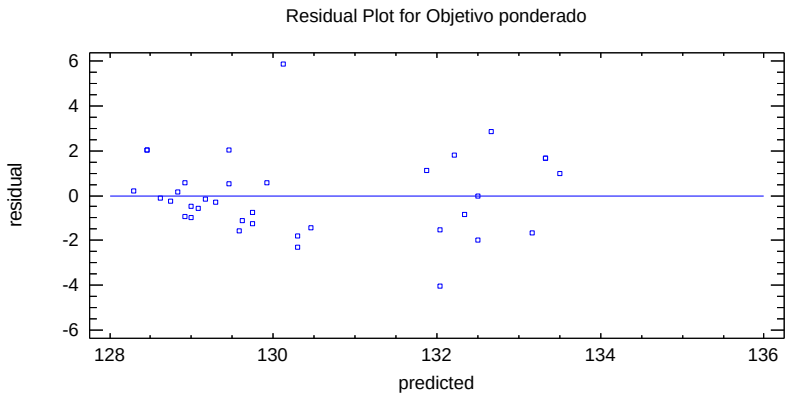


Figura 149. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9).

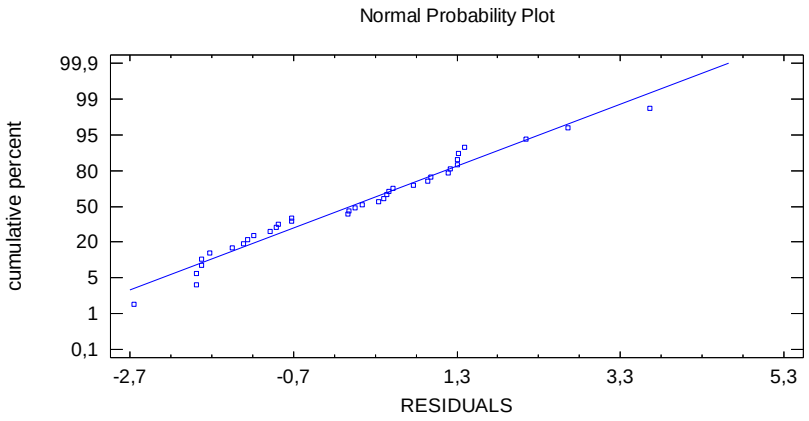


Figura 150. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 50% y β = 50% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9).

Diseño de experimentos para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9):

Tabla 123

Análisis de varianza para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema grande (Problema 9)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	2,73722	2	1,36861	0,29	0,7492
B:Porcentaje de elitismo	42,5706	2	21,2853	4,57	0,0269
C:Probabilidad de cruce	2,05444	1	2,05444	0,44	0,5159
D:Probabilidad de mutación	2,89	1	2,89	0,62	0,4423
INTERACTIONS					
AB	5,87444	4	1,46861	0,32	0,8634
AC	2,68389	2	1,34194	0,29	0,7534
AD	27,7217	2	13,8608	2,98	0,0796
BC	4,84056	2	2,42028	0,52	0,6043
BD	7,80167	2	3,90083	0,84	0,4507
CD	14,6944	1	14,6944	3,16	0,0946
RESIDUAL	74,4833	16	4,65521		
TOTAL (CORRECTED)	188,352	35			

Al realizar el análisis de varianza el porcentaje de elitismo tiene efectos significativos sobre el objetivo ponderado de éste problema.

Tabla 124

Análisis de varianza mejorado para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	2,73722	2	1,36861	0,33	0,7184
B:Porcentaje de elitismo	42,5706	2	21,2853	5,21	0,0122
C:Probabilidad de cruce	2,05444	1	2,05444	0,50	0,4845
D:Probabilidad de mutación	2,89	1	2,89	0,71	0,4078
INTERACTIONS					
AD	27,7217	2	13,8608	3,39	0,0486
RESIDUAL	110,378	27	4,08809		
TOTAL (CORRECTED)	188,352	35			

Después de realizar una mejora del análisis de varianza el mismo factor sigue teniendo efectos significativos sobre el objetivo, al igual que la interacción entre la intensidad de mutación y la probabilidad de mutación.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 125

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9) (Parte 1 de 2)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	155,672			
Intensidad de mutación					
1	12	156,017	0,583673	154,819	157,214
2	12	155,342	0,583673	154,144	156,539
4	12	155,658	0,583673	154,461	156,856
Porcentaje de elitismo					
0	12	157,092	0,583673	155,894	158,289
25	12	155,475	0,583673	154,277	156,673
50	12	154,45	0,583673	153,252	155,648
Probabilidad de cruce					

60	18	155,433	0,476567	154,455	156,411
90	18	155,911	0,476567	154,933	156,889

Tabla 126

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9) (Parte 2 de 2)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	155,672			
Probabilidad de mutación					
10	18	155,956	0,476567	154,978	156,933
50	18	155,389	0,476567	154,411	156,367
Intensidad de mutación by Probabilidad de mutación					
1,10	6	155,367	0,825438	153,673	157,06
1,50	6	156,667	0,825438	154,973	158,36
2,10	6	155,383	0,825438	153,69	157,077
2,50	6	155,3	0,825438	153,606	156,994
4,10	6	157,117	0,825438	155,423	158,81
4,50	6	154,2	0,825438	152,506	155,894

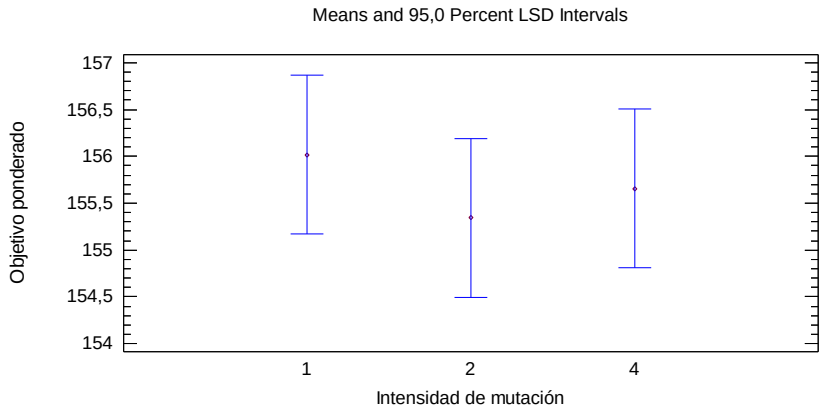


Figura 151. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema

grande (Problema 9).

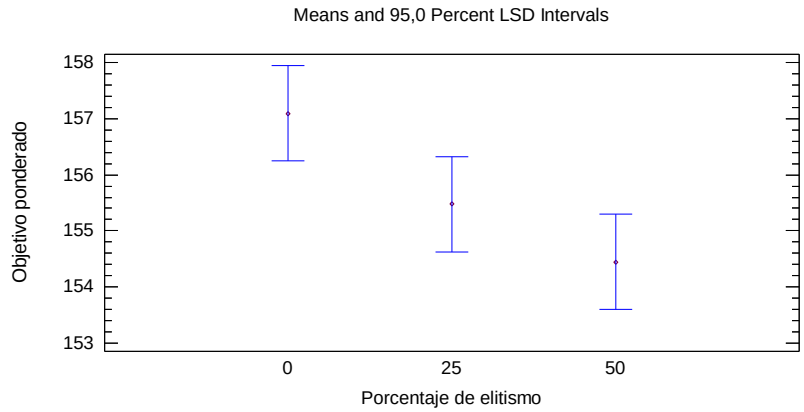


Figura 152. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

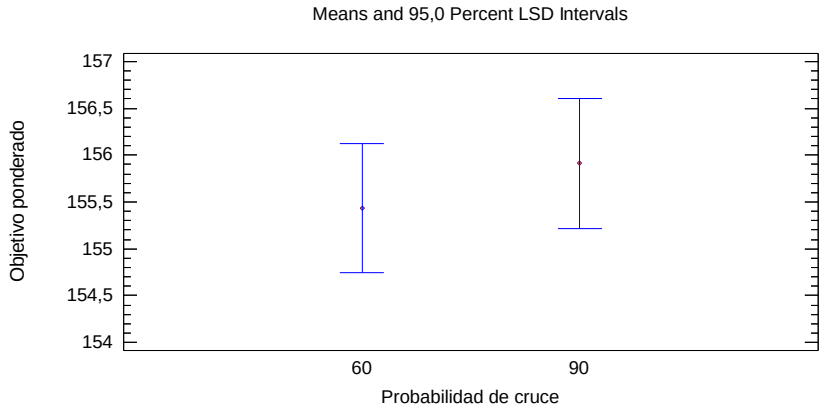


Figura 153. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

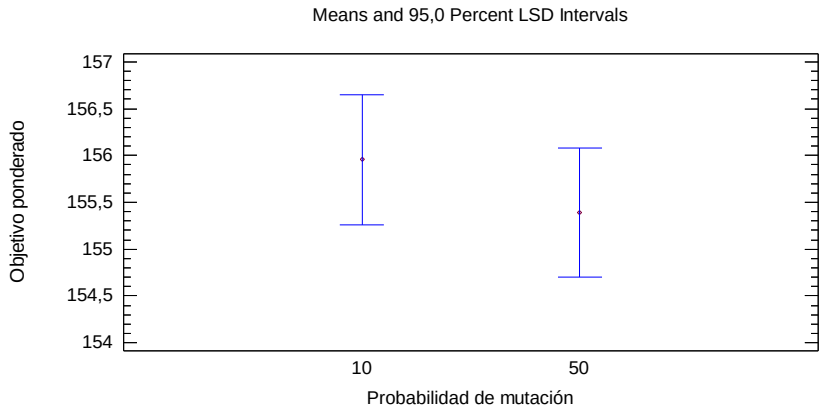


Figura 154. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9).

La gráfica de interacción de los factores que tienen relación se encuentra a continuación:

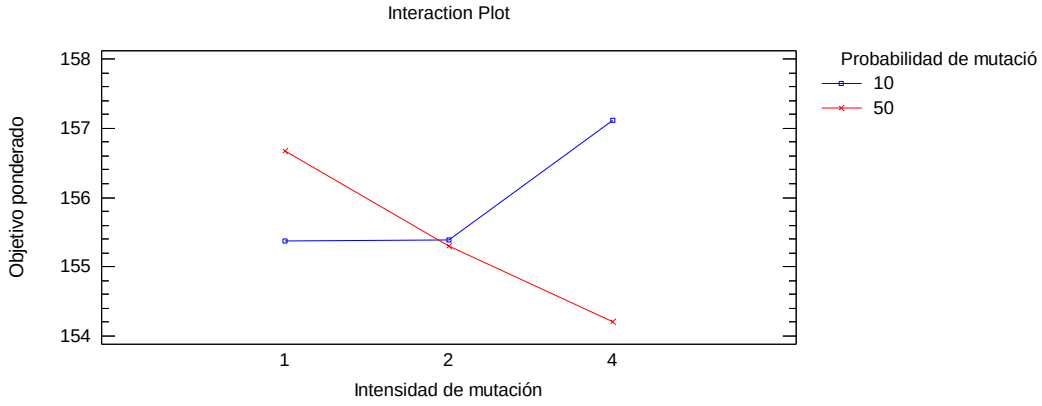


Figura 155. Gráfica de inteacción entre los factores intensidad de mutación y probabilidad de mutación para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9).

A continuación se muestra la validación de supuestos:

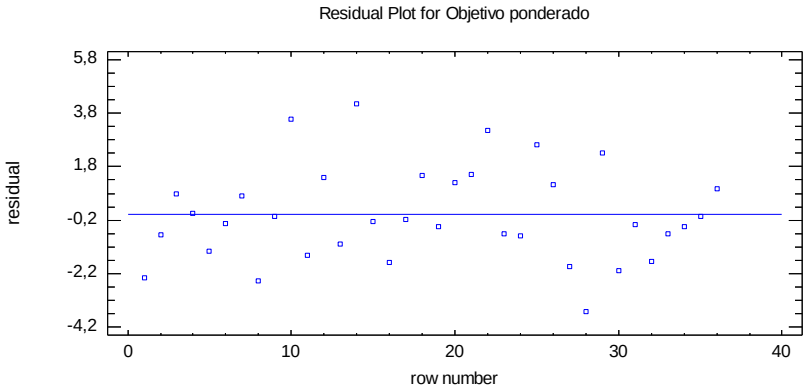


Figura 156. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

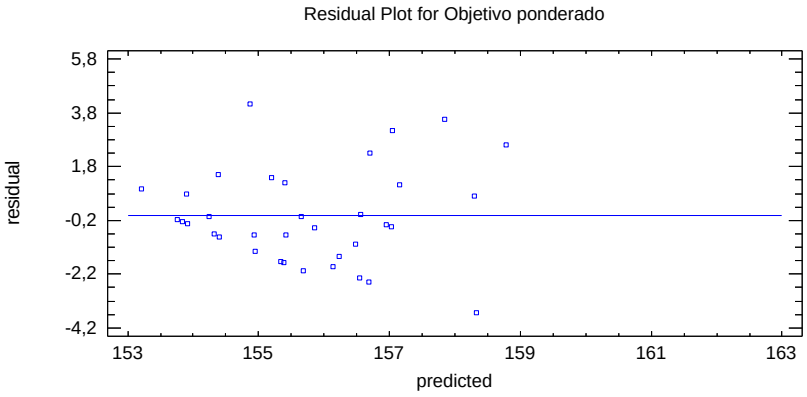


Figura 157. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

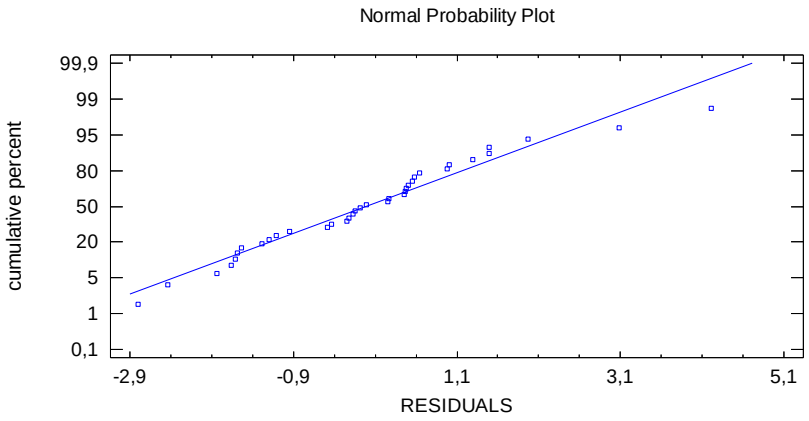


Figura 158. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 60% y β = 40% del Objetivo Ponderado $(\alpha\text{Makespan} + \beta\text{TTP})$ en el problema grande (Problema 9).

Diseño de experimentos para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado $(\boldsymbol{\alpha}\mathbf{Makespan} + \boldsymbol{\beta}\mathbf{TTP})$ en el problema grande (Problema 9):

Tabla 127

Análisis de varianza para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema grande (Problema 9)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	0,405	2	0,2025	0,10	0,9081
B:Porcentaje de elitismo	15,4067	2	7,70333	3,69	0,0482
C:Probabilidad de cruce	4,06694	1	4,06694	1,95	0,1820
D:Probabilidad de mutación	5,36694	1	5,36694	2,57	0,1285
INTERACTIONS					
AB	17,3083	4	4,32708	2,07	0,1324
AC	3,22389	2	1,61194	0,77	0,4787
AD	0,0272222	2	0,0136111	0,01	0,9935
BC	1,97556	2	0,987778	0,47	0,6317
BD	5,29556	2	2,64778	1,27	0,3083
CD	2,50694	1	2,50694	1,20	0,2895
RESIDUAL	33,4244	16	2,08903		
TOTAL (CORRECTED)	89,0075	35			

Al realizar el análisis de varianza el porcentaje de elitismo tiene efectos significativos sobre el objetivo ponderado de éste problema.

Tabla 128

Análisis de varianza mejorado para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	0,405	2	0,2025	0,09	0,9123
B:Porcentaje de elitismo	15,4067	2	7,70333	3,50	0,0434
C:Probabilidad de cruce	4,06694	1	4,06694	1,85	0,1843
D:Probabilidad de mutación	5,36694	1	5,36694	2,44	0,1291
INTERACTIONS					
RESIDUAL	63,7619	29	2,19869		
TOTAL (CORRECTED)	89,0075	35			

Después de realizar una mejora del análisis de varianza el mismo factor sigue teniendo efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 129

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	180,875			
Intensidad de mutación					
1	12	180,95	0,428047	180,075	181,825
2	12	180,725	0,428047	179,85	181,6
4	12	180,95	0,428047	180,075	181,825
Porcentaje de elitismo					
0	12	181,792	0,428047	180,916	182,667
25	12	180,308	0,428047	179,433	181,184
50	12	180,525	0,428047	179,65	181,4
Probabilidad de cruce					
60	18	180,539	0,349499	179,824	181,254
90	18	181,211	0,349499	180,496	181,926
Probabilidad de mutación					
10	18	181,261	0,349499	180,546	181,976
50	18	180,489	0,349499	179,774	181,204

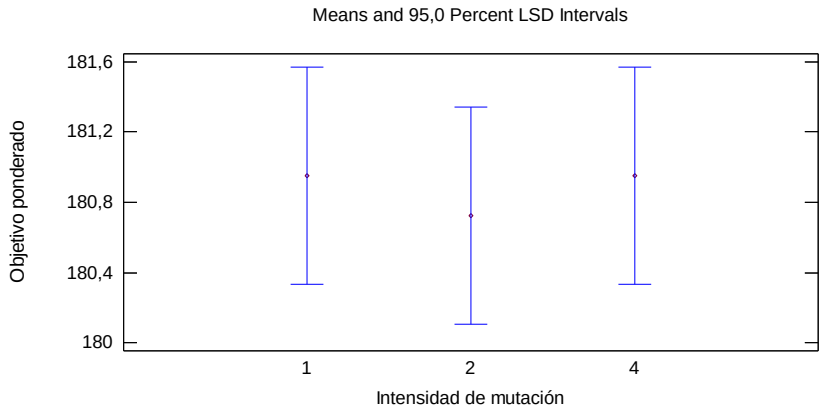


Figura 159. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado ($\alpha \mathrm{Makespan} + \beta \mathrm{TTP}$) en el problema grande (Problema 9).

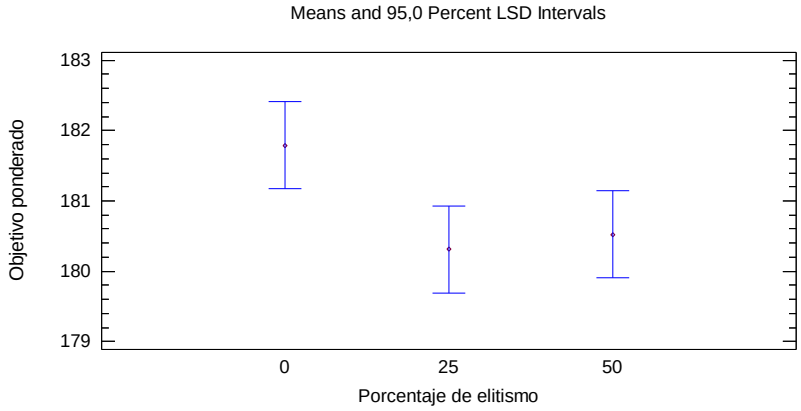


Figura 160. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado ($\alpha \mathrm{Makespan} + \beta \mathrm{TTP}$) en el problema grande (Problema 9).

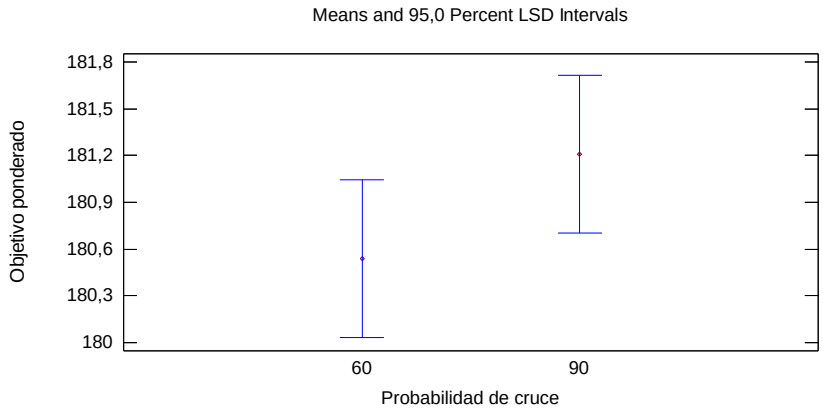


Figura 161. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

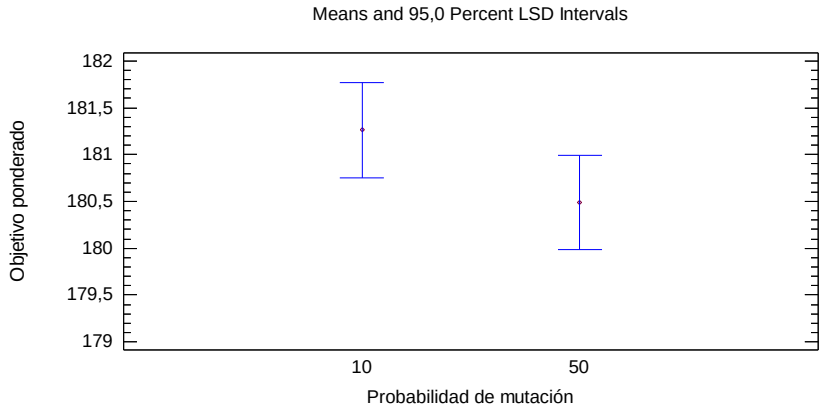


Figura 162. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

A continuación se muestra la validación de supuestos:

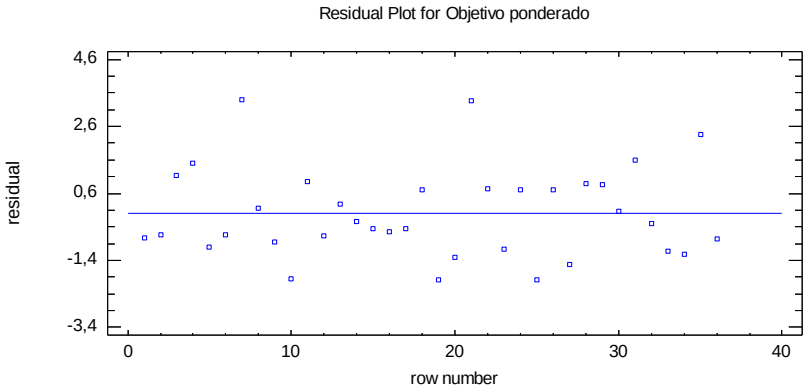


Figura 163. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

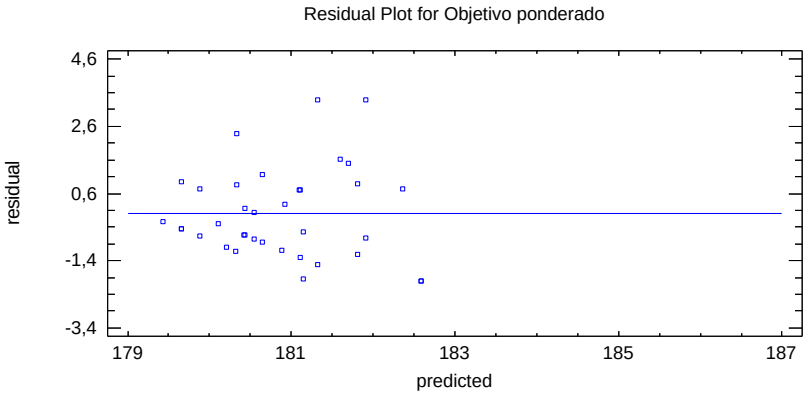


Figura 164. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

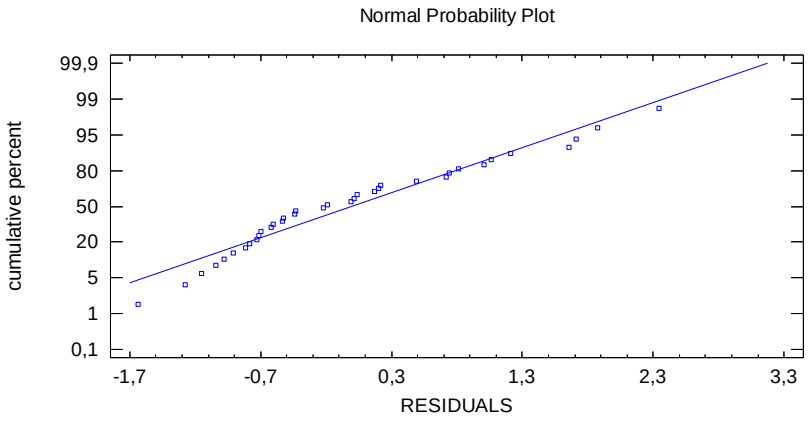


Figura 165. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 70% y β = 30% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9).

Diseño de experimentos para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado $(\alpha \text{Makespan} + \beta \text{TTP})$ en el problema grande (Problema 9):

Tabla 130

Análisis de varianza para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ en el problema grande (Problema 9)

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	1,97556	2	0,987778	0,53	0,5966
B:Porcentaje de elitismo	4,02889	2	2,01444	1,09	0,3605
C:Probabilidad de cruce	7,29	1	7,29	3,94	0,0646
D:Probabilidad de mutación	14,6944	1	14,6944	7,94	0,0124
INTERACTIONS					
AB	7,30444	4	1,82611	0,99	0,4428
AC	1,12667	2	0,563333	0,30	0,7418
AD	4,60222	2	2,30111	1,24	0,3149
BC	2,64667	2	1,32333	0,71	0,5043
BD	1,97556	2	0,987778	0,53	0,5966
CD	6,93444	1	6,93444	3,75	0,0708
RESIDUAL	29,62	16	1,85125		
TOTAL (CORRECTED)	82,1989	35			

Después de realizar el análisis de varianza la probabilidad de mutación tiene efectos significativos sobre el objetivo ponderado de éste problema.

Tabla 131

Análisis de varianza mejorado para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	1,97556	2	0,987778	0,53	0,5951
B:Porcentaje de elitismo	4,02889	2	2,01444	1,08	0,3536
C:Probabilidad de cruce	7,29	1	7,29	3,90	0,0579
D:Probabilidad de mutación	14,6944	1	14,6944	7,86	0,0089
INTERACTIONS					
RESIDUAL	54,21	29	1,86931		
TOTAL (CORRECTED)	82,1989	35			

Al realizar una mejora del análisis de varianza el mismo factor sigue teniendo efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 132

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	206,406			
Intensidad de mutación					
1	12	206,633	0,394685	205,826	207,441
2	12	206,083	0,394685	205,276	206,891
4	12	206,5	0,394685	205,693	207,307
Porcentaje de elitismo					
0	12	206,867	0,394685	206,059	207,674
25	12	206,083	0,394685	205,276	206,891
50	12	206,267	0,394685	205,459	207,074
Probabilidad de cruce					
60	18	205,956	0,322259	205,296	206,615
90	18	206,856	0,322259	206,196	207,515
Probabilidad de mutación					
10	18	207,044	0,322259	206,385	207,704
50	18	205,767	0,322259	205,108	206,426

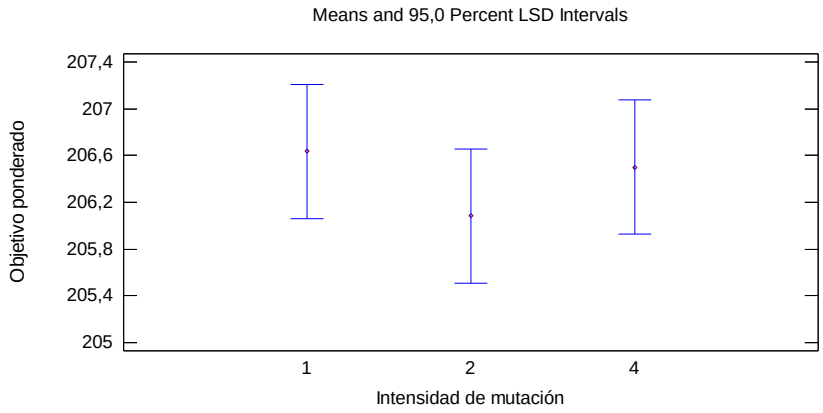


Figura 166. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

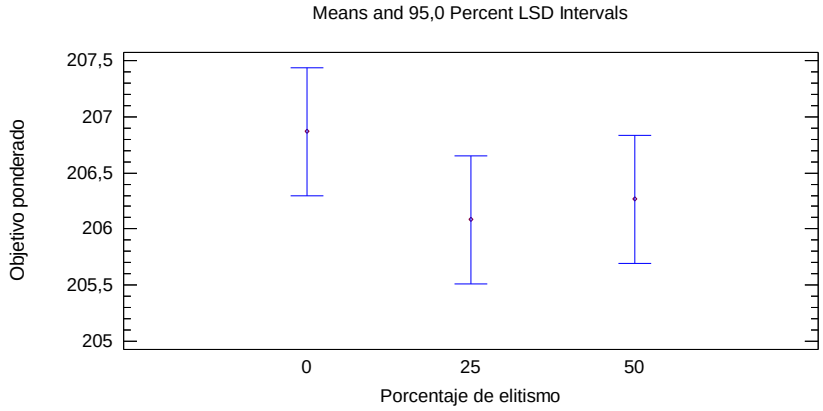


Figura 167. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

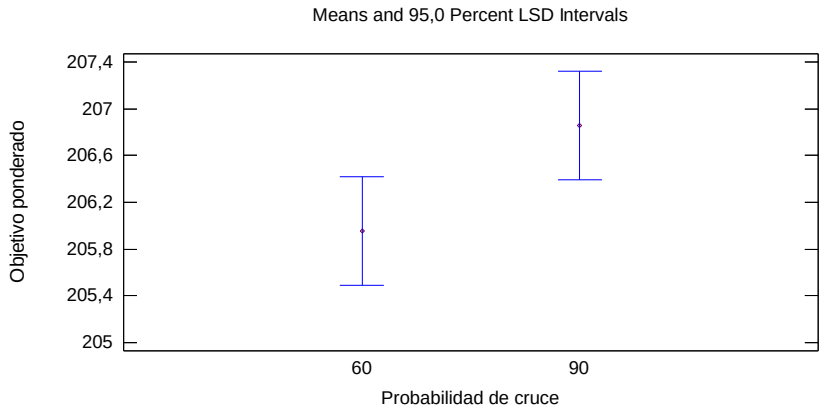


Figura 168. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

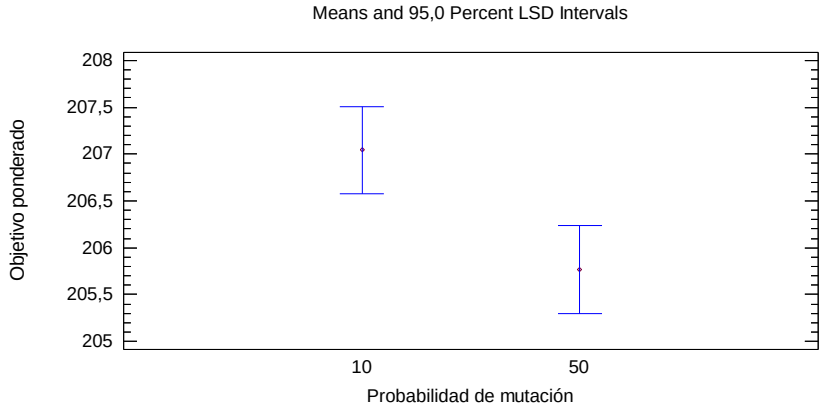


Figura 169. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

A continuación se muestra la validación de supuestos:

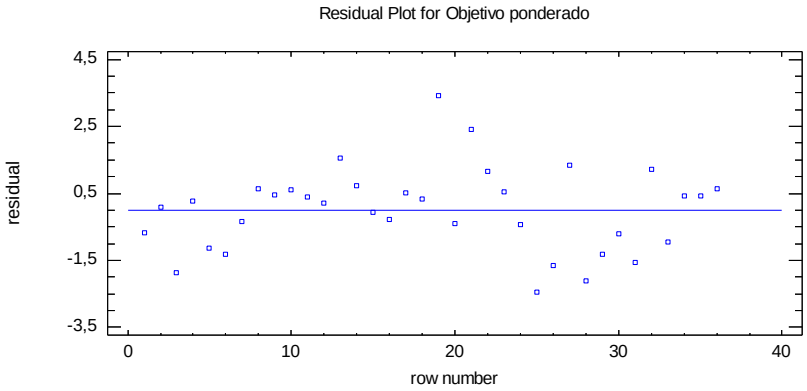


Figura 170. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9).

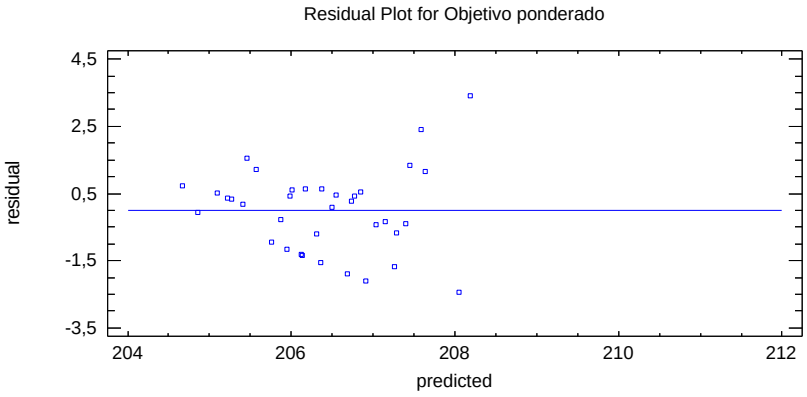


Figura 171. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9).

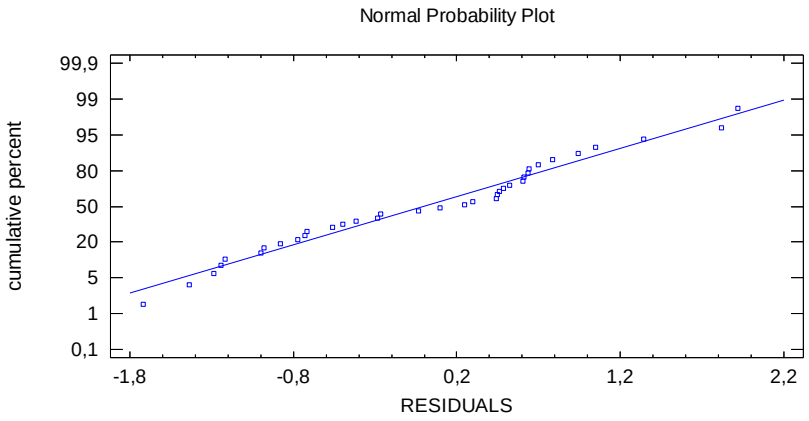


Figura 172. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 80% y β = 20% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9).

Diseño de experimentos para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado (αMakespan + βTTP) en el problema grande (Problema 9):

Tabla 133

Análisis de varianza para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	4,95389	2	2,47694	1,82	0,1933
B:Porcentaje de elitismo	0,310556	2	0,155278	0,11	0,8926
C:Probabilidad de cruce	15,21	1	15,21	11,20	0,0041
D:Probabilidad de mutación	0,587778	1	0,587778	0,43	0,5199
INTERACTIONS					
AB	7,56778	4	1,89194	1,39	0,2806
AC	7,835	2	3,9175	2,89	0,0851
AD	0,827222	2	0,413611	0,30	0,7415
BC	0,181667	2	0,0908333	0,07	0,9355
BD	0,650556	2	0,325278	0,24	0,7897
CD	1,13778	1	1,13778	0,84	0,3735
RESIDUAL	21,72	16	1,3575		
TOTAL (CORRECTED)	60,9822	35			

De acuerdo con el análisis de varianza el único factor que tiene efectos significativos sobre el objetivo ponderado de éste problema es la probabilidad de cruce.

Tabla 134

Análisis de varianza mejorado para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
MAIN EFFECTS					
A:Intensidad de mutación	4,95389	2	2,47694	1,80	0,1834
B:Porcentaje de elitismo	0,310556	2	0,155278	0,11	0,8937
C:Probabilidad de cruce	15,21	1	15,21	11,05	0,0024
D:Probabilidad de mutación	0,587778	1	0,587778	0,43	0,5186
INTERACTIONS					
RESIDUAL	39,92	29	1,37655		
TOTAL (CORRECTED)	60,9822	35			

Después de realizar una mejora del análisis de varianza el mismo factor sigue teniendo efectos significativos sobre el objetivo.

A continuación se presenta la tabla de medias y sus gráficas:

Tabla 135

Tabla de medias de los resultados obtenidos para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado $(\alpha Makespan + \beta TTP)$ *en el problema grande (Problema 9)*

			Stnd.	*Lower*	*Upper*
Level	*Count*	*Mean*	*Error*	*Limit*	*Limit*
GRAND MEAN	36	232,078			
Intensidad de mutación					
1	12	231,592	0,338693	230,899	232,284
2	12	232,492	0,338693	231,799	233,184
4	12	232,15	0,338693	231,457	232,843
Porcentaje de elitismo					
0	12	232,208	0,338693	231,516	232,901
25	12	232,0	0,338693	231,307	232,693
50	12	232,025	0,338693	231,332	232,718
Probabilidad de cruce					
60	18	231,428	0,276541	230,862	231,993
90	18	232,728	0,276541	232,162	233,293
Probabilidad de mutación					
10	18	232,206	0,276541	231,64	232,771
50	18	231,95	0,276541	231,384	232,516

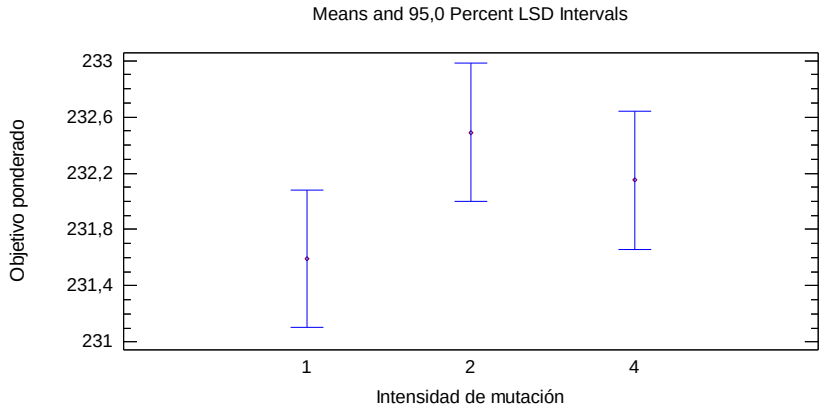


Figura 173. Gráfica de medias para el factor intensidad de mutación para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

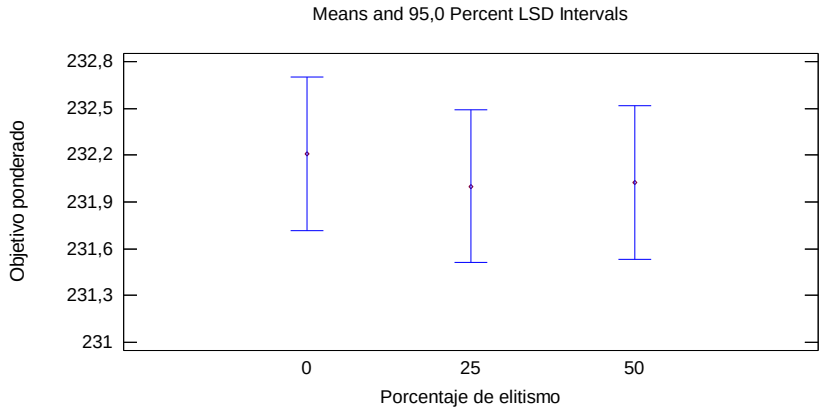


Figura 174. Gráfica de medias para el factor porcentaje de elitismo para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

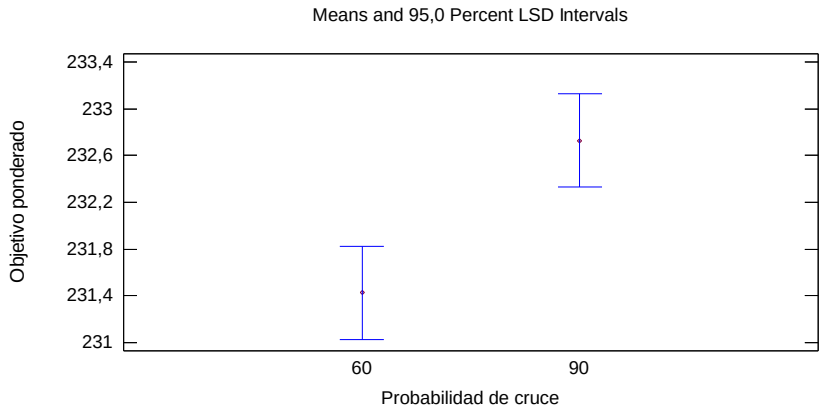


Figura 175. Gráfica de medias para el factor probabilidad de cruce para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

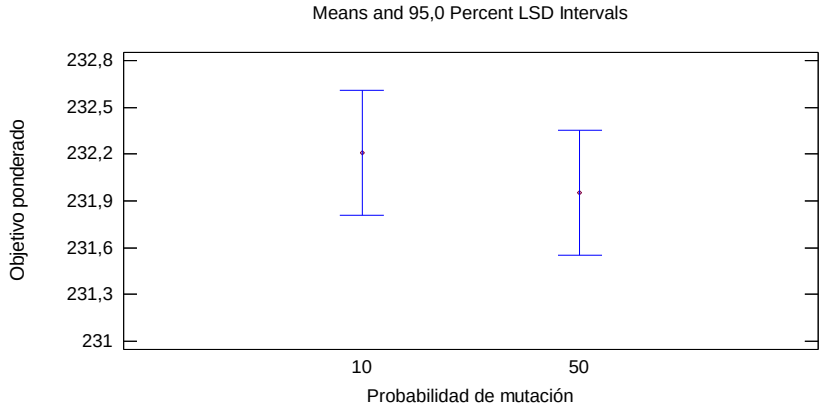


Figura 176. Gráfica de medias para el factor probabilidad de mutación para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

A continuación se muestra la validación de supuestos:

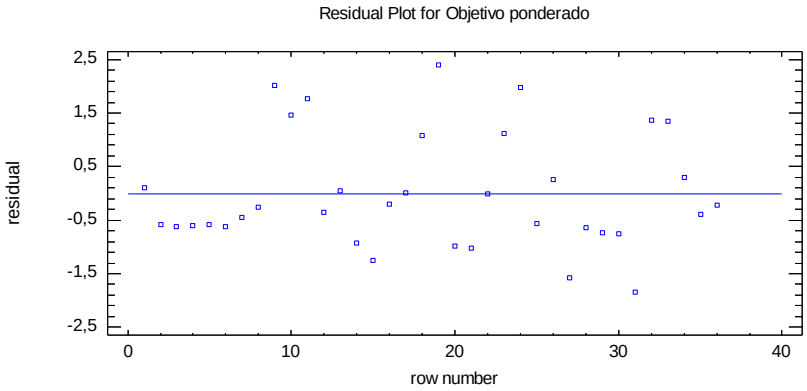


Figura 177. Gráfica de residuales para la validación del supuesto independencia o aleatoriedad para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

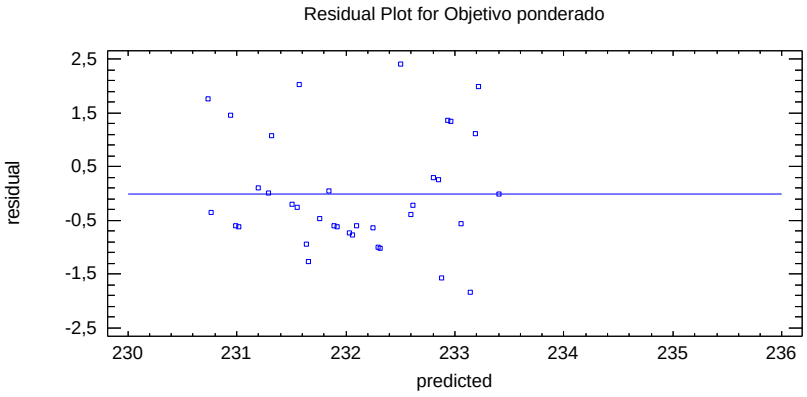


Figura 178. Gráfica de residuos vs predichos para la validación del supuesto de homocedasticidad para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

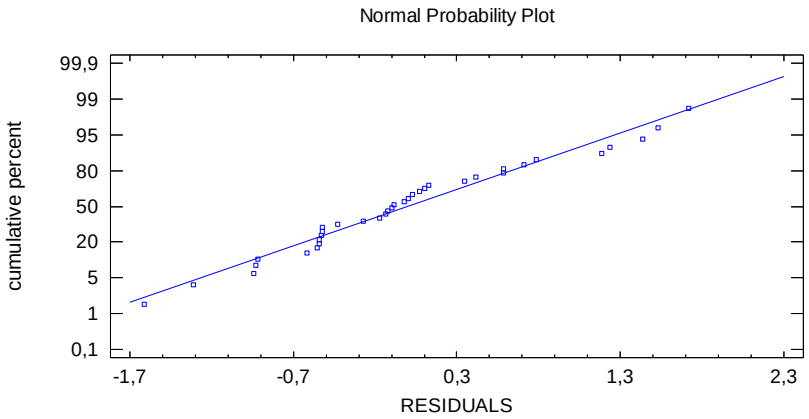


Figura 179. Gráfica de probabilidad normal para la validación del supuesto de normalidad para la evaluación de la combinación α = 90% y β = 10% del Objetivo Ponderado ($\alpha \text{Makespan} + \beta \text{TTP}$) en el problema grande (Problema 9).

ANEXO D.

RESULTADOS DE LAS CORRIDAS DE PRUEBA DEL ALGORITMO GENÉTICO PARA LAS DIFERENTES COMBINACIONES DE PONDERACIÓN DEL OBJETIVO PONDERADO

Tabla 136

Resultados de las corridas de prueba de los problemas pequeños para las diferentes combinaciones de ponderación del objetivo ponderado

						Problema 1				Problema 2				Problema 3			
Alfa	**Beta**	**Pc**	**Pm**	**Im**	**Pe1**	**Objetivo ponderado**	**Makespan**	**TTP**	**Gen**	**Objetivo ponderado**	**Makespan**	**TTP**	**Gen**	**Objetivo ponderado**	**Makespan**	**TTP**	**Gen**
0	**100**	90	50	4	50	0	436	0	0	0	324	0	0	157,7	716	1577	3
100	**0**	60	50	2	50	436	436	0	0	324	324	0	0	53,6	536	2418	3
50	**50**	60	50	4	50	218	436	0	0	162	324	0	0	111,45	604	1625	36
10	**90**	60	50	4	25	43,6	436	0	0	32,4	324	0	0	149,08	716	1577	27
20	**80**	90	50	4	25	87,2	436	0	0	64,8	324	0	0	140,06	675	1582	13
30	**70**	60	50	4	25	130,8	436	0	0	97,2	324	0	0	130,98	675	1582	143
40	**60**	90	50	2	25	174,4	436	0	0	129,6	324	0	0	121,66	604	1625	12
60	**40**	90	50	4	50	261,6	436	0	0	194,4	324	0	0	101,24	604	1625	36
70	**30**	60	10	4	50	305,2	436	0	0	226,8	324	0	0	92,96	572	1764	100
80	**20**	60	50	4	25	348,8	436	0	0	259,2	324	0	0	80,96	536	1904	125
90	**10**	90	10	4	25	392,4	436	0	0	291,6	324	0	0	68,62	536	2038	162

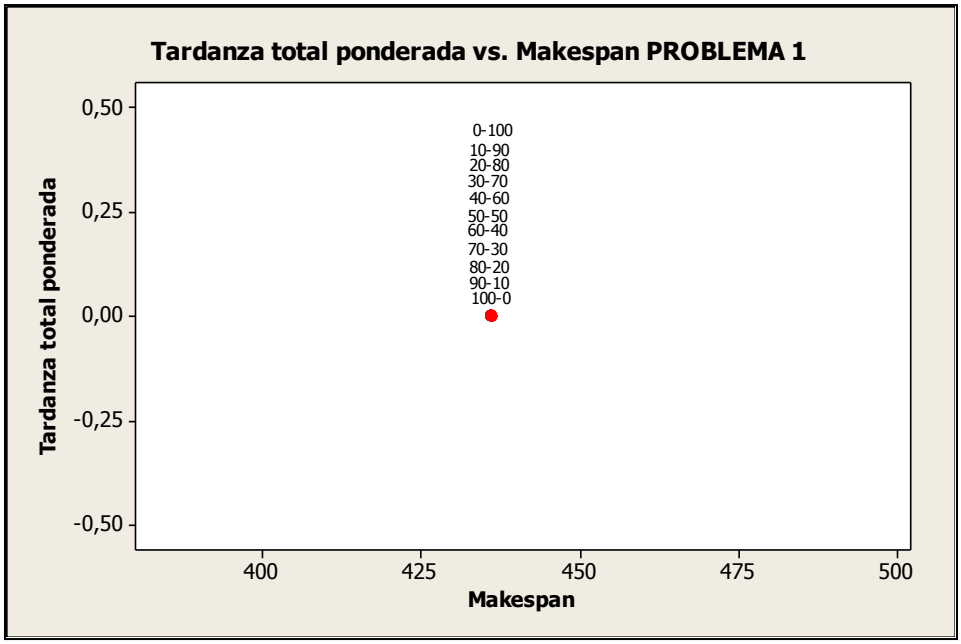


Figura 180. Gráfica de pareto de los resultados de las corridas de prueba del problema 1 para las diferentes combinaciones de ponderación del objetivo ponderado.

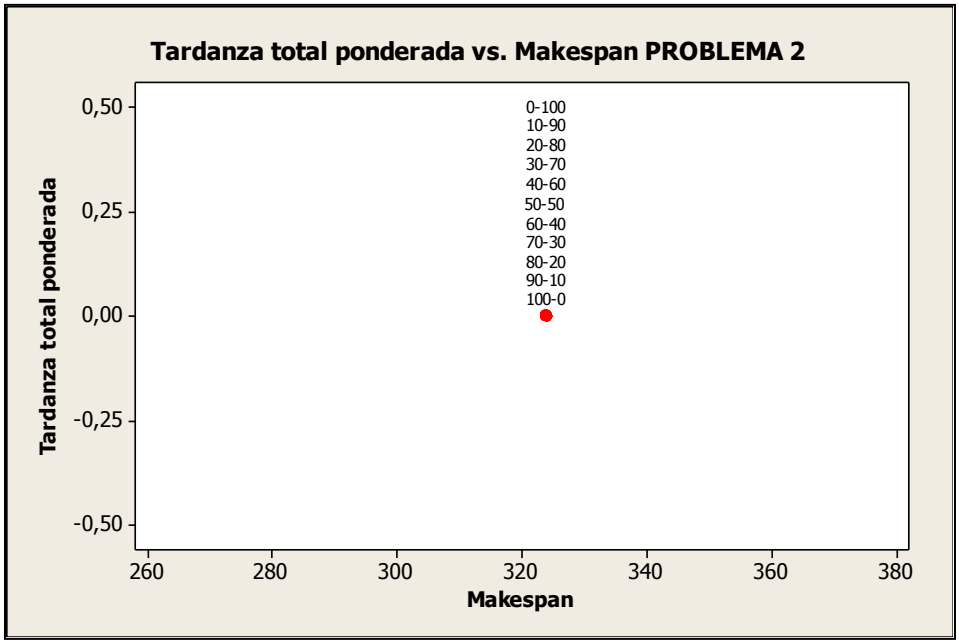


Figura 181. Gráfica de pareto de los resultados de las corridas de prueba del problema 2 para las diferentes combinaciones de ponderación del objetivo ponderado.

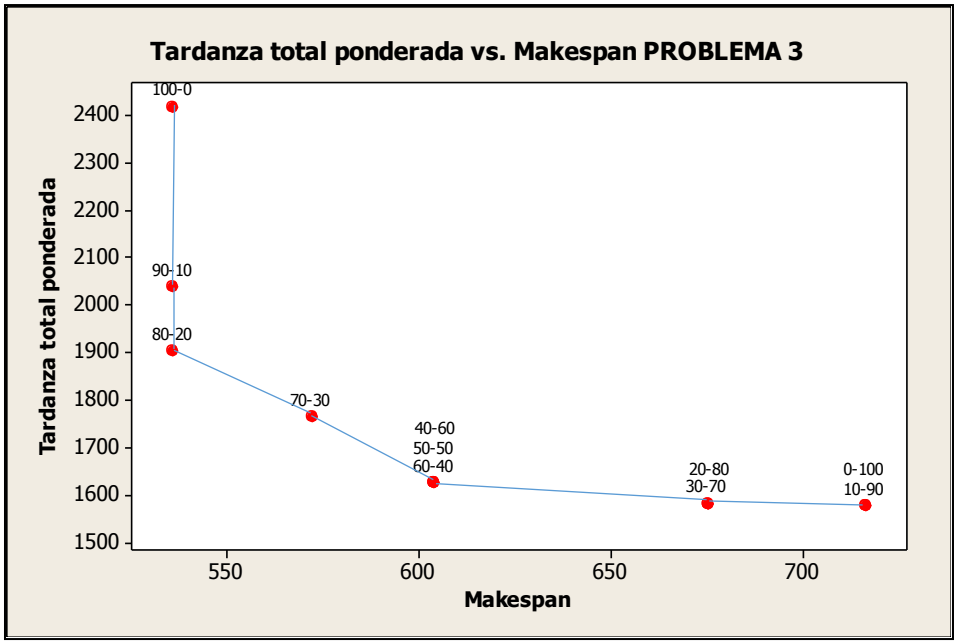


Figura 182. Gráfica de pareto de los resultados de las corridas de prueba del problema 3 para las diferentes combinaciones de ponderación del objetivo ponderado.

Tabla 137

Resultados de las corridas de prueba de los problemas medianos para las diferentes combinaciones de ponderación del objetivo ponderado

						Problema 4				Problema 5			
Alfa	**Beta**	**Pc**	**Pm**	**Im**	**Pe1**	**Objetivo Ponderado**	**Makespan**	**TTP**	**Gen**	**Objetivo ponderado**	**Makespan**	**TTP**	**Gen**
0	**100**	90	50	4	50	40	445	40	27	155	460	155	56
100	**0**	60	50	2	50	394	394	228	27	379	379	1014	56
50	**50**	60	50	4	50	242,5	445	40	10	305	441	169	74
10	**90**	60	50	4	25	80,5	445	40	80	196,1	467	166	24
20	**80**	90	50	4	25	121	445	40	11	216	460	155	69
30	**70**	60	50	4	25	161,5	445	40	38	246,5	460	155	20
40	**60**	90	50	2	25	202	445	40	39	283,6	466	162	45
60	**40**	90	50	4	50	283	445	40	19	332,2	441	169	15
70	**30**	60	10	4	50	318,5	407	112	1	358,6	434	183	8
80	**20**	60	50	4	25	344,8	394	148	14	381,4	399	311	131
90	**10**	90	10	4	25	370	394	154	56	387	390	360	16
						Problema 6				**Problema 7**			
Alfa	**Beta**	**Pc**	**Pm**	**Im**	**Pe1**	**Objetivo ponderado**	**Makespan**	**TTP**	**Gen**	**Objetivo ponderado**	**Makespan**	**TTP**	**Gen**
0	**100**	90	50	4	50	0	332	0	0	546	1059	546	168
100	**0**	60	50	2	50	332	332	0	0	994	994	813	11
50	**50**	60	50	4	50	166	332	0	0	826	1087	565	18
10	**90**	60	50	4	25	33,2	332	0	0	563,5	1135	500	33
20	**80**	90	50	4	25	66,4	332	0	0	636,6	1031	538	67
30	**70**	60	50	4	25	99,6	332	0	0	685,9	1031	538	101
40	**60**	90	50	2	25	132,8	332	0	0	802,6	1072	623	13
60	**40**	90	50	4	50	199,2	332	0	0	833,8	1031	538	32
70	**30**	60	10	4	50	232,4	332	0	0	936,7	1075	714	6
80	**20**	60	50	4	25	265,6	332	0	0	563,5	1135	500	33
90	**10**	90	10	4	25	298,8	332	0	0	971,1	994	765	15

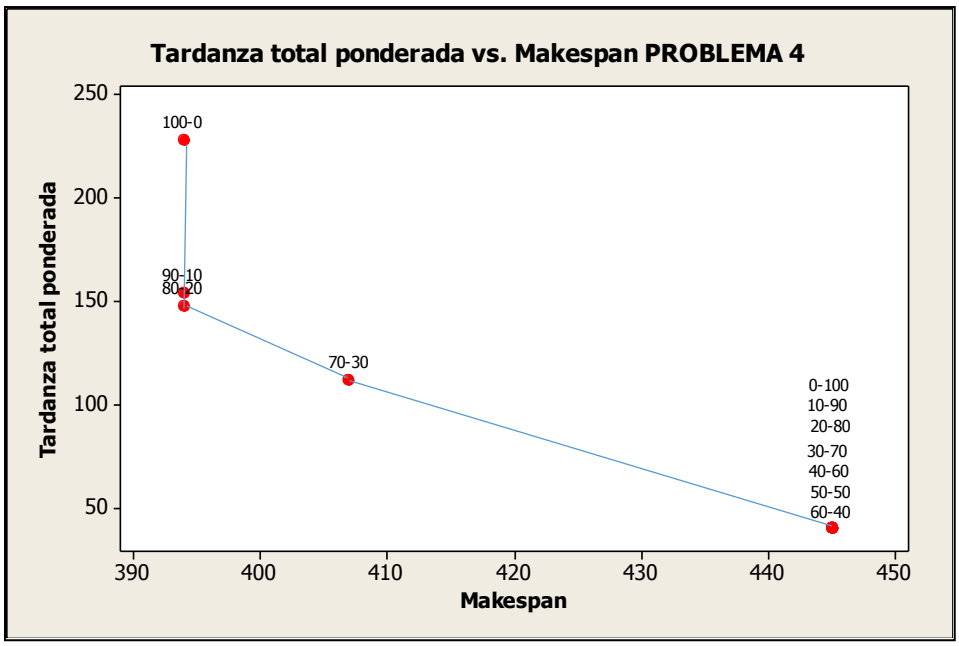


Figura 183. Gráfica de pareto de los resultados de las corridas de prueba del problema 4 para las diferentes combinaciones de ponderación del objetivo ponderado.

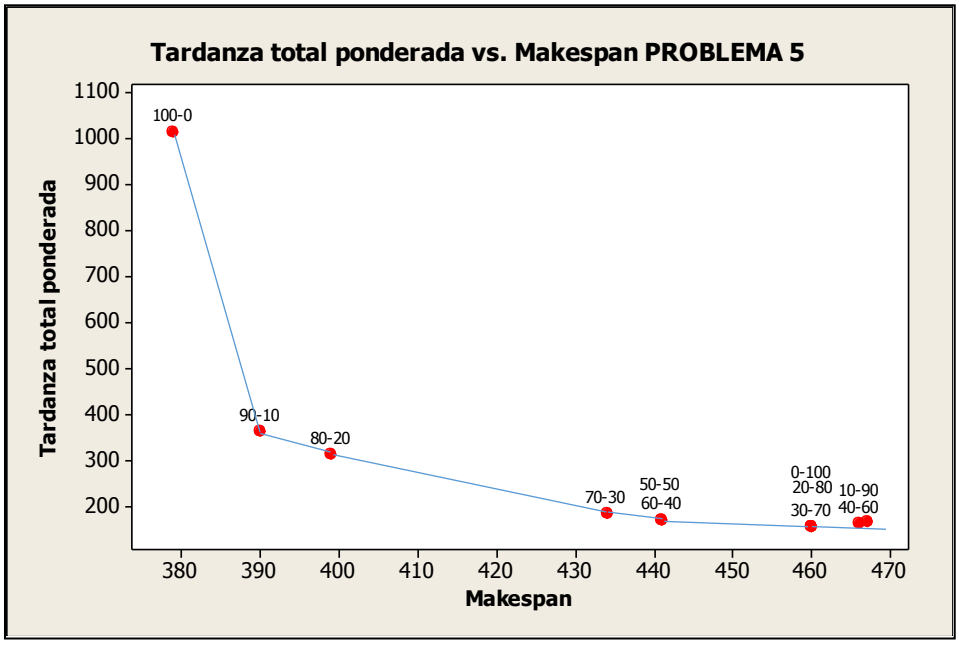


Figura 184. Gráfica de pareto de los resultados de las corridas de prueba del problema 5 para las diferentes combinaciones de ponderación del objetivo ponderado.

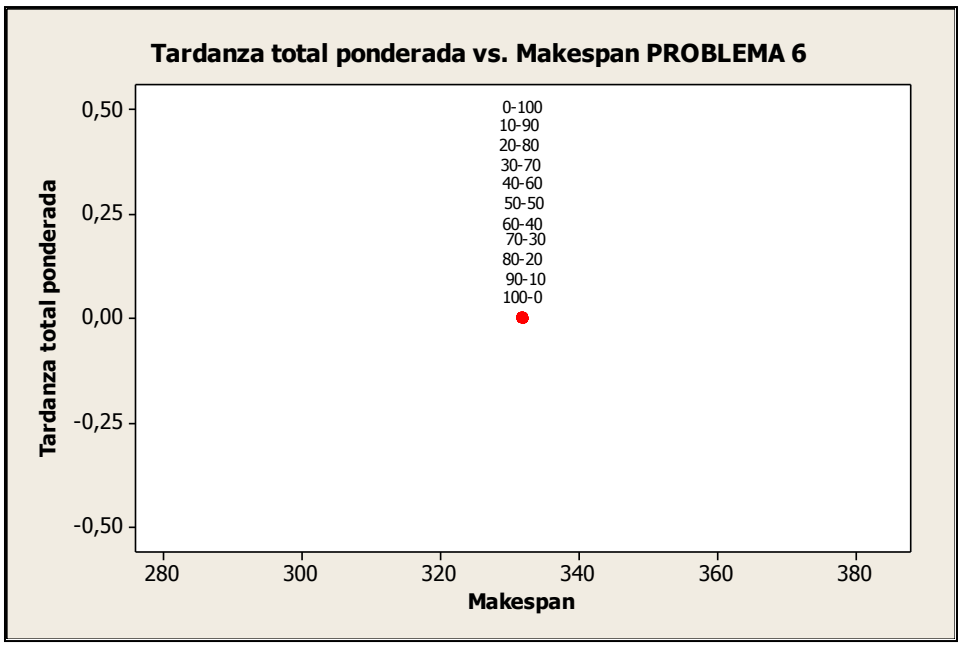


Figura 185. Gráfica de pareto de los resultados de las corridas de prueba del problema 6 para las diferentes combinaciones de ponderación del objetivo ponderado.

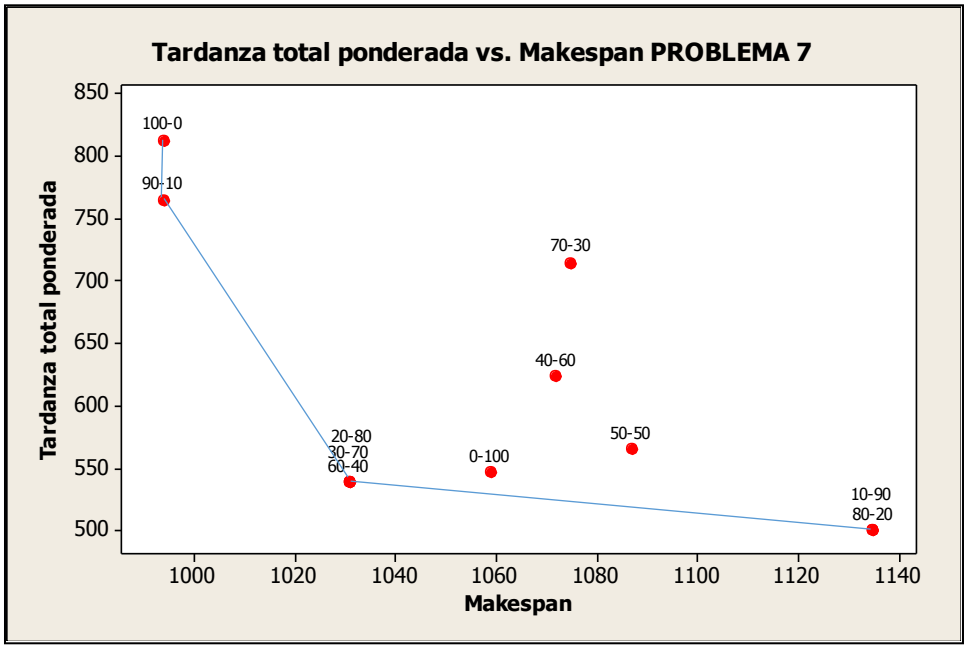


Figura 186. Gráfica de pareto de los resultados de las corridas de prueba del problema 7 para las diferentes combinaciones de ponderación del objetivo ponderado.

Tabla 138

Resultados de las corridas de prueba de los problemas grandes para las diferentes combinaciones de ponderación del objetivo ponderado

						Problema 8				Problema 9				Problema 10			
Alfa	**Beta**	**Pc**	**Pm**	**Im**	**Pe1**	**Objetivo ponderado**	**Makespan**	**TTP**	**Gen**	**Objetivo ponderado**	**Makespan**	**TTP**	**Gen**	**Objetivo ponderado**	**Makespan**	**TTP**	**Gen**
0	**100**	90	50	4	50	306	556	306	199	0	291	0	0	7147	762	7147	197
100	**0**	60	50	2	0	459	459	1043	199	250	250	34	5	753	753	11380	197
50	**50**	90	50	4	50	367,5	578	157	182	128,5	257	0	38	3609,5	762	6457	162
10	**90**	60	50	4	25	223,7	590	183	170	26,1	261	0	59	5653,5	762	6197	197
20	**80**	60	50	4	50	328	564	269	166	52,2	261	0	39	5435,2	784	6598	196
30	**70**	60	50	4	50	369,4	578	280	97	76,8	256	0	85	4746	789	6442	199
40	**60**	90	50	4	50	424	607	302	90	102,4	256	0	91	4102,6	775	6321	190
60	**40**	60	50	4	50	424,4	584	185	87	154,2	257	0	59	3120,4	784	6625	47
70	**30**	60	50	2	25	465,6	555	257	66	179,2	256	0	92	2541,5	782	6647	61
80	**20**	60	50	2	25	494,8	483	542	98	205,4	256	3	179	1920,2	762	6553	197
90	**10**	60	50	1	25	487,1	453	794	148	232,5	257	12	3	1390	762	7042	82

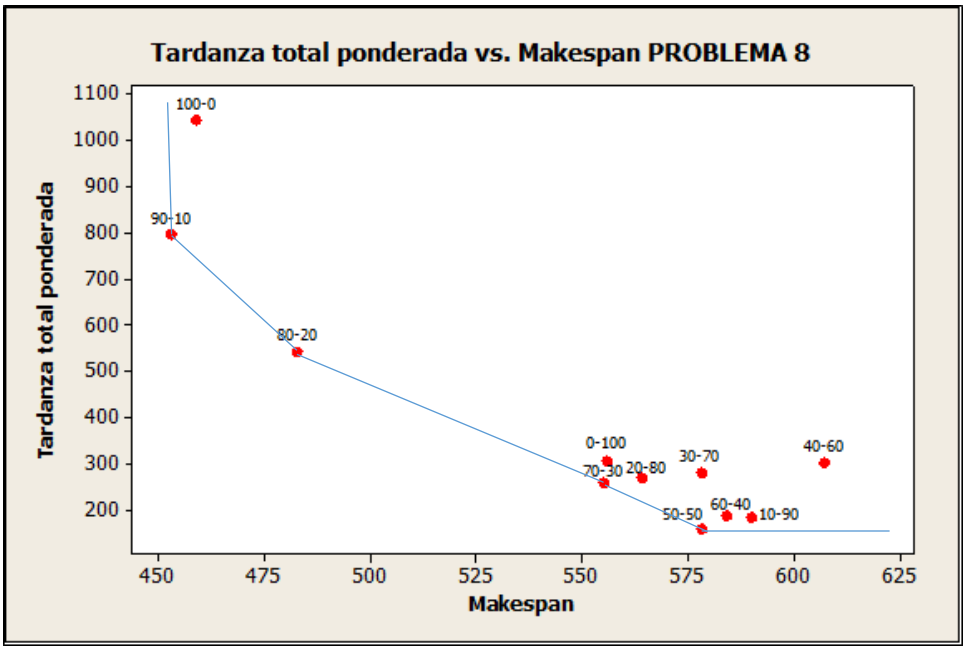


Figura 187. Gráfica de pareto de los resultados de las corridas de prueba del problema 8 para las diferentes combinaciones de ponderación del objetivo ponderado.

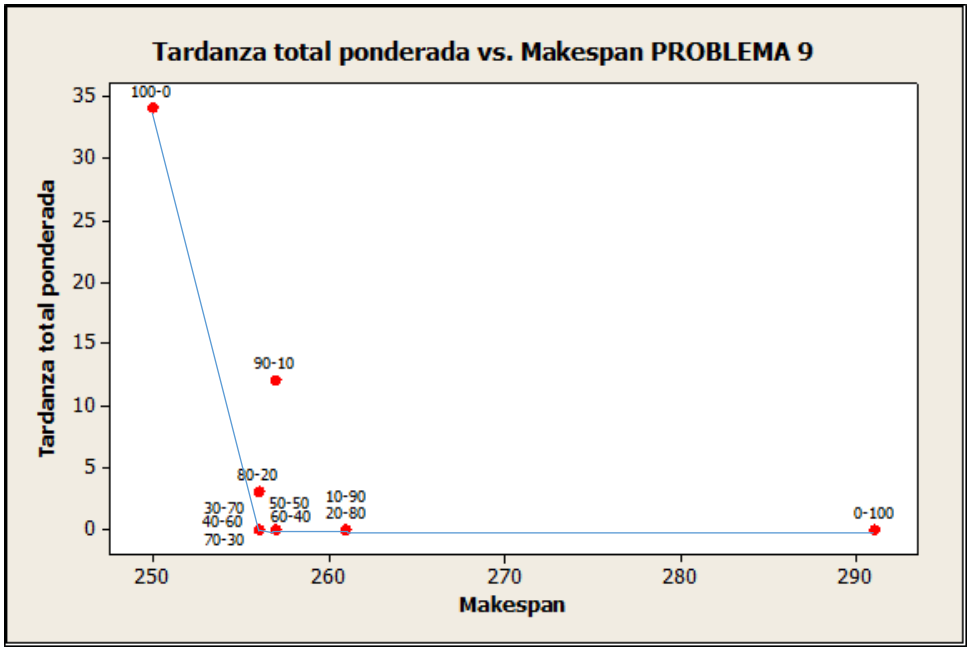


Figura 188. Gráfica de pareto de los resultados de las corridas de prueba del problema 9 para las diferentes combinaciones de ponderación del objetivo ponderado.

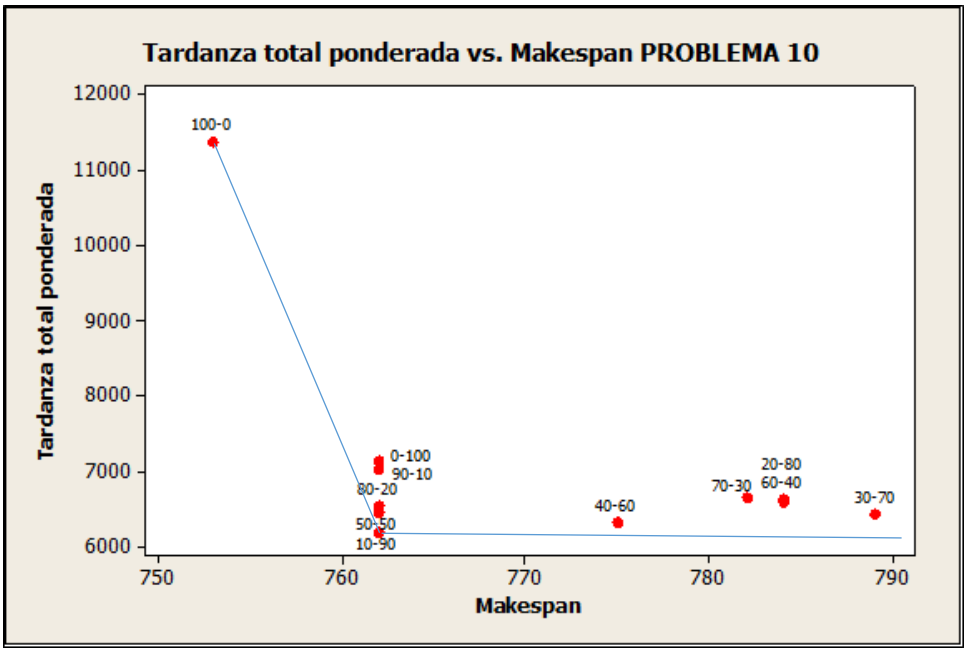


Figura 189. Gráfica de pareto de los resultados de las corridas de prueba del problema 10 para las diferentes combinaciones de ponderación del objetivo ponderado.

ANEXO E.

CÓDIGO DE PROGRAMACIÓN PetNMA

---------- **main.cpp** ----------

```
#include <iostream>
#include "Tjob.h"
#include "WkCenter.h"
#include "PetriNet.h"
#include "Schedule.h"
#include "Memetic.h"
#include "Individual.h"
#include <codecogs/statistics/distributions/continuous/stdnormal/cdf_inv.h>
#include <cdf_inv.h>
using namespace Stats::Dists::Continuous::LogNormal;

#pragma warning (disable : 4786)
#pragma warning (disable : 4076)
#pragma warning (disable : 4083)

#include <stdlib.h>
#include <stdio.h>
#include <vector>
#include <conio.h>
#include <fstream>
#include <math.h>
#include <random>
#include <ctime>



using namespace std;
```

```
void generateMachineBreakdown (PetriNet& pn, int nStations,  int tnow, vector<intvec> marking,
vector<WkCenter>& wkcArray, intvec& timeU, deque<int>& u, vec2dim& operEndTime, vec2dim&
machBreakDown, const int Pavg, int& mainCont2);


void jobOperationIndex(const intvec& um ,const vec2dim& transitions, int& oper, int i, int
numJob);


void generateMachineBreakdown (PetriNet& pn, int nStations,  int tnow, vector<intvec> marking,
vector<WkCenter>& wkcArray, intvec& timeU, deque<int>& u, vec2dim& operEndTime, int& nbreaks,
vec2dim& machBreakDown, const int Pavg, int& mainCont2) {


        int breakTime;

        int breakStation;

        //Estación en la que se encuentra la máquina que falla

        int shootTransition;

        int lastShootedTransition;

        //Última transición disparada antes de la falla

        int breakMachine;

        //Máquina que falla

        int operbreakdown;

        //Operación que se ve interrumpida por la falla

        int transitionBreakDown;

        int BreakJob;

        //Trabajo al que pertenece la operación que se ve interrumpida por la falla

        int alfa = 1.4;

        double breakDownTime;

        //Tiempo de falla o reparación

        double gamma1;

        double gamma2;

        double p;

        double m;

        double s;
```

```
vector<intvec> bussyPlaces;//Plazas ocupadas


srand(time(NULL));
//rand().seed();

vec2dim conp;
vec2dim Cmin;
vec2dim Cplus;


srand(time(NULL));


nbreaks = 1;


machBreakDown.resize(nbreaks);



for(int i=0; i<nbreaks; i++){//Instante en que sucede la falla de máquina
        machBreakDown[i].resize(10);
        int instante;
        int instante1;
        if(mainCont2==0){
                instante=0;
                instante1=tnow/2;
                gamma1 = 0.5;
                gamma2 = 0.02;
        }else if(mainCont2==1){
                instante=tnow/2;
                instante1=tnow;
                gamma1 = 0.5;
                gamma2 = 0.05;
        }else if(mainCont2==2){
                instante=tnow/2;
                instante1=tnow;
```

```
                gamma1 = 1.0;
                gamma2 = 0.02;
        }else if(mainCont2==3){
                instante=0;
                instante1=tnow/2;
                gamma1 = 1.0;
                gamma2 = 0.05;
        }

        breakTime = instante + rand()%(instante1-instante);
        machBreakDown[i][0]=breakTime;
}

if(nbreaks > 1){
        int p=0;
        do{
                if( machBreakDown[p][0] < machBreakDown[p+1][0]){
                        p++;
                }else{
                        int instante = tnow/2;
                        breakTime = rand()%instante;
                        machBreakDown[p+1][0]=breakTime;
                }
        }while(p<nbreaks-1);
}

for(int k=0;k<nbreaks; k++){//Ciclo para n machBreakDown

        int nMachines = 0;
        intvec bussymachines;
        intvec operationsBreakdown;
```

```
intvec transitions;
intvec transitions2;
intvec operations;

for(int i = 0; i< marking.size(); i++){
//Se identifica la transición disparada antes de la falla de la máquina
	if(machBreakDown[k][0]>=marking[i][marking[i].size()-1]){
		if(machBreakDown[k][0]<marking[i+1][marking[i].size()-1]){
			shootTransition = i;
			lastShootedTransition = marking[i][marking[i].size()-1];
		}
	}
}

for(int i = 0; i<nStations; i ++){
//Ciclo que cálcula el número de máquinas en el sistema
	for(int j = 0; j< wkcArray[i].getNumMch(); j++){
		nMachines++;
	}
}

	for(int i = pn.getMp().size() - (2*nMachines); i<pn.getMp().size()-nMachines;
i++){//Ciclo que identifica las máquinas en el sistema que estaban ocupadas en el momento de
la falla
		if(marking[shootTransition][i] == 0){
			bussymachines.push_back(i);
		}
	}
}

breakMachine = bussymachines[rand()%bussymachines.size()];
machBreakDown[k][1]= breakMachine;

int repairMachine = breakMachine + nMachines;
//Plaza de reparación de la máquina que falla
```

```
int num = rand() % 2;


double double_rand();
double double_rand_sin_uno();


//Tiempo medio de reparación
//gamma1 = 1.0;
//0.5, 1.00


//Varianza del tiempo de reparación de la máquina
//gamma2 = 0.05;
//0.02, 0.05



p = (rand() / (double)RAND_MAX + 0.0);
m = log(Pavg*gamma1);
s = gamma2;


breakDownTime = CDF_inv(p,m,s);
ceil(breakDownTime);




machBreakDown[k][2]= ceil(breakDownTime);


for(int i = 0; i<pn.getMp().size() - nMachines; i++){
        if(marking[shootTransition][i] == 1){
                operations.push_back(i);
        }
}


for(int i=0; i<operations.size();i++){
```

```
        for(int j = 0; j< pn.getCplus().size(); j++){
                if(operations[i] == pn.getCplus()[j][0]){
                        transitions2.push_back(j);
                }
        }
}

for(int i=0; i<u.size();i++){
        if(pn.getCmin()[u[i]].size() > 1){
                if(machBreakDown[k][1] == allMchNo(pn,u[i])){
                        if(timeU[i] <= machBreakDown[k][0]){
                                transitions.push_back(u[i]);
                                //Transición que me dispara la máquina que falla
                        }
                }
        }
}

int cont4 = 0;

for(int i=0; i<transitions.size(); i++){
        for(int j=0; j<transitions2.size(); j++){
                if(transitions[i] == transitions2[j]){
                        cont4 += 1;
                        transitionBreakDown = transitions2[j];
                }
        }
}

if(cont4 == 0){
        break;
}
machBreakDown[k][4] = transitionBreakDown;
```

```
                //Transición que toca volver a disparar (Operación que toca rehacer)
                machBreakDown[k][6] = transitionBreakDown+2;
                //Transición a disparar para mandar la máquina a reparación
                machBreakDown[k][7] = pn.getCmin()[transitionBreakDown][0];
                //Plaza a la que regresa el token (Inicio de operación)
                machBreakDown[k][8] = pn.getCplus()[transitionBreakDown][0];
                //Plaza a la que toca remover el token (Operación dañada)
                BreakJob = jobNo(pn,transitionBreakDown);
                //Trabajo al que pertenece la operación que se ve interrumpida por la falla

                for(int j = 0; j < operEndTime[BreakJob].size(); j++){
                //Operación que se ve interrumpida por la falla
                        if(operEndTime[BreakJob][j] > machBreakDown[k][0]){
                                operationsBreakdown.push_back(j);
                        }
                }

                machBreakDown[k][3] = operationsBreakdown[0];
                machBreakDown[k][5] = BreakJob;
                machBreakDown[k][9] = repairMachine;
        }

}

int main(int argc, char* argv[]){

        //Se inicia la programación
        //Se declaran los vectores, matrices y variables

        vec2dim final;

        final.resize(31);
```

```
int mainCont2 = 0;//Contador para recorrer los escenarios

do{
int mainCont = 0;//Contador para realizar varias corridas dentro de un mismo escenario



do{

        if (argc != 3)
        { ////cout << "Illegal Execution!\n";
                exit(1);
        }

        vector<Tjob> jobArray;
        char jobtype[50];

        //Se leen los archivos provenientes del software Lekin

        ReadJobFile(jobArray,jobtype);//reads the file _user.job

        vector<WkCenter> wkcArray;
        char wktype[50];

        ReadWkcFile(wkcArray, wktype);//reads the file _user.mch

        int tejeci,tejecf, tejec;
        //Variables correspondientes a los tiempos
                tejeci = time(0);
                tejecf = time(0);
                tejec = tejecf - tejeci;

        Individual ind;
        PetriNet pn;  //Petri net Object
```

```
Schedule pnSchedule;


double exp;

double MinInitialObjective = 1000000000000000;

//Valor inicial para el objetivo a evaluar

double initialObjective;

//Valor del objetivo inicial

double PrMakespan;

//Porcentaje de ponderación del makespan

double PrtotalWeightedTardiness;

//Porcentaje de ponderación de la tardanza total ponderada

double SPTobjective, LPTobjective, MRWTobjective, EDDobjective;

double CRobjective , ATCobjective, MSLACKobjective, AGobjective;

//Valores objetivos obtenidos por los programas activos con la regla de de
//despacho para seleccinar transiciones

double alfa, beta;


int nbreaks;

//Número de fallas de máquina a simular

int tnowSPT, tnowLPT, tnowMRWT, tnowEDD, tnowCR, tnowATC, tnowMSLACK, tnowAG;

//Tiempo de finalización de cada una de las secuencias de programas activos

int makespan;

//Tiempo de finalización de la programación inicial

int Pavg;

//Tiempo operacional promedio

int Mintnow = 100000000000;

//Mínimo tiempo de terminación

int ta = 0;

//Tiempo actual

int tnow = 0;

//tiempo inicial

int njobs = jobArray.size();

//Número de trabajos en el sistema

int nmach = wkcArray.size();
```

```
//Número de máquinas en el sistema

int nStations = nmach;

//Número de estaciones en el sistema

int ActTTP;

int TTPIS;

int generationIS;


intvec Mf, Mp, Mr, p_id;

//Final Marking

intvec um, ump;

// vector of active transitions

intvec pe_id, pf_id;

// vector plazas de entrada y salida de los job

intvec tr;

//Vector de las tiempos de disparos de las transiciones de la programación
final

intvec PTj;

//Vector de los tiempos de procesamiento de los trabajos

intvec timeU;

//Vector de las tiempos de disparos de las transiciones de la programación
inicial

intvec machinesU;

//Vector de las máquinas en que se desarrollan las operaciones en la
//Programación inicial

intvec jobsU;

//Vector de los trabajos en la programación inicial

intvec operationsU;

//Vector de las operaciones en la programación inicial

intvec ptU;

//Vector de los tiempos de procesamiento de las operaciones en la programación
//inicial

intvec jobsR;

//Vector de los trabajos en la programación final

intvec operationsR;

//Vector de las operaciones en la programación final
```

```
intvec machinesR;

//Vector de las máquinas en que se desarrollan las operaciones en la
//programación final

intvec ptR;

//Vector de los tiempos de procesamiento de las operaciones en la programación
//final

//Vectores que guardan las secuencias obtenidas en la programación inicial

intvec sequenceActSched;

intvec sequenceSPT;

intvec sequenceLPT;

intvec sequenceMRWT;

intvec sequenceEDD;

intvec sequenceCR;

intvec sequenceATC;

intvec sequenceMSLACK;

intvec sequenceAG;

//Vectores que guardan los cromosomas codificados de las secuencias obtenidas
//en la programación inicial

intvec chrmCoded;

intvec chrmCodedSPT;

intvec chrmCodedLPT;

intvec chrmCodedMRWT;

intvec chrmCodedATC;

intvec chrmCodedEDD;

intvec chrmCodedCR;

intvec chrmCodedMSLACK;


deque<int> u;

//Secuencia de transiciones a disparar en la programación inicial

deque<int> u_activo;

//Secuencia de transiciones a disparar en la programación obtenida por el
//algoritmo genético

deque<int> t;

deque<int> ur;

deque<int> td;
```

```
vector<intvec> marking;
vec2dim transitions;
//Almacena las transiciones que corresponden a la operación de cada trabajo
vec2dim operEndTime;
//Almacena los tiempos de finalización de cada operación
vec2dim machBreakDown;
//Matriz que guarda la información de la falla de máquina

//Se realiza la configuración de la red de petri (Modelaje del sistema)
convertToPN(jobArray,wkcArray,pn,Mf, transitions , Pavg, PTj);
//Se convierte el sistema en una red de petri

t.clear();

cons_pe_id(pn, p_id, pe_id);
cons_pf_id(pn, p_id, pf_id);

srand(time(0));
//Se construye el vector de transitions habillitadas
constructU(pn,um);
constructUmp(pn,um,ump);

pn.reset();
//Se especifica el objetivo de la programación
int objective = 9;
//7 = Tardanza total ponderada
//8 = Makespan
//9 = Objetivo ponderado
int rule;
//1 = SPT
//2 = LPT
```

```
//3 = MRWT

//4 = EDD

//5 = CR

//6 = ATC

//7 = MSLACK

//8 = AG


int ProblemSize;

//0 = Pequeño

//1 = Mediano

//2 = Grande


ProblemSize = 2;




sequenceActSched.clear();

sequenceSPT.clear();

sequenceLPT.clear();

sequenceMRWT.clear();

sequenceEDD.clear();

sequenceCR.clear();

sequenceATC.clear();

sequenceMSLACK.clear();

sequenceAG.clear();


if(objective == 9){

        PrMakespan = alfa;

        PrtotalWeightedTardiness = beta;

}


////Se obtienen las secuencias y sus tiempos por medio del algoritmo para
programas activos
```

```
tnow = 0;

int temp1;


rule = 1;

tnowSPT = ActSched(pn, u, jobArray, transitions, marking, timeU, jobsU,
operationsU, machinesU, ptU, operEndTime, PTj, initialObjective, pf_id,
chrmCodedSPT, objective, PrMakespan, PrtotalWeightedTardiness, rule, ActTTP);

for(int j=0;j<u.size();j++){

        sequenceSPT.push_back(u[j]);

}

SPTobjective = initialObjective;

if(SPTobjective <      MinInitialObjective){

        MinInitialObjective = SPTobjective;

        temp1 = 1;

}


rule = 2;

tnowLPT = ActSched(pn, u, jobArray, transitions, marking, timeU, jobsU,
operationsU, machinesU, ptU, operEndTime, PTj, initialObjective, pf_id,
chrmCodedLPT, objective, PrMakespan, PrtotalWeightedTardiness, rule, ActTTP);

for(int j=0;j<u.size();j++){

        sequenceLPT.push_back(u[j]);

}

LPTobjective = initialObjective;

if(LPTobjective <      MinInitialObjective){

        MinInitialObjective = LPTobjective;

        temp1 = 2;

}


rule = 3;

tnowMRWT = ActSched(pn, u, jobArray, transitions, marking, timeU, jobsU,
operationsU, machinesU, ptU, operEndTime, PTj, initialObjective, pf_id,
chrmCodedMRWT, objective, PrMakespan, PrtotalWeightedTardiness, rule, ActTTP);

for(int j=0;j<u.size();j++){

        sequenceMRWT.push_back(u[j]);

}
```

```
MRWTobjective = initialObjective;

if(MRWTobjective <     MinInitialObjective){

        MinInitialObjective = MRWTobjective;

        temp1 = 3;

}


rule = 4;

tnowEDD = ActSched(pn, u, jobArray, transitions, marking, timeU, jobsU,
operationsU, machinesU, ptU, operEndTime, PTj, initialObjective, pf_id,
chrmCodedEDD, objective, PrMakespan, PrtotalWeightedTardiness, rule, ActTTP);

for(int j=0;j<u.size();j++){

        sequenceEDD.push_back(u[j]);

}

EDDobjective = initialObjective;

if(EDDobjective <      MinInitialObjective){

        MinInitialObjective = EDDobjective;

        temp1 = 4;

}


rule = 5;

tnowCR = ActSched(pn, u, jobArray, transitions, marking, timeU, jobsU,
operationsU, machinesU, ptU, operEndTime, PTj, initialObjective, pf_id,
chrmCodedCR, objective, PrMakespan, PrtotalWeightedTardiness, rule, ActTTP);

for(int j=0;j<u.size();j++){

        sequenceCR.push_back(u[j]);

}

CRobjective = initialObjective;

if(CRobjective < MinInitialObjective){

        MinInitialObjective = CRobjective;

        temp1 = 5;

}


rule = 6;

tnowATC = ActSched(pn, u, jobArray, transitions, marking, timeU, jobsU,
operationsU, machinesU, ptU, operEndTime, PTj, initialObjective, pf_id,
chrmCodedATC, objective, PrMakespan, PrtotalWeightedTardiness, rule, ActTTP);
```

```
for(int j=0;j<u.size();j++){
        sequenceATC.push_back(u[j]);
}
ATCobjective = initialObjective;
if(ATCobjective <      MinInitialObjective){
        MinInitialObjective = ATCobjective;
        temp1 = 6;
}

rule = 7;
tnowSLACK = ActSched(pn, u, jobArray, transitions, marking, timeU, jobsU,
operationsU, machinesU, ptU, operEndTime, PTj, initialObjective, pf_id,
chrmCodedMSLACK, objective, PrMakespan, PrtotalWeightedTardiness, rule,
ActTTP);
for(int j=0;j<u.size();j++){
        sequenceMSLACK.push_back(u[j]);
}
MSLACKobjective = initialObjective;
if(MSLACKobjective <   MinInitialObjective){
        MinInitialObjective = MSLACKobjective;
        temp1 = 7;
}

//Se escoge la secuencia que tenga el mejor valor objetivo

int chrmSize = u.size();
//Tamaño del cromosoma
int mchBreakdownInstant;
//Tiempo en el que sucede la falla de máquina
int GammaIS = 0 ;
//Porcentaje del cromosoma que se perturba en la búsqueda local
intvec allFitnessGA;
//Todos los fitness de la población
intvec uTemporal;
```

```
//Vector en donde se almacenan la secuencia de transiciones encontrada por el
//AG de programación inicial

intvec Cj (u.size());

Individual winner;

//Mejor solución encontrada en la generación

Individual bestSolution;

//Mejor solución encontrada en la búsqueda local

Individual neighbour;

//Vecino

intvec winnerDec;

//Cromosoma decodificado de la mejor solución encontrada en la solución

vec2dim sched;


int maxGenIS;

//Número de generaciones del AG en la programación inicial

int psizeIS;

//Tamaño de la población del AG en la programación inicial

int elitismPercentageIS;

int mutIntIS;

int NoJob_t = 0;

int bestPopulationPercentageIS = 100;

int TTPAGIS;

//Tardanza total ponderada del algoritmo genético de la programación inicial

double prCrossIS;

//Probabilidad de cruce en la programación inicial

double prMutIS;

//Probabilidad de mutación en la programación inicial

double initialObjectiveIS;

//Valor objetivo de la programación inicial


bool applyMemeticIS = true;

bool elitismIS = true;

bool crossoverTypeIS = true;

// false = two points, true = one Point
```

```
//****************************************
bool applyLocalSearchIS = false;
bool localSearchIS = false;
//true = HillClimbing, false = Simulated Annealing
bool localSearchTypeIS = false;
// false = iRZ, true = Swap
bool iRZtypeIS = true;
//true = termina al encontrar mejor; false = todas las posibilidades
//****************************************

if(objective == 7){//Parámetros para la tardanza total ponderada
        maxGenIS=200;
        psizeIS=100;
        prCrossIS=0.9;
        prMutIS=0.5;
        mutIntIS = 4;
        elitismIS = true;
        elitismPercentageIS = 50;
}else if(objective == 8){//Parámetros para el makespan
        maxGenIS=200;
        psizeIS=100;
        prCrossIS=0.9;
        prMutIS=0.5;
        mutIntIS = 4;
        elitismIS = false;
        elitismPercentageIS = 50;
}else if(objective == 9){//Parámetros para el makespan
        if(alfa == 1 && beta == 0){
                if(ProblemSize == 0 || ProblemSize == 1){
                //PROBLEMAS PEQUEÑOS Y MEDIANOS
                        maxGenIS=100;
                        psizeIS=100;
```

```
                prCrossIS=0.6;
                prMutIS=0.5;
                mutIntIS = 2;
                elitismIS = true;
                elitismPercentageIS = 50;
        }else if(ProblemSize == 2){
        //PROBLEMAS GRANDES
                maxGenIS=100;
                psizeIS=100;
                prCrossIS=0.6;
                prMutIS=0.5;
                mutIntIS = 2;
                elitismIS = false;
                elitismPercentageIS = 0;
        }
}

if(alfa == 0 && beta == 1){
        if(ProblemSize == 0 || ProblemSize == 1 || ProblemSize == 2){
        //TODOS LOS PROBLEMAS
                maxGenIS=200;
                psizeIS=100;
                prCrossIS=0.9;
                prMutIS=0.5;
                mutIntIS = 4;
                elitismIS = true;
                elitismPercentageIS = 50;
        }
}

if(alfa == 0.5 && beta == 0.5){
        if(ProblemSize == 0 || ProblemSize == 1){
        //PROBLEMAS PEQUEÑOS Y MEDIANOS
```

```
                        maxGenIS=200;
                        psizeIS=100;
                        prCrossIS=0.6;
                        prMutIS=0.5;
                        mutIntIS = 4;
                        elitismIS = true;
                        elitismPercentageIS = 50;
                }else if(ProblemSize == 2){
                //PROBLEMAS GRANDES
                        maxGenIS=200;
                        psizeIS=200;
                        prCrossIS=0.9;
                        prMutIS=0.5;
                        mutIntIS = 4;
                        elitismIS = true;
                        elitismPercentageIS = 50;
                }
        }

}



uTemporal.clear();


if(applyMemeticIS == true){
        rule = 8;
        tnow = 0;


        Memetic mem(prMutIS, prCrossIS, mutIntIS, maxGenIS , elitismIS ,
        crossoverTypeIS , GammaIS , applyLocalSearchIS , localSearchIS,
        localSearchTypeIS , iRZtypeIS);


        //Se genera la población inicial
```

```
mem.initPopulationIS(pn, tnow, pf_id, allFitnessGA, chrmCoded,
chrmCodedSPT, chrmCodedLPT, chrmCodedMRWT, chrmCodedEDD, chrmCodedCR,
chrmCodedATC, ta, psizeIS, chrmSize, objective, jobArray,
mchBreakdownInstant, um, PrMakespan, PrtotalWeightedTardiness, ur, td,
PTj);

//Se ejecuta el algoritmo genético

mem.runIS(pn, tnow, pf_id, allFitnessGA, ta, elitismPercentageIS,
bestPopulationPercentageIS, winner, neighbour, objective, jobArray,
mchBreakdownInstant, PrMakespan, PrtotalWeightedTardiness,
initialObjective, ur, td, generationIS);

//Se genera el programa inicial utilizando el cromosoma del individuo
//con mejor fitness


winner.generateScheduleIS(pn, jobArray, tnow, pf_id, NoJob_t, sched, u,
tr, winnerDec, transitions, Cj, PrMakespan, PrtotalWeightedTardiness,
TTPAGIS);


for(int i=0; i<u.size();i++){

        uTemporal.push_back(u[i]);

}

//Teniendo el mejor individuo generado por el AG se realiza una
//simulación en la Red de Petri con esa secuencia y se actualizan los
//vectores

tnowAG = InitialSched(pn, u, jobArray, transitions, marking, timeU,
jobsU, operationsU, machinesU, ptU, operEndTime, initialObjectiveIS,
pf_id, uTemporal, objective, PrMakespan, PrtotalWeightedTardiness,
TTPAGIS);

for(int j=0; j<u.size(); j++){

        sequenceAG.push_back(u[j]);

}

AGobjective = initialObjectiveIS;

if(AGobjective <       MinInitialObjective){

        MinInitialObjective = AGobjective;

        temp1 = 8;

}



//Se selecciona el mejor programa inicial

if(temp1 == 1){

        rule = 1;
```

```
            u.clear();

            tnow = ActSched(pn, u, jobArray, transitions, marking, timeU,
            jobsU, operationsU, machinesU, ptU, operEndTime, PTj,
            initialObjective, pf_id, chrmCoded, objective, PrMakespan,
            PrtotalWeightedTardiness, rule, ActTTP);

            TTPIS = ActTTP;

        }else if(temp1 == 2){

            rule = 2;

            u.clear();

            tnow = ActSched(pn, u, jobArray, transitions, marking, timeU,
            jobsU, operationsU, machinesU, ptU, operEndTime, PTj,
            initialObjective, pf_id, chrmCoded, objective, PrMakespan,
            PrtotalWeightedTardiness, rule, ActTTP);

            TTPIS = ActTTP;

        }else if(temp1 == 3){

            rule = 3;

            u.clear();

            tnow = ActSched(pn, u, jobArray, transitions, marking, timeU,
            jobsU, operationsU, machinesU, ptU, operEndTime, PTj,
            initialObjective, pf_id, chrmCoded, objective, PrMakespan,
            PrtotalWeightedTardiness, rule, ActTTP);

            TTPIS = ActTTP;

        }else if(temp1 == 4){

            rule = 4;

            u.clear();

            tnow = ActSched(pn, u, jobArray, transitions, marking, timeU,
            jobsU, operationsU, machinesU, ptU, operEndTime, PTj,
            initialObjective, pf_id, chrmCoded, objective, PrMakespan,
            PrtotalWeightedTardiness, rule, ActTTP);

            TTPIS = ActTTP;

        }else if(temp1 == 5){

            rule = 5;

            u.clear();

            tnow = ActSched(pn, u, jobArray, transitions, marking, timeU,
            jobsU, operationsU, machinesU, ptU, operEndTime, PTj,
            initialObjective, pf_id, chrmCoded, objective, PrMakespan,
            PrtotalWeightedTardiness, rule, ActTTP);

            TTPIS = ActTTP;

        }else if(temp1 == 6){

            rule = 6;
```

```
                u.clear();

                tnow = ActSched(pn, u, jobArray, transitions, marking, timeU,
                jobsU, operationsU, machinesU, ptU, operEndTime, PTj,
                initialObjective, pf_id, chrmCoded, objective, PrMakespan,
                PrtotalWeightedTardiness, rule, ActTTP);

                TTPIS = ActTTP;

        }else if(temp1 == 7){

                rule = 7;

                u.clear();

                tnow = ActSched(pn, u, jobArray, transitions, marking, timeU,
                jobsU, operationsU, machinesU, ptU, operEndTime, PTj,
                initialObjective, pf_id, chrmCoded, objective, PrMakespan,
                PrtotalWeightedTardiness, rule, ActTTP);

                TTPIS = ActTTP;

        }else if(temp1 == 8){

                rule = 8;

                u.clear();

                tnow = InitialSched(pn, u, jobArray, transitions, marking,
                timeU, jobsU, operationsU, machinesU, ptU, operEndTime,
                initialObjectiveIS, pf_id, uTemporal, objective, PrMakespan,
                PrtotalWeightedTardiness, TTPAGIS);

                TTPIS = TTPAGIS;

        }


}


makespan = tnow;

tejecf = time(0);

tejec = tejecf - tejeci;


for(int j=0;j<u.size();j++){

        sequenceActSched.push_back(u[j]);

}


//Se genera la programación inicial teniendo en cuenta la secuencia obtenida

pnSchedule.genSchedule(pn,ta,u,t);
```

```
                //Impresión del resultado en Lekin
                pnSchedule.printToLekin(wkcArray);

                //Declaración de las variables, vectores y matrices necesarias para la
simulación y reprogramación

                int p = 0;
                //Tiempo de reparación
                int q;
                //Máquina que falla
                int z = 0;
                //Posición de la transición escogida para ser disparada en el vector um
                int h = 0;
                //Contador correspondiente a las fallas de máquinas
                int d = 0;
                //Contador que recorre el vector u de trnsiciones en la programación inicial
                int w;
                //Adicionar token a la plaza w después de la falla
                int r;
                //(Operación dañada) Remover token
                int delta;
                NoJob_t = 0;
                int Dj ;
                //Fecha de entrega del trabajo j
                int Wj ;
                //Peso del trabajo j
                int numJob;
                int index;
                int pt;
                int oper;
                int machine;

                int bestPopulationPercentage = 100;
                intvec VecWjTjTran;
```

```
//Vector que guarda las transiciones que habilitan la última operación de cada
trabajo a medida que van iniciando dichas operaciones

intvec VecWjTjJobs;

intvec VecWjTj;

//Vector que guarda la sumatoria de las tardanzas ponderadas totales en el
memento que se van disparando las trnasicines que dan inicio a las últimas
operaciones de ccada trabajo


intvec jobTimes;

jobTimes.resize(jobArray.size());


double sumWjTj = 0;

//Sumatoria de la tardanzas ponderada de los trabajos

double sumCj = 0;

//Sumatoria de los tiempos de finalización de los trabajos

double sumTj = 0;

//Sumatoria de las tardanzas de los trabajos

double sumWjCj = 0;

//Sumatoria de los tiempos de finalización ponderadas de los trabajos



Cj.clear();

Cj.resize(u.size());

double objectiveValue;

//Valor del objetivo final

double TTPRS;

double desviation;


VecWjTjJobs.clear();

VecWjTjTran.clear();

VecWjTj.clear();

mchBreakdownInstant = 0;

//Instante de la falla de máquina
```

```
		int maxGen;
		//Número de generaciones para el AM de la reprogramación
		int psize;
		//Tamaño de la población para el AM de la reprogramación
		double prCross;
		//Porcentaje de cruce para el AM de la reprogramación
		double prMut;
		//Porcentaje de mutación para el AM de la reprogramación
		int mutInt;
		bool elitism;
		int elitismPercentage;
		double decrease;
		//Simulated annealing


	//Configurar**************************************************************************
**

		if(alfa == 1 && beta == 0){//MAKESPAN
			if(ProblemSize == 0){
					maxGen = 100;
					//Número de generaciones para el AM de la reprogramación
					psize = 100;
					//Tamaño de la población para el AM de la reprogramación
					prCross = 0.9;
					//Porcentaje de cruce para el AM de la reprogramación
					prMut = 0.1;
					//Porcentaje de mutación para el AM de la reprogramación
					mutInt = 1;
					elitism = true;
					elitismPercentage = 50;
					decrease = 10;
					//Simulated annealing
			}else if(ProblemSize == 1){
```

```
                        maxGen = 100;
                        //Número de generaciones para el AM de la reprogramación
                        psize = 100;
                        //Tamaño de la población para el AM de la reprogramación
                        prCross = 0.9;
                        //Porcentaje de cruce para el AM de la reprogramación
                        prMut = 0.5;
                        //Porcentaje de mutación para el AM de la reprogramación
                        mutInt = 2;
                        elitism = false;
                        elitismPercentage = 0;
                        decrease = 1;//Simulated annealing
                }else if(ProblemSize == 2){
                        maxGen = 100;
                        //Número de generaciones para el AM de la reprogramación
                        psize = 50;
                        //Tamaño de la población para el AM de la reprogramación
                        prCross = 0.6;
                        //Porcentaje de cruce para el AM de la reprogramación
                        prMut = 0.1;
                        //Porcentaje de mutación para el AM de la reprogramación
                        mutInt = 1;
                        elitism = false;
                        elitismPercentage = 0;
                        decrease = 1;
                        //Simulated annealing
                }
        }

        if(alfa == 0 && beta == 1){
                if(ProblemSize == 0){
                        maxGen = 200;
                        //Número de generaciones para el AM de la reprogramación
```

```
                psize = 100;
                //Tamaño de la población para el AM de la reprogramación
                prCross = 0.6;
                //Porcentaje de cruce para el AM de la reprogramación
                prMut = 0.5;
                //Porcentaje de mutación para el AM de la reprogramación
                mutInt = 2;
                elitism = false;
                elitismPercentage = 0;
                decrease = 10;
                //Simulated annealing
        }else if(ProblemSize == 1){
                maxGen = 200;
                //Número de generaciones para el AM de la reprogramación
                psize = 100;
                //Tamaño de la población para el AM de la reprogramación
                prCross = 0.6;
                //Porcentaje de cruce para el AM de la reprogramación
                prMut = 0.5;
                //Porcentaje de mutación para el AM de la reprogramación
                mutInt = 2;
                elitism = false;
                elitismPercentage = 0;
                decrease = 10;
                //Simulated annealing
        }else if(ProblemSize == 2){
                maxGen = 200;
                //Número de generaciones para el AM de la reprogramación
                psize = 50;
                //Tamaño de la población para el AM de la reprogramación
                prCross = 0.9;
                //Porcentaje de cruce para el AM de la reprogramación
                prMut = 0.5;
```

```
                        //Porcentaje de mutación para el AM de la reprogramación
                        mutInt = 2;
                        elitism = true;
                        elitismPercentage = 50;
                        decrease = 1;
                        //Simulated annealing
        }
}

if(alfa = 0.5 && beta == 0.5){
        if(ProblemSize == 0){
        //Problemas pequeños
                        maxGen = 200;
                        //Número de generaciones para el AM de la reprogramación
                        psize = 50;
                        //Tamaño de la población para el AM de la reprogramación
                        prCross = 0.9;
                        //Porcentaje de cruce para el AM de la reprogramación
                        prMut = 0.1;
                        //Porcentaje de mutación para el AM de la reprogramación
                        mutInt = 4;
                        elitism = false;
                        elitismPercentage = 0;
                        decrease = 10;
                        //Simulated annealing
        }else if(ProblemSize == 1){
        //Problemas medianos
                        maxGen = 100;
                        //Número de generaciones para el AM de la reprogramación
                        psize = 100;
                        //Tamaño de la población para el AM de la reprogramación
                        prCross = 0.9;
                        //Porcentaje de cruce para el AM de la reprogramación
```

```
                prMut = 0.5;
                //Porcentaje de mutación para el AM de la reprogramación
                mutInt = 4;
                elitism = false;
                elitismPercentage = 0;
                decrease = 1;
                //Simulated annealing
            }else if(ProblemSize == 2){
            //Problemas grandes
                maxGen = 200;
                //Número de generaciones para el AM de la reprogramación
                psize = 50;
                //Tamaño de la población para el AM de la reprogramación
                prCross = 0.9;
                //Porcentaje de cruce para el AM de la reprogramación
                prMut = 0.5;
                //Porcentaje de mutación para el AM de la reprogramación
                mutInt = 1;
                elitism = true;
                elitismPercentage = 50;
                decrease = 1;
                //Simulated annealing
            }
        }


//*****************************************************************************

        int Gamma = 30;
        //Porcentaje del cromosoma que se perturba en la búsqueda local
        bool applyMemetic = true;
        bool crossoverType = true;// false = two points, true = one Point
        bool applyLocalSearch = true;
```

```
bool localSearch = false; //true = HillClimbing, false = Simulated Annealing


//*****************

bool localSearchType1 = true;// false = iRZ, true = Swap

bool iRZtype = true; //true = termina al encontrar mejor; false = todas las
//posibilidades

//*****************


bool ReactiveScheduling = true;


sched.clear();


int tejec1;

tejec1 = tejecf - tejeci;


if(ReactiveScheduling == true){

        tejeci = time(0);

        tejecf = time(0);

        tejec = tejecf - tejeci;


        generateMachineBreakdown (pn, nStations, tnow, marking, wkcArray,
timeU, u, operEndTime, nbreaks, machBreakDown , Pavg, mainCont2);


        um.resize(u.size());

        tnow = 0;


        while (d < u.size()) {


                if (h < nbreaks) {


                        if (t[z]>machBreakDown[h][0]-ta) {


                                w = machBreakDown[h][7];

                                //Adicionar token a la plaza w
```

```
r = machBreakDown[h][8];

//(Operación dañada) Remover token

q = machBreakDown[h][9];

//Máquina en reparación

p = machBreakDown[h][2];

//Tiempo de reparación

mchBreakdownInstant = machBreakDown[h][0];


pn.changePT( p, q, pn);

//Cambia los tiempos de procesamientos de las
//plazas que pertenencen a la máquina en
reparación


if(objective == 8 ){//Tardanza total ponderada

        for(int i=0; i<VecWjTjTran.size(); i++){


        if(machBreakDown[h][4]==VecWjTjTran[i]){

                sumWjTj = sumWjTj - VecWjTj[i];

                }

        }

}


//Se halla el mayor tamaño del cromosoma para la
reprogramación (Puede variar dependiendo de la
falla)


if(h==0){

        chrmSize = u.size();

}


chrmSize = u.size() - z + 2 ;



//Se cambia el estado de los vecotres Mp y Mr por
//el disparo de la transición con la falla de
//máquina
```

```
delta = machBreakDown[h][0] - td[td.size()-1];

um[z] = machBreakDown[h][6];

ur.push_back(um[z]);

pn.changeMachineBreakDown(um,z,delta);


tnow = tnow + delta;

td.push_back(tnow);


numJob = jobNo(pn,um[z]);


jobOperationIndex(um, transitions, oper, z,
numJob);


pt = pn.getPT()[r];


jobTimes[numJob]= jobTimes[numJob] - pt;


machine = q;

machinesR.push_back(machine);


numJob = 0;//********

oper = 0;//*********

pt = p;//******


jobsR.push_back(numJob);

operationsR.push_back(oper);


ptR.push_back(pt);



d = d + 1;

z=z+1;

h = h + 1;
```

```
ta = machBreakDown[h-1][0];


//******REPROGRAMACIÓN*******


applyMemetic = true;

//Se dan los parámetros para el algorito memético


if(applyMemetic == true){

        Memetic mem(prMut , prCross , mutInt,
        maxGen , elitism , crossoverType , Gamma
        , applyLocalSearch , localSearch,
        localSearchType1 , iRZtype);

        //Se genera la población inicial del AM

        mem.initPopulation(pn, tnow, pf_id,
        allFitnessGA, ta, psize, chrmSize,
        objective, jobArray, mchBreakdownInstant,
        um, p, PrMakespan,
        PrtotalWeightedTardiness, ur, td,
        sumWjTj, sumCj, sumTj, sumWjCj, NoJob_t,
        Cj, PTj, jobTimes);

        //Evolución del AM

        mem.run(pn, tnow, pf_id, allFitnessGA,
        ta, elitismPercentage,
        bestPopulationPercentage, winner,
        chrmSize, neighbour, objective, jobArray,
        mchBreakdownInstant, p, PrMakespan,
        PrtotalWeightedTardiness,
        initialObjective, ur, td, sumWjTj, sumCj,
        sumTj, sumWjCj, NoJob_t, Cj, decrease);


        //Se genera la programación final y se
        //actualizan los vectores


        winner.generateSchedule(pn, jobArray,
        tnow, pf_id, NoJob_t, sched, u, tr,
        jobsR, operationsR, machinesR, ptR,
        winnerDec, transitions, Cj, sumWjTj,
        PrMakespan, PrtotalWeightedTardiness,
        TTPRS);



}


for(int j=0; j<u.size(); j++){
```

```
                ur.push_back(u[j]);
                td.push_back(tr[j]);
            }

            t.clear();
            pnSchedule.genSchedule(pn,ta,u,t);

            break;
        }
        else {

            um[z] = u[z];
            ur.push_back(um[z]);
            pn.changeState(um,z,delta);
            d = d + 1;
            tnow = tnow + delta;
            td.push_back(tnow);
            numJob = jobNo(pn,um[z]);
            jobOperationIndex(um, transitions, oper, z,
            numJob);
            index = pn.getCplus()[um[z]][0];
            pt = pn.getPT()[index];

            if(pt>0){
                machine = allMchNo(pn,um[z]);
                machinesR.push_back(machine);
            }else{
                machinesR.push_back(0);
                machine = 0;
            }

            jobsR.push_back(numJob+1);
            operationsR.push_back(oper+1);
            ptR.push_back(pt);
```

```
if(pn.getp_id()[pn.getCmin()[u[z]][0]]>=0){

        int jobNo = jobArray
        [pn.getp_id()[pn.getCmin()[u[z]][0]]].get
        Number();


        jobTimes[jobNo] = jobTimes[jobNo] + pt;


        if (pn.getCplus()[ur[d-1]][0] ==
        pf_id[jobNo]) {


                Cj[NoJob_t] = tnow;


                Wj = jobArray
                [pn.getp_id()[pn.getCmin()[u[z]][
                0]]].getWeight();

                Dj = jobArray
                [pn.getp_id()[pn.getCmin()[u[z]][
                0]]].getDue();


                if (objective == 4) {

                        sumCj = sumCj +
                        Cj[NoJob_t];

                }

                else if (objective == 5) {

                        sumWjCj = sumWjCj +
                        Wj*(Cj[NoJob_t]);

                        }

                else if (objective == 6) {

                        if ( (Cj[NoJob_t] - Dj) >=
                        0)

                                sumTj = sumTj +
                                (Cj[NoJob_t] -
                                Dj);

                        }

                else if (objective == 7 ||
                objective == 8 || objective == 9)
                {
```

```
                                                        if ( (Cj[NoJob_t] - Dj) >=
                                                        0) {
                                                                sumWjTj = sumWjTj
                                                                + Wj *
                                                                (Cj[NoJob_t] -
                                                                Dj);

VecWjTjJobs.push_back(numJob);

VecWjTjTran.push_back(u[z]);

VecWjTj.push_back(Wj * (Cj[NoJob_t] - Dj));
                                                        }


                                                                }
                                                                NoJob_t = NoJob_t + 1;
                                                                }
                                                        else;
                                                }

                                                z=z+1;

                                                 if (objective == 4)  {//sum Cj
                                                                objectiveValue = sumCj;
                                                        }
                                                        else if (objective == 5)  {
                                                                objectiveValue = sumWjCj;
                                                        }
                                                        else if (objective == 6)  {
                                                                objectiveValue = sumTj;
                                                        }
                                                        else if (objective == 7)  {
                                                                objectiveValue = sumWjTj;

                                                        }
                                                        else if (objective == 8)  {
```

```
                                                        objectiveValue = tnow;

                                                    }
                                                    else if (objective == 9)  {
                                                        objectiveValue =
(PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);
                                                    }



                                    }
                                }
                            }


                    }// if(ReactiveSchduling = true)



                    //************ RESUMEN FINAL *************

                    final[0].push_back(MinInitialObjective*10);
                    final[1].push_back(makespan);
                    final[2].push_back(TTPIS);
                    final[3].push_back(maxGenIS);
                    final[4].push_back(psizeIS);
                    final[5].push_back(prCrossIS*100);
                    final[6].push_back(prMutIS*100);
                    final[7].push_back(mutIntIS);
                    if(elitismIS == true){
                    final[8].push_back(elitismPercentageIS);
                    }else{
                    final[8].push_back(0);
                    }
                    final[9].push_back(generationIS);
                    final[10].push_back(tejec1);
```

```
if(ReactiveScheduling == true){

//****** FALLA DE LA MÁQUINA *****

        for(int i=0; i<nbreaks; i++){
                final[11].push_back(machBreakDown[i][0]);
                final[12].push_back(machBreakDown[i][1]);
                final[13].push_back(machBreakDown[i][2]);
                final[14].push_back(machBreakDown[i][5]);
                final[15].push_back(machBreakDown[i][3]);
                final[16].push_back(machBreakDown[i][4]);
                final[17].push_back(machBreakDown[i][6]);
        }

        //****** PROGRAMACIÓN REACTIVA *****

        final[18].push_back(winner.fitness*10);
        final[19].push_back(tnow);
        final[20].push_back(TTPRS);

        if(MinInitialObjective == 0){
                desviation = winner.fitness;
        }else{
                desviation = (winner.fitness - MinInitialObjective) /
                MinInitialObjective;
        }

        final[21].push_back(desviation*100);



                final[22].push_back(maxGen);
                final[23].push_back(psize);
                final[24].push_back(prCross*100);
```

```
                        final[25].push_back(prMut*100);
                        final[26].push_back(mutInt);
                        if(elitismIS == true){
                                final[27].push_back(elitismPercentage);
                        }else{
                                final[27].push_back(0);
                        }
                        final[28].push_back(decrease);

                        tejecf = time(0);
                        tejec = tejecf - tejeci;
                        final[29].push_back(tejec+tejec1);
                        final[30].push_back(tejec);
                }

                mainCont = mainCont + 1;

        }while(mainCont < 1);

        mainCont2++;

        }while(mainCont2 < 4);

        return 0;
}
```

---

funvectors.h

```
#include <vector>
```

```
#include <fstream>
#include <iostream>
#include <iterator>
#include <algorithm>
#include <cassert>
using namespace std;

//vector of integers
typedef vector<int> intvec;

//a vector of vectors of integers. Each vector can be of a different size
typedef vector<intvec> vec2dim;

//overloads the operator << to display intvecs and vec2dims
ostream& operator << (ostream& os, const intvec& iv);
ostream& operator << (ostream& os, const vec2dim& iv2);

//Reads a vector from an istream
//The istream can be cin or any valid input file
void read_vector(istream&,intvec&);
```

---

funvectors.cpp

```
#include "funvectors.h"

ostream& operator << (ostream& os, const intvec& iv)
{
        copy (iv.begin(),iv.end(),ostream_iterator<int> (os, " "));
        return os;
```

```
}

ostream& operator << (ostream& os, const vec2dim& iv2)
{
        vec2dim::const_iterator vecPtr=iv2.begin();

        int i=0;
        while (vecPtr != iv2.end()) {

                copy (vecPtr[i].begin(),vecPtr[i].end(),ostream_iterator<int> (os, " "));
                os << endl;
                vecPtr++;

        }

                return os;
}

//*********************************************************
vector<int> operator+ (const vector<int>& v1, const vector<int>& v2) {

        vector<int> v3;
        assert(v1.size()==v2.size());
        v3.resize(v1.size());
        for (int i=0;i<v1.size();i++)
                v3[i]= v1[i]+v2[i];
        return v3;
}

void read_vector(istream& is, intvec& iv){

        int i,n=iv.size();
        iv.clear();
```

```
        istream_iterator<int> inputint(is);
        iv.push_back(*inputint);
        i= 1;

        while (i<n) {
                ++ inputint;
                iv.push_back(*inputint);
                ++i;
        }


}
```

---

individual.h: interface for the individual class.

```
#include <iostream>
#include "funvectors.h"
#include "Tjob.h"
#include "WkCenter.h"
#include "PetriNet.h"
#include "Schedule.h"

#include <stdio.h>
#include <vector>
using namespace std;

#if !defined(AFX_Individual_H__77B31817_21E0_42AB_BCB4_827B4C401662__INCLUDED_)
#define AFX_Individual_H__77B31817_21E0_42AB_BCB4_827B4C401662__INCLUDED_
```

```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000



/**
* \file    Individual.h
* \brief   Clase Individual
* \version 1.0
* \date    25/02/2005 cambiar Fecha!!!
* \author  Jennifer Grice
* \par     Proyecto Investigaci&oacute;n
*
* \par     Descripci&oacute;n
*                  Esta clase representa un individuo (cromosoma) en la poblacion del Algoritmo
Mem&eacute;tico.
*                  En esta clase se inicializa un individuo y se calcula su fitness y su
objetivo ponderado.
*
*/


class Individual
{
friend bool operator < (const Individual& ind1, const Individual& ind2);
friend ostream& operator << (ostream& , const Individual& );


public:
               /*! \fn Individual()
                *  \brief Constructor por defecto de la Clase Individual.
                */
        Individual();
```

```
        /*! \fn Individual(int& size)
         *
         *  \brief Constructor Clase Individual.
         *
         *  \param size tama&ntilde;o o longitud del cromosoma.
         */
    Individual(int& size);


        /*! \fn ~Individual()
        *  \brief Destructor de la Clase Individual.
        */
    virtual ~Individual();
        /*! \fn evalObjectiveMSLACK(PetriNet pn, int tnow, intvec& pf_id, int ta, const
        int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p,
        intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
        PrtotalWeightedTardiness, double& sumWjTj, double& sumCj, double& sumTj,
        double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec& jobTimes);
         *
         *  \brief Función que calcula el objetivo ponderado del cromosoma usando la
         regla de despacho MSLACK y el vector um para hallar las transiciones
         habilitadas.
         */
        void evalObjectiveMSLACK(PetriNet pn, int tnow, intvec& pf_id, int ta, const
        int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p,
        intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
        PrtotalWeightedTardiness, double& sumWjTj, double& sumCj, double& sumTj,
        double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec& jobTimes);


        /*! \fn evalObjectiveMSLACKIS(PetriNet pn, int tnow, intvec& pf_id, int ta,
         * const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant,
        intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
        PrtotalWeightedTardiness, intvec& PTj);
         *
         * \brief Función que calcula el objetivo ponderado del cromosoma usando la
        regla de despacho MSLACK y el vector um para hallar las transiciones
        habilitadas.
         */
        void evalObjectiveMSLACKIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const
        int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec&
```

```
chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, intvec& PTj);


/*! \fn evalObjectiveEDD(PetriNet pn, int tnow, intvec& pf_id, int ta,const
int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p,
intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, double& sumWjTj, double& sumCj, double& sumTj,
double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec& jobTimes);

 *

 *  \brief Función que calcula el objetivo ponderado del cromosoma usando la
regla de despacho EDD y el vector um para hallar las transiciones habilitadas.

 */

void evalObjectiveEDD(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p,
intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, double& sumWjTj, double& sumCj, double& sumTj,
double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec& jobTimes);


/*! \fn evalObjectiveEDDIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const
int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec&
chrmDec, intvec& chrmCoded, double& PrMakespan, double
PrtotalWeightedTardiness, intvec& PTj);

 *

 *  \brief Función que calcula el objetivo ponderado del cromosoma usando la
regla de despacho EDD y el vector um para hallar las transiciones habilitadas.

 */

void evalObjectiveEDDIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const
int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec&
chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, intvec& PTj);


/*! \fn void evalObjectiveCR(PetriNet pn, int tnow, intvec& pf_id, int ta,
const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant,
int p, intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, double& sumWjTj, double& sumCj, double& sumTj,
double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec& jobTimes);

 *

 *  \brief Función que calcula el objetivo ponderado del cromosoma usando la
regla de despacho CR y el vector um para hallar las transiciones habilitadas.

 */

void evalObjectiveCR(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p,
intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
```

```
PrtotalWeightedTardiness, double& sumWjTj, double& sumCj, double& sumTj,
double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec& jobTimes);


/*! \fn void evalObjectiveCRIS(PetriNet pn, int tnow, intvec& pf_id, int ta,
const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant,
intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, intvec& PTj);              *

 *

 * \brief Función que calcula el objetivo ponderado del cromosoma usando la
regla de despacho CR y el vector um para hallar las transiciones habilitadas.

 */

void evalObjectiveCRIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec&
chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, intvec& PTj);


/*! \fn void evalObjectiveATC(PetriNet pn, int tnow, intvec& pf_id, int ta,
const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant,
int p, intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, double& sumWjTj, double& sumCj, double& sumTj,
double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec& jobTimes);

 *

 *  \brief Función que calcula el objetivo ponderado del cromosoma usando la
regla de despacho ATC y el vector um para hallar las transiciones habilitadas.

 */

void evalObjectiveATC(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p,
intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, double& sumWjTj, double& sumCj, double& sumTj,
double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec& jobTimes);


/*! \fn void evalObjectiveATCIS(PetriNet pn, int tnow, intvec& pf_id, int ta,
const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant,
intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, intvec& PTj);

 *

 *  \brief Función que calcula el objetivo ponderado del cromosoma usando la
regla de despacho ATC y el vector um para hallar las transiciones habilitadas.

 */

void evalObjectiveATCIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const
int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec&
chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, intvec& PTj);
```

```
/*! \fn void evalObjectiveMRWT(PetriNet pn, int tnow, intvec& pf_id, int ta,
const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant,
intvec& temporal);

 *

 *  \brief Función que calcula el objetivo ponderado del cromosoma usando la
regla de despacho MRWT y el vector um para hallar las transiciones habilitadas.

 */

void evalObjectiveMRWT(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p,
intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, double& sumWjTj, double& sumCj, double& sumTj,
double& sumWjCj, int& NoJob_t, intvec& Cj);


/*! \fn void evalObjectiveMRWTIS(PetriNet pn, int tnow, intvec& pf_id, int ta,
const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant,
intvec& temporal);

 *

 *  \brief Función que calcula el objetivo ponderado del cromosoma en la
programación inicial usando la regla de despacho MRWT y el vector ump para
hallar las transiciones habilitadas.

 */

void evalObjectiveMRWTIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const
int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec&
chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness);


/*! \fnvoid evalObjectiveLPT(PetriNet pn, int tnow, intvec& pf_id, int ta,
const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant,
intvec& temporal);

 *

 *  \brief Función que calcula el objetivo ponderado del cromosoma usando la
regla de despacho LPT y el vector um para hallar las transiciones habilitadas.

 */

void evalObjectiveLPT(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p,
intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, double& sumWjTj, double& sumCj, double& sumTj,
double& sumWjCj, int& NoJob_t, intvec& Cj);


/*! \fnvoid evalObjectiveLPTIS(PetriNet pn, int tnow, intvec& pf_id, int ta,
const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant,
intvec& temporal);
```

```
 *

*  \brief Función que calcula el objetivo ponderado del cromosoma en la
programación inicial usando la regla de despacho LPT y el vector ump para
hallar las transiciones habilitadas.

 */

void evalObjectiveLPTIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const
int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec&
chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness);


/*! \fn void evalObjectiveSPT(PetriNet pn, int tnow, intvec& pf_id, int ta,
const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant,
int p, intvec& temporal);

 *

 *  \brief Función que calcula el objetivo ponderado del cromosoma  usando la
regla de despacho SPT y el vector um para hallar las transiciones habilitadas.

 */

void evalObjectiveSPT(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p,
intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, double& sumWjTj, double& sumCj, double& sumTj,
double& sumWjCj, int& NoJob_t, intvec& Cj);


/*! \fnvoid evalObjectiveSPTIS(PetriNet pn, int tnow, intvec& pf_id, int ta,
const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant,
intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness);

 *

 *  \brief Función que calcula el objetivo ponderado del cromosoma en la
programación inicial usando la regla de despacho SPT y el vector ump para
hallar las transiciones habilitadas.

 */

void evalObjectiveSPTIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const
int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec&
chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness);


/*! \fn void evalObjective(PetriNet pn, int tnow, intvec& pf_id, int ta, const
int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p,
intvec& temporal);

 *

 *  \brief Funciónn que calcula el objetivo ponderado del cromosoma utilizando
el vector um para hallar las transiciones habilitadas.
```

```
 */

void evalObjective(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p,
intvec& chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, deque<int>& ur, deque<int>& td, double& sumWjTj,
double& sumCj, double& sumTj, double& sumWjCj, int& NoJob_t, intvec& Cj);


/*! \fn void evalObjective(PetriNet pn, int tnow, intvec& pf_id, int ta, const
int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p,
intvec& temporal);

 *

 *  \brief Funciónn que calcula el objetivo ponderado del cromosoma utilizando
el vector ump para hallar las transiciones habilitadas.

 */

void evalObjectiveIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec&
chrmDec, intvec& chrmCoded, double& PrMakespan, double&
PrtotalWeightedTardiness, deque<int>& ur, deque<int>& td);


/*! \fn void generateScheduleIS(PetriNet pn, int& tnow, vec2dim& sched,
deque<int>& u, intvec& tr, intvec& winnerDec);

 *

 *  \brief Función que dispara la secuencia de transciones obtenidas y genera
el programa inicial de producción

 */

void generateScheduleIS(PetriNet pn, const vector<Tjob>& jobArray, int tnow,
intvec& pf_id, int& NoJob_t, vec2dim& sched, deque<int>& u, intvec& tr, intvec&
winnerDec, vec2dim& transitions, intvec& Cj, double& PrMakespan, double&
PrtotalWeightedTardiness, int& TTPAGIS);


/*! \fn void generateSchedule(PetriNet pn, int& tnow, vec2dim& sched,
deque<int>& u, intvec& tr, intvec& winnerDec);

 *

 *  \brief Función que dispara la secuencia de transciones obtenidas y genera
el programa reactivo de producción

 */

void generateSchedule(PetriNet pn, const vector<Tjob>& jobArray, int& tnow,
intvec& pf_id,int& NoJob_t, vec2dim& sched, deque<int>& u, intvec& tr, intvec&
jobsR, intvec& operationsR, intvec& machinesR, intvec& ptR, intvec& winnerDec,
vec2dim& transitions, intvec& Cj, int sumWjTj, double& PrMakespan, double&
PrtotalWeightedTardiness, double& TTPS);
```

```
        //public


        double fitness;
                double fitness1;
                double fitness2;
                double fitness3;
                int position;
                intvec chrm;
                intvec chrmDec;
};


#endif // !defined(AFX_Individual_H__77B31817_21E0_42AB_BCB4_827B4C401662__INCLUDED_)
```

---

individual.cpp: implementation of the individual class.

```
#include <iostream>
#include <iomanip>
#include "Individual.h"
#include <stdlib.h>
#include "Tjob.h"
#include "WkCenter.h"
#include "PetriNet.h"
#include "Schedule.h"
#include <stdio.h>
#include <vector>
using namespace std;


int findsumremtime (const intvec& Mp, const intvec& Mr, const intvec& rwt, const intvec&
busy_places);
```

```
int findmaxremtime ( const intvec& Mr, const intvec& rwt, const intvec& Mp, const intvec&
busy_places);

int findminremtime (const intvec& Mr, const intvec& rwt, const intvec& busy_places);

int findnextmaxtime(const intvec& um, const intvec& iv, const PetriNet& pn);

int findpresmaxtime(const intvec& um, const intvec& iv, const PetriNet& pn);

int findnextmintime(const intvec& um, const intvec& iv, const PetriNet& pn);

int findnextmintime2(const intvec& um, const intvec& iv, const PetriNet& pn, int&
MinTransition);

int EDD(const intvec& um, const intvec& iv, const PetriNet& pn, const vector<Tjob>& jobArray,
int MachineRestartTime);

int CR(const intvec& um, const intvec& iv, const PetriNet& pn, const vector<Tjob>& jobArray,
intvec& pf_id, int& time, intvec& PTj, intvec& jobTimes);

int ATC(const intvec& um, const intvec& iv, const PetriNet& pn, const vector<Tjob>& jobArray,
intvec& pf_id, int& time, intvec& PTj, intvec& jobTimes);

int MSLACK(const intvec& um, const intvec& iv, const PetriNet& pn, const vector<Tjob>&
jobArray, intvec& pf_id, int& time, intvec& PTj, intvec& jobTimes);

int max(int n, int m);


void jobOperationIndex(const intvec& um ,const vec2dim& transitions, int& oper, int i, int
numJob);


//////////////////////////////////////////////////////////////////////

// Construction/Destruction

//////////////////////////////////////////////////////////////////////


Individual::Individual()

{


}


Individual::Individual(int& size)

{

        chrm.clear();

        chrm.resize(size);


}
```

```
Individual::~Individual()
{

}

bool operator < (const Individual& ind1, const Individual& ind2) {
//boolean operator that checks if ind1.fitness is < ind2.fitness

        return (ind1.fitness < ind2.fitness);
}

ostream& operator << (ostream& os, const Individual& ind1)
{
        os << ind1.fitness;
        return os;
}

void Individual::evalObjectiveSPTIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec& chrmDec, intvec&
chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness) {


        intvec Mf, Mp, Mr, p_id, Cmin, Cplus;
        intvec um, ump;
        int A, delta, r;
        int NoJob_t = 0;
        int MinTransition;
        deque<int> u_activo;
        u_activo.clear();

        constructU(pn,um);
        constructUmp(pn,um, ump);
```

```
int sumWjTj = 0;
int sumCj = 0;
int sumTj = 0;
int sumWjCj = 0;
int Dj ;
int Wj ;
intvec Cj (chrm.size());

int remainingTime;

chrmCoded.clear();
chrmDec.clear();

for (int j = 0; j < chrm.size(); j++) {
                A = findnextmintime(ump,pn.getPT(),pn);
                u_activo.push_back(ump[A]);
                chrmDec.push_back(ump[A]);

                int meme=0;
                int number=0;

                number = rand() % chrm.size();
                meme = (number*ump.size()) + A;
                chrmCoded.push_back(meme);

                int pt = pn.getPT()[pn.getCplus()[ump[A]][0]];
                int pindx = pn.getCmin()[ump[A]][0];
                r = j;
                pn.changeStateR(u_activo, r, delta);
                tnow = tnow + delta;
                if(pn.getp_id()[pn.getCmin()[ump[A]][0]]>=0){
```

```
        int jobNo = jobArray
        [pn.getp_id()[pn.getCmin()[ump[A]][0]]].getNumber();


        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {


                Cj[NoJob_t] = tnow;

                Wj = jobArray
                [pn.getp_id()[pn.getCmin()[ump[A]][0]]].getWeight();

                Dj = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
                ].getDue();

                if (objective == 4) {

                        sumCj = sumCj + Cj[NoJob_t];

                        if ( (Cj[NoJob_t] - Dj) >= 0){

                                sumWjTj = sumWjTj + Wj * (Cj[NoJob_t] -
                                Dj);


                NoJob_t = NoJob_t + 1;

                }

                }

                else if (objective == 5) {

                        sumWjCj = sumWjCj + Wj*(Cj[NoJob_t]);

                        }

                else if (objective == 6) {

                        if ( (Cj[NoJob_t] - Dj) >= 0)

                                sumTj = sumTj + (Cj[NoJob_t] - Dj);

                        }

                else if (objective == 7 || objective == 9) {

                        if ( (Cj[NoJob_t] - Dj) >= 0)

                                sumWjTj = sumWjTj + Wj * (Cj[NoJob_t] -
                                Dj);

                        }

                NoJob_t = NoJob_t + 1;

                }

        else;

}
```

```
                constructU(pn,um);
                constructUmp(pn,um, ump);
        }

        if (objective == 4)  {
                fitness = sumCj;
        }
        else if (objective == 5)  {
                fitness = sumWjCj;
        }
        else if (objective == 6)  {
                fitness = sumTj;
        }
        else if (objective == 7)  {
                fitness = sumWjTj;
        }
        else if (objective == 8)  {
                fitness = tnow;
        }
        else if (objective == 9)  {
                fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);
        }

        fitness1 = tnow;
        fitness2 = sumWjTj;
        fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);

}

void Individual::evalObjectiveLPTIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec& chrmDec, intvec&
chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness) {
```

```
intvec Mf, Mp, Mr, p_id, Cmin, Cplus;
intvec um, ump;

int A, delta, r;
int NoJob_t = 0;
int MinTransition;
deque<int> u_activo;
u_activo.clear();



constructU(pn,um);
constructUmp(pn,um, ump);

int sumWjTj = 0;
int sumCj = 0;
int sumTj = 0;
int sumWjCj = 0;
int Dj ;
int Wj ;
int remainingTime;
intvec Cj (chrm.size());

chrmCoded.clear();
chrmDec.clear();

for(int j=0;j<chrm.size();j++){
                A = findnextmaxtime(ump,pn.getPT(),pn);
                u_activo.push_back(ump[A]);
                chrmDec.push_back(ump[A]);

                int meme=0;
```

```
int numpber=0;


numpber = rand() % chrm.size();

meme = (numpber*ump.size()) + A;

chrmCoded.push_back(meme);


int pt = pn.getPT()[pn.getCplus()[ump[A]][0]];

int pindx = pn.getCmin()[ump[A]][0];

r = j;

pn.changeStateR(u_activo, r, delta);

tnow = tnow + delta;

if(pn.getp_id()[pn.getCmin()[ump[A]][0]]>=0){

        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
        ].getNumber();

        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {


                Cj[NoJob_t] = tnow;

                Wj = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
                ].getWeight();

                Dj = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
                ].getDue();


                if (objective == 4) {

                        sumCj = sumCj + Cj[NoJob_t];


                }

                else if (objective == 5) {

                        sumWjCj = sumWjCj + Wj*(Cj[NoJob_t]);

                        }

                else if (objective == 6) {

                        if ( (Cj[NoJob_t] - Dj) >= 0)

                                sumTj = sumTj + (Cj[NoJob_t] - Dj);

                        }

                else if (objective == 7 || objective == 9) {

                        if ( (Cj[NoJob_t] - Dj) >= 0)
```

```
                                        sumWjTj = sumWjTj + Wj * (Cj[NoJob_t] -
                                        Dj);
                                }
                        NoJob_t = NoJob_t + 1;
                        }
                else;
        }

        constructU(pn,um);
        constructUmp(pn,um, ump);
}

if (objective == 4)  {
        fitness = sumCj;
}
else if (objective == 5)  {
        fitness = sumWjCj;
}
else if (objective == 6)  {
        fitness = sumTj;
}
else if (objective == 7)  {
        fitness = sumWjTj;
}
else if (objective == 8)  {
        fitness = tnow;
}
else if (objective == 9)  {
        fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);
}

fitness1 = tnow;
fitness2 = sumWjTj;
fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);
```

```
}


void Individual::evalObjectiveMRWTIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec& chrmDec, intvec&
chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness) {



        intvec Mf, Mp, Mr, p_id, Cmin, Cplus;

        intvec um;

        intvec ump;

        int A, delta, r;

        int NoJob_t = 0;

        int remainingTime;

        int MinTransition;

        deque<int> u_activo;

        u_activo.clear();



        constructU(pn,um);

        constructUmp(pn,um, ump);



        int sumWjTj = 0;

        int sumCj = 0;

        int sumTj = 0;

        int sumWjCj = 0;

        int Dj ;

        int Wj ;

        intvec Cj (chrm.size());


        chrmCoded.clear();

        chrmDec.clear();
```

```
for (int j = 0; j < chrm.size(); j++) {
                A = findnextmaxtime(ump,pn.getrwt(),pn);
                u_activo.push_back(ump[A]);
                chrmDec.push_back(ump[A]);
                int meme=0;
                int number=0;

                number = rand() % chrm.size();
                meme = (number*ump.size()) + A;
                chrmCoded.push_back(meme);

                int pt = pn.getPT()[pn.getCplus()[ump[A]][0]];
                int pindx = pn.getCmin()[ump[A]][0];
                r = j;
                pn.changeStateR(u_activo, r, delta);
                tnow = tnow + delta;
                if(pn.getp_id()[pn.getCmin()[ump[A]][0]]>=0){
                        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
                        ].getNumber();
                        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {

                                Cj[NoJob_t] = tnow;

                                Wj = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
                                ].getWeight();
                                Dj = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
                                ].getDue();

                                if (objective == 4) {
                                        sumCj = sumCj + Cj[NoJob_t];
                                }
                                else if (objective == 5) {
                                        sumWjCj = sumWjCj + Wj*(Cj[NoJob_t]);
                                        }
```

```
                                else if (objective == 6) {
                                        if ( (Cj[NoJob_t] - Dj) >= 0)
                                                sumTj = sumTj + (Cj[NoJob_t] - Dj);
                                        }
                                else if (objective == 7 || objective == 9) {
                                        if ( (Cj[NoJob_t] - Dj) >= 0)
                                                sumWjTj = sumWjTj + Wj * (Cj[NoJob_t] -
                                                Dj);
                                        }
                                NoJob_t = NoJob_t + 1;
                                }
                        else;
                }


                constructU(pn,um);
                constructUmp(pn,um, ump);
}


if (objective == 4)  {
                fitness = sumCj;
}
else if (objective == 5)  {


                fitness = sumWjCj;
}
else if (objective == 6)  {
                fitness = sumTj;
}
else if (objective == 7)  {
                fitness = sumWjTj;
}
else if (objective == 8)  {
                fitness = tnow;
}
```

```
        else if (objective == 9)  {
                        fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);
        }

        fitness1 = tnow;
        fitness2 = sumWjTj;
        fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);

}

void Individual::evalObjectiveEDDIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec& chrmDec, intvec&
chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, intvec& PTj) {

        intvec Mf, Mp, Mr, p_id, Cmin, Cplus;
        intvec um;
        intvec ump;
        int A, delta, r;
        int NoJob_t = 0;
        int remainingTime;
        int MinTransition;
        deque<int> u_activo;
        u_activo.clear();
        int nJobs = jobArray.size();

        intvec jobTimes;
        jobTimes.resize(nJobs);

        constructU(pn,um);
        constructUmp(pn,um, ump);
```

```
int sumWjTj = 0;

int sumCj = 0;

int sumTj = 0;

int sumWjCj = 0;

int Dj ;

int Wj ;

int MachineRestartTime = 0;

intvec Cj (chrm.size());


chrmCoded.clear();

chrmDec.clear();


for (int j = 0; j < chrm.size(); j++) {
                A = EDD(ump, pn.getPT(), pn, jobArray, MachineRestartTime);

                u_activo.push_back(ump[A]);

                chrmDec.push_back(ump[A]);

                int meme=0;

                int number=0;


                number = rand() % chrm.size();

                meme = (number*ump.size()) + A;

                chrmCoded.push_back(meme);


                int pt = pn.getPT()[pn.getCplus()[ump[A]][0]];

                int pindx = pn.getCmin()[ump[A]][0];

                r = j;

                pn.changeStateR(u_activo, r, delta);

                tnow = tnow + delta;

                if(pn.getp_id()[pn.getCmin()[ump[A]][0]]>=0){

                        int jobNo = jobArray [pn.getp_id() [pn.getCmin()[ump[A]] [0]]
                        ].getNumber();

                        jobTimes[jobNo] = jobTimes[jobNo] + pt;

                        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {
```

```
                                Cj[NoJob_t] = tnow;


                                Wj = jobArray [pn.getp_id()[pn.getCmin()[ump[A]][0]]
                                ].getWeight();

                                Dj = jobArray [pn.getp_id() [pn.getCmin()[ump[A]][0]]
                                ].getDue();


                                if (objective == 4) {

                                        sumCj = sumCj + Cj[NoJob_t];



                                }

                                else if (objective == 5) {

                                        sumWjCj = sumWjCj + Wj*(Cj[NoJob_t]);

                                        }

                                else if (objective == 6) {

                                        if ( (Cj[NoJob_t] - Dj) >= 0)

                                                sumTj = sumTj + (Cj[NoJob_t] - Dj);

                                        }

                                else if (objective == 7 || objective == 9) {

                                        if ( (Cj[NoJob_t] - Dj) >= 0)

                                                sumWjTj = sumWjTj + Wj * (Cj[NoJob_t] -
                                                Dj);

                                        }

                                NoJob_t = NoJob_t + 1;

                                }

                        else;

                }

                constructU(pn,um);

                constructUmp(pn,um, ump);

}


if (objective == 4)  {

                fitness = sumCj;

}

else if (objective == 5)  {
```

```
                fitness = sumWjCj;
        }
        else if (objective == 6)  {
                fitness = sumTj;
        }
        else if (objective == 7)  {
                fitness = sumWjTj;
        }
        else if (objective == 8)  {
                fitness = tnow;
        }
        else if (objective == 9)  {
                fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);
        }

        fitness1 = tnow;
        fitness2 = sumWjTj;
        fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);

}

void Individual::evalObjectiveCRIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec& chrmDec, intvec&
chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, intvec& PTj) {

        intvec Mf, Mp, Mr, p_id, Cmin, Cplus;
        intvec um;
        intvec ump;
        int A, delta, r;
        int NoJob_t = 0;
        int remainingTime;
        int MinTransition;
```

```
deque<int> u_activo;
u_activo.clear();
int nJobs = jobArray.size();

intvec jobTimes;
jobTimes.resize(nJobs);

constructU(pn,um);
constructUmp(pn,um, ump);



int sumWjTj = 0;
int sumCj = 0;
int sumTj = 0;
int sumWjCj = 0;
int Dj ;
int Wj ;
intvec Cj (chrm.size());

chrmCoded.clear();
chrmDec.clear();

for (int j = 0; j < chrm.size(); j++) {
                A = CR(ump, pn.getPT(), pn, jobArray, pf_id, tnow, PTj, jobTimes);

                u_activo.push_back(ump[A]);
                chrmDec.push_back(ump[A]);
                int meme=0;
                int number=0;

                number = rand() % chrm.size();
                meme = (number*ump.size()) + A;
                chrmCoded.push_back(meme);
```

```
int pt = pn.getPT()[pn.getCplus()[ump[A]][0]];



int pindx = pn.getCmin()[ump[A]][0];

r = j;

pn.changeStateR(u_activo, r, delta);

tnow = tnow + delta;

if(pn.getp_id()[pn.getCmin()[ump[A]][0]]>=0){

        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
        ].getNumber();

        jobTimes[jobNo] = jobTimes[jobNo] + pt;

        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {


                Cj[NoJob_t] = tnow;

                Wj = jobArray [pn.getp_id() [pn.getCmin()[ump[A]][0]]
                ].getWeight();

                Dj = jobArray [pn.getp_id()[pn.getCmin()[ump[A]][0]]
                ].getDue();


                if (objective == 4) {

                        sumCj = sumCj + Cj[NoJob_t];


                }

                else if (objective == 5) {

                        sumWjCj = sumWjCj + Wj*(Cj[NoJob_t]);

                        }

                else if (objective == 6) {

                        if ( (Cj[NoJob_t] - Dj) >= 0)

                                sumTj = sumTj + (Cj[NoJob_t] - Dj);

                        }

                else if (objective == 7 || objective == 9) {

                        if ( (Cj[NoJob_t] - Dj) >= 0)

                                sumWjTj = sumWjTj + Wj * (Cj[NoJob_t] -
                                Dj);

                        }
```

```
                                NoJob_t = NoJob_t + 1;
                                }
                        else;
                }

                constructU(pn,um);
                constructUmp(pn,um, ump);
        }

        if (objective == 4)  {
                fitness = sumCj;
        }
        else if (objective == 5)  {
                fitness = sumWjCj;
        }
        else if (objective == 6)  {
                fitness = sumTj;
        }
        else if (objective == 7)  {
                fitness = sumWjTj;
        }
        else if (objective == 8)  {
                fitness = tnow;
        }
        else if (objective == 9)  {
                fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);
        }

        fitness1 = tnow;
        fitness2 = sumWjTj;
        fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);

}
```

```
void Individual::evalObjectiveATCIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec& chrmDec, intvec&
chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, intvec& PTj) {



        intvec Mf, Mp, Mr, p_id, Cmin, Cplus;

        intvec um;

        intvec ump;

        int A, delta, r;

        int NoJob_t = 0;

        int remainingTime;

        int MinTransition;

        deque<int> u_activo;

        u_activo.clear();

        int nJobs = jobArray.size();


        intvec jobTimes;

        jobTimes.resize(nJobs);


        constructU(pn,um);

        constructUmp(pn,um, ump);



        int sumWjTj = 0;

        int sumCj = 0;

        int sumTj = 0;

        int sumWjCj = 0;

        int Dj ;

        int Wj ;

        intvec Cj (chrm.size());


        chrmCoded.clear();
```

```
chrmDec.clear();


for (int j = 0; j < chrm.size(); j++) {

                A = ATC(ump, pn.getPT(), pn, jobArray, pf_id, tnow, PTj, jobTimes);
                u_activo.push_back(ump[A]);
                chrmDec.push_back(ump[A]);
                int meme=0;
                int number=0;


                number = rand() % chrm.size();
                meme = (number*ump.size()) + A;
                chrmCoded.push_back(meme);


                int pt = pn.getPT()[pn.getCplus()[ump[A]][0]];



                int pindx = pn.getCmin()[ump[A]][0];
                r = j;
                pn.changeStateR(u_activo, r, delta);
                tnow = tnow + delta;
                if(pn.getp_id()[pn.getCmin()[ump[A]][0]]>=0){
                        int jobNo = jobArray [pn.getp_id()[pn.getCmin()[ump[A]][0]]
                        ].getNumber();
                        jobTimes[jobNo] = jobTimes[jobNo] + pt;
                        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {

                                Cj[NoJob_t] = tnow;


                                Wj = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
                                ].getWeight();
                                Dj = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
                                ].getDue();


                                if (objective == 4) {
```

```
                                        sumCj = sumCj + Cj[NoJob_t];


                                }
                                else if (objective == 5) {
                                        sumWjCj = sumWjCj + Wj*(Cj[NoJob_t]);
                                        }
                                else if (objective == 6) {
                                        if ( (Cj[NoJob_t] - Dj) >= 0)
                                                sumTj = sumTj + (Cj[NoJob_t] - Dj);
                                        }
                                else if (objective == 7 || objective == 9) {
                                        if ( (Cj[NoJob_t] - Dj) >= 0)
                                                sumWjTj = sumWjTj + Wj * (Cj[NoJob_t] -
                                                Dj);
                                        }
                                NoJob_t = NoJob_t + 1;
                                }
                        else;
                }


                constructU(pn,um);
                constructUmp(pn,um, ump);
}


if (objective == 4)  {
                fitness = sumCj;
}
else if (objective == 5)  {
                fitness = sumWjCj;
}
else if (objective == 6)  {
                fitness = sumTj;
}
else if (objective == 7)  {
```

```
                fitness = sumWjTj;
        }
        else if (objective == 8)  {
                fitness = tnow;
        }
        else if (objective == 9)  {
                fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);
        }

        fitness1 = tnow;
        fitness2 = sumWjTj;
        fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);

}

void Individual::evalObjectiveMSLACKIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const
int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec& chrmDec,
intvec& chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, intvec& PTj) {

        intvec Mf, Mp, Mr, p_id, Cmin, Cplus;
        intvec um;
        intvec ump;
        int A, delta, r;
        int NoJob_t = 0;
        int remainingTime;
        int MinTransition;
        deque<int> u_activo;
        u_activo.clear();
        int nJobs = jobArray.size();

        intvec jobTimes;
        jobTimes.resize(nJobs);
```

```
constructU(pn,um);
constructUmp(pn,um, ump);



int sumWjTj = 0;
int sumCj = 0;
int sumTj = 0;
int sumWjCj = 0;
int Dj ;
int Wj ;
intvec Cj (chrm.size());

chrmCoded.clear();
chrmDec.clear();

for (int j = 0; j < chrm.size(); j++) {

                A = MSLACK(ump,pn.getPT(),pn,jobArray, pf_id, tnow, PTj, jobTimes);

                u_activo.push_back(ump[A]);
                chrmDec.push_back(ump[A]);
                int meme=0;
                int number=0;

                number = rand() % chrm.size();
                meme = (number*ump.size()) + A;
                chrmCoded.push_back(meme);

                int pt = pn.getPT()[pn.getCplus()[ump[A]][0]];
                int pindx = pn.getCmin()[ump[A]][0];
                r = j;
                pn.changeStateR(u_activo, r, delta);
```

```
tnow = tnow + delta;

if(pn.getp_id()[pn.getCmin()[ump[A]][0]]>=0){

        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
        ].getNumber();

        jobTimes[jobNo] = jobTimes[jobNo] + pt;

        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {


                Cj[NoJob_t] = tnow;

                Wj = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
                ].getWeight();

                Dj = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
                ].getDue();



                if (objective == 4) {

                        sumCj = sumCj + Cj[NoJob_t];

                }

                else if (objective == 5) {

                        sumWjCj = sumWjCj + Wj*(Cj[NoJob_t]);

                        }

                else if (objective == 6) {

                        if ( (Cj[NoJob_t] - Dj) >= 0)

                                sumTj = sumTj + (Cj[NoJob_t] - Dj);

                        }

                else if (objective == 7 || objective == 9) {

                        if ( (Cj[NoJob_t] - Dj) >= 0)

                                sumWjTj = sumWjTj + Wj * (Cj[NoJob_t] -
                                Dj);

                        }

                NoJob_t = NoJob_t + 1;

                }

        else;

}


constructU(pn,um);

constructUmp(pn,um, ump);
```

```
	}


	if (objective == 4)  {
			fitness = sumCj;
	}
	else if (objective == 5)  {
			fitness = sumWjCj;
	}
	else if (objective == 6)  {
			fitness = sumTj;
	}
	else if (objective == 7)  {
			fitness = sumWjTj;
	}
	else if (objective == 8)  {
			fitness = tnow;
	}
	else if (objective == 9)  {
			fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);
	}


	fitness1 = tnow;
	fitness2 = sumWjTj;
	fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);


}



void Individual::evalObjectiveIS(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec& chrmDec, intvec&
chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, deque<int>& ur, deque<int>&
td) {
```

```
intvec Mf, Mp, Mr, p_id, Cmin, Cplus;
intvec um, ump;
int A, delta, r;
int NoJob_t = 0;
int MinTransition;
deque<int> u_activo;
u_activo.clear();

intvec shootTimes;
shootTimes.clear();
chrmDec.clear();



constructU(pn,um);
constructUmp(pn, um, ump);

int sumWjTj = 0;
int sumCj = 0;
int sumTj = 0;
int sumWjCj = 0;
int Dj ;
int Wj ;
intvec Cj (chrm.size());
int remainingTime;
int temp = 0;
int n = 0;
int pt;

for (int j = 0; j < chrm.size(); j++) {
        int temp = 0;
        int minMr = 1000000;
        int cont1 = 0;
```

```
A = chrm[j] % ump.size();

u_activo.push_back(ump[A]);

chrmDec.push_back(ump[A]);


int pt = pn.getPT()[pn.getCplus()[ump[A]][0]];

int pindx = pn.getCmin()[ump[A]][0];


r = j;

pn.changeStateR(u_activo, r, delta);


tnow = tnow + delta;


shootTimes.push_back(tnow);


if(pn.getp_id()[pn.getCmin()[ump[A]][0]]>=0){

        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
        ].getNumber();


        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {

                Cj[NoJob_t] = tnow;

                Wj = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]
                ].getWeight();

                Dj = jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]].getDue();

                if (objective == 4) {

                        sumCj = sumCj + Cj[NoJob_t];

                }

                else if (objective == 5) {

                        sumWjCj = sumWjCj + Wj*(Cj[NoJob_t]);

                        }

                else if (objective == 6) {

                        if ( (Cj[NoJob_t] - Dj) >= 0)

                                sumTj = sumTj + (Cj[NoJob_t] - Dj);

                        }
```

```
                        else if (objective == 7 || objective == 9) {
                                if ( (Cj[NoJob_t] - Dj) >= 0)
                                        sumWjTj = sumWjTj + Wj * (Cj[NoJob_t] - Dj);
                                }
                        NoJob_t = NoJob_t + 1;
                        }
                else;
        }

        constructU(pn,um);
        constructUmp(pn,um, ump);


}

if (objective == 4)  {
                fitness = sumCj;
}
else if (objective == 5)  {
                fitness = sumWjCj;
}
else if (objective == 6)  {
                fitness = sumTj;
}
else if (objective == 7)  {
                fitness = sumWjTj;
}
else if (objective == 8)  {
                fitness = tnow;
}
else if (objective == 9)  {
                fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);

}
```

```
        fitness1 = tnow;

        fitness2 = sumWjTj;

        fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTj);


}



void Individual::evalObjectiveSPT(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p, intvec& chrmDec,
intvec& chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, double& sumWjTj,
double& sumCj, double& sumTj, double& sumWjCj, int& NoJob_t, intvec& Cj) {



        intvec Mf, Mp, Mr, p_id, Cmin, Cplus;

        intvec um, ump;

        int A, delta, r;

        int MinTransition;

        deque<int> u_activo;

        u_activo.clear();



        constructU(pn,um);

        constructUmp(pn,um, ump);


        double sumWjTjRS = sumWjTj;

        double sumCjRS = sumCj;

        double sumTjRS = sumTj;

        double sumWjCjRS = sumWjCj;

        int NoJob_tRS = NoJob_t;

        int Dj ;

        int Wj ;

        int pt;

        intvec CjRS;
```

```
CjRS.clear();
for(int i=0; i<Cj.size();i++){
        CjRS.push_back(Cj[i]);
}               int remainingTime;


chrmCoded.clear();
chrmDec.clear();


for (int j = 0; j < chrm.size(); j++) {


        int temp = 0;
        int cont1 = 0;


        int posRepair;


                for(int i = 0; i < um.size(); i++){
                        int pindx = pn.getCmin()[um[i]][0];
                        pt = pn.getPT()[pn.getCplus()[um[i]][0]];
                        if(pn.getp_id()[pn.getCmin()[um[i]][0]] < 0 && pn.getPT()
                        [pn.getCmin()[um[i]][0]] > 0){
                                cont1 = cont1 + 1;
                                posRepair=i;
                        }


                }


                int NextMinTime;
                //Se encuentra la posición de la transición habilitada que inicia la
                //operación con mínimo tiempo de procesamiento
                NextMinTime = findnextmintime2(um,pn.getPT(),pn, MinTransition);


                if(um.size() > 1 && cont1 > 0  && (tnow + NextMinTime >=
                mchBreakdownInstant + p)){
```

```
        A = posRepair;
}else{
        A = findnextmintime(um,pn.getPT(),pn);

}

int meme=0;
int number=0;
int cont3 = 0;

number = rand() % chrm.size();
meme = (number*(um.size())) + A;
chrm[j] = meme;

if(j == chrm.size()-1){
        if(pn.getp_id()[pn.getCmin()[um[0]][0]]<0){
                break;
        }
}



u_activo.push_back(um[A]);
chrmDec.push_back(um[A]);
pt = pn.getPT()[pn.getCplus()[um[A]][0]];

int pindx = pn.getCmin()[um[A]][0];
r = j;
pn.changeStateR(u_activo, r, delta);
tnow = tnow + delta;

if(pn.getp_id()[pn.getCmin()[um[A]][0]]>=0){
        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
        ].getNumber();
```

```
        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {


                CjRS[NoJob_tRS] = tnow;

                Wj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
                ].getWeight();

                Dj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
                ].getDue();

                if (objective == 4) {

                        sumCjRS = sumCjRS + CjRS[NoJob_tRS];

                        if ( (CjRS[NoJob_tRS] - Dj) >= 0){

                                sumWjTjRS = sumWjTjRS + Wj *
                                (CjRS[NoJob_tRS] - Dj);


                NoJob_tRS = NoJob_tRS + 1;

                }

                }

                else if (objective == 5) {

                        sumWjCjRS = sumWjCjRS + Wj*(CjRS[NoJob_tRS]);

                        }

                else if (objective == 6) {

                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)

                                sumTjRS = sumTjRS + (CjRS[NoJob_tRS] -
                                Dj);

                        }

                else if (objective == 7 || objective == 9) {

                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)

                                sumWjTjRS = sumWjTjRS + Wj *
                                (CjRS[NoJob_tRS] - Dj);

                        }

                NoJob_tRS = NoJob_tRS + 1;

                }

        else;

}


if(pn.getp_id()[pn.getCmin()[um[A]][0]]<0){

        int q = pn.getCmin()[um[A]][0];
```

```
                        int repairTime = 100000000;
                        pn.changePT( repairTime, q, pn);
                }

                constructU(pn,um);
                constructUmp(pn,um, ump);
        }

        if (objective == 4)  {
                fitness = sumCjRS;
        }
        else if (objective == 5)  {
                fitness = sumWjCjRS;
        }
        else if (objective == 6)  {
                fitness = sumTjRS;
        }
        else if (objective == 7)  {
                fitness = sumWjTjRS;
        }
        else if (objective == 8)  {
                fitness = tnow;
        }
        else if (objective == 9)  {
                fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);

        }
        fitness1 = tnow;
        fitness2 = sumWjTjRS;
        fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);

}
```

```
void Individual::evalObjectiveLPT(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p, intvec& chrmDec,
intvec& chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, double& sumWjTj,
double& sumCj, double& sumTj, double& sumWjCj, int& NoJob_t, intvec& Cj) {



        intvec Mf, Mp, Mr, p_id, Cmin, Cplus;

        intvec um, ump;


        int A, delta, r;

        int MinTransition;

        deque<int> u_activo;

        u_activo.clear();



        constructU(pn,um);

        constructUmp(pn,um, ump);

        double sumWjTjRS = sumWjTj;

        double sumCjRS = sumCj;

        double sumTjRS = sumTj;

        double sumWjCjRS = sumWjCj;

        int NoJob_tRS = NoJob_t;

        int Dj ;

        int Wj ;

        int pt;

        int remainingTime;

        intvec CjRS;

        CjRS.clear();

        for(int i=0; i<Cj.size();i++){

                CjRS.push_back(Cj[i]);

        }


        chrmCoded.clear();
```

```
chrmDec.clear();


for(int j=0;j<chrm.size();j++){
        int temp = 0;
        int cont1 = 0;


                int posRepair;


                for(int i = 0; i < um.size(); i++){
                        int pindx = pn.getCmin()[um[i]][0];
                        pt = pn.getPT()[pn.getCplus()[um[i]][0]];
                        if(pn.getp_id()[pn.getCmin()[um[i]][0]] < 0 &&
                        pn.getPT()[pn.getCmin()[um[i]][0]] > 0){
                                cont1 = cont1 + 1;
                                posRepair=i;
                        }
                }


                int NextMinTime;
                //Se encuentra la posición de la transición habilitada que inicia la
operación con mínimo tiempo de procesamiento
                NextMinTime = findnextmintime2(um,pn.getPT(),pn, MinTransition);


                if(um.size() > 1 && cont1 > 0  && (tnow + NextMinTime >=
                mchBreakdownInstant + p)){
                        A = posRepair;
                }else{
                        A = findnextmaxtime(um,pn.getPT(),pn);


                }


                //Se codifica la transición


                int meme=0;
                int number=0;
```

```
int cont3 = 0;


number = rand() % chrm.size();

meme = (number*(um.size())) + A;

chrm[j] = meme;


if(j == chrm.size()-1){

        if(pn.getp_id()[pn.getCmin()[um[0]][0]]<0){

                break;

        }

}


u_activo.push_back(um[A]);

chrmDec.push_back(um[A]);




int pt = pn.getPT()[pn.getCplus()[um[A]][0]];


int pindx = pn.getCmin()[um[A]][0];

r = j;

pn.changeStateR(u_activo, r, delta);

tnow = tnow + delta;



if(pn.getp_id()[pn.getCmin()[um[A]][0]]>=0){

        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
        ].getNumber();

        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {


                CjRS[NoJob_tRS] = tnow;

                Wj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
                ].getWeight();
```

```
                Dj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
                ].getDue();


                if (objective == 4) {
                        sumCjRS = sumCjRS + CjRS[NoJob_tRS];

                }
                else if (objective == 5) {
                        sumWjCjRS = sumWjCjRS + Wj*(CjRS[NoJob_tRS]);
                        }
                else if (objective == 6) {
                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)
                                sumTjRS = sumTjRS + (CjRS[NoJob_tRS] -
                                Dj);
                        }
                else if (objective == 7 || objective == 9) {
                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)
                                sumWjTjRS = sumWjTjRS + Wj *
                                (CjRS[NoJob_tRS] - Dj);
                        }
                NoJob_tRS = NoJob_tRS + 1;
                }
        else;
}



if(pn.getp_id()[pn.getCmin()[um[A]    ][0]]<0){
        int q = pn.getCmin()[um[A]][0];
        int repairTime = 100000000;
        pn.changePT(repairTime, q, pn);

}


constructU(pn,um);
constructUmp(pn,um, ump);
```

```
	}


	if (objective == 4)  {
			fitness = sumCjRS;
	}
	else if (objective == 5)  {
			fitness = sumWjCjRS;
	}
	else if (objective == 6)  {
			fitness = sumTjRS;
	}
	else if (objective == 7)  {
			fitness = sumWjTjRS;
	}
	else if (objective == 8)  {
			fitness = tnow;
	}
	else if (objective == 9)  {
			fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);

	}


	fitness1 = tnow;
	fitness2 = sumWjTjRS;
	fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);

}



void Individual::evalObjectiveMRWT(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p, intvec& chrmDec,
intvec& chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, double& sumWjTj,
double& sumCj, double& sumTj, double& sumWjCj, int& NoJob_t, intvec& Cj) {
```

```
intvec Mf, Mp, Mr, p_id, Cmin, Cplus;
intvec um;
intvec ump;
int A, delta, r;
int remainingTime;
int MinTransition;
deque<int> u_activo;
u_activo.clear();



constructU(pn,um);
constructUmp(pn,um, ump);
double sumWjTjRS = sumWjTj;
double sumCjRS = sumCj;
double sumTjRS = sumTj;
double sumWjCjRS = sumWjCj;
int NoJob_tRS = NoJob_t;

int Dj ;
int Wj ;
intvec CjRS;
CjRS.clear();
for(int i=0; i<Cj.size();i++){
        CjRS.push_back(Cj[i]);
}

chrmCoded.clear();
chrmDec.clear();

for (int j = 0; j < chrm.size(); j++) {
        int temp = 0;
        int cont1 = 0;
```

```
                int posRepair;

                int pt;


                //Se encuentra la posición de la transición de rehabilitación de la máquina en
                //el vector um si aún está habilitada

                for(int i = 0; i < um.size(); i++){

                                int pindx = pn.getCmin()[um[i]][0];

                                pt = pn.getPT()[pn.getCplus()[um[i]][0]];

                                if(pn.getp_id()[pn.getCmin()[um[i]][0]] < 0 &&
pn.getPT()[pn.getCmin()[um[i]][0]] > 0){

                                        cont1 = cont1 + 1;

                                        posRepair=i;

                                }

                }


                int NextMinTime;

                //Se encuentra la posición de la transición habilitada que inicia la operación
                //con mínimo tiempo de procesamiento

                NextMinTime = findnextmintime2(um,pn.getPT(),pn, MinTransition);


                if(um.size() > 1 && cont1 > 0  && (tnow + NextMinTime >= mchBreakdownInstant +
                p)){

                        A = posRepair;

                }else{

                        A = findnextmaxtime(um,pn.getrwt(),pn);

                }




                //Se codifica la transición


                int meme=0;

                int number=0;


                int cont3 = 0;
```

```
number = rand() % chrm.size();
meme = (number*(um.size())) + A;
chrm[j] = meme;

if(j == chrm.size()-1){
        if(pn.getp_id()[pn.getCmin()[um[0]][0]]<0){
                break;
        }
}

u_activo.push_back(um[A]);
chrmDec.push_back(um[A]);



pt = pn.getPT()[pn.getCplus()[um[A]][0]];
int pindx = pn.getCmin()[um[A]][0];
r = j;
pn.changeStateR(u_activo, r, delta);
tnow = tnow + delta;

if(pn.getp_id()[pn.getCmin()[um[A]][0]]>=0){
        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]].getNumber();

        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {

                CjRS[NoJob_tRS] = tnow;

                Wj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]].getWeight();
                Dj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]].getDue();

                if (objective == 4) {
                        sumCjRS = sumCjRS + Cj[NoJob_tRS];
                }
                else if (objective == 5) {
```

```
                                sumWjCjRS = sumWjCjRS + Wj*(CjRS[NoJob_tRS]);
                                }
                        else if (objective == 6) {
                                if ( (CjRS[NoJob_tRS] - Dj) >= 0)
                                        sumTjRS = sumTjRS + (CjRS[NoJob_tRS] - Dj);
                                }
                        else if (objective == 7 || objective == 9) {
                                if ( (CjRS[NoJob_tRS] - Dj) >= 0)
                                        sumWjTjRS = sumWjTjRS + Wj * (CjRS[NoJob_tRS] -
                                        Dj);
                                }
                        NoJob_tRS = NoJob_tRS + 1;
                        }
                else;
        }


        //Cuando se termina de reparar la máquina, la plaza de repaaración vuelve a
        //tomar el valor inicial
        if(pn.getp_id()[pn.getCmin()[um[A]][0]]<0){
                int q = pn.getCmin()[um[A]][0];
                int repairTime = 100000000;
                pn.changePT( repairTime, q, pn);
        }


        constructU(pn,um);
        constructUmp(pn,um, ump);
}


if (objective == 4)  {
                fitness = sumCjRS;
}
else if (objective == 5)  {
                fitness = sumWjCjRS;
}
```

```
        else if (objective == 6)  {
                        fitness = sumTjRS;
        }
        else if (objective == 7)  {
                        fitness = sumWjTjRS;
        }
        else if (objective == 8)  {
                        fitness = tnow;
        }
        else if (objective == 9)  {
                        fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);
        }

        fitness1 = tnow;
        fitness2 = sumWjTjRS;
        fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);

}

void Individual::evalObjectiveEDD(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p, intvec& chrmDec,
intvec& chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, double& sumWjTj,
double& sumCj, double& sumTj, double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec&
jobTimes) {

        intvec Mf, Mp, Mr, p_id, Cmin, Cplus;
        intvec um;
        intvec ump;
        int A, delta, r;
        int remainingTime;
        int MinTransition;
        deque<int> u_activo;
        u_activo.clear();
```

```
constructU(pn,um);

constructUmp(pn,um, ump);

double sumWjTjRS = sumWjTj;

double sumCjRS = sumCj;

double sumTjRS = sumTj;

double sumWjCjRS = sumWjCj;

int NoJob_tRS = NoJob_t;


int Dj ;

int Wj ;

intvec CjRS;

CjRS.clear();

for(int i=0; i<Cj.size();i++){

        CjRS.push_back(Cj[i]);

}


chrmCoded.clear();

chrmDec.clear();


for (int j = 0; j < chrm.size(); j++) {


        int temp = 0;

        int cont1 = 0;

        int posRepair;

        int pt;


        //Se encuentra la posición de la transición de rehabilitación de la máquina en
        //el vector um si aún está habilitada

        for(int i = 0; i < um.size(); i++){

                int pindx = pn.getCmin()[um[i]][0];

                pt = pn.getPT()[pn.getCplus()[um[i]][0]];
```

```
        if(pn.getp_id()[pn.getCmin()[um[i]][0]] < 0 &&
        pn.getPT()[pn.getCmin()[um[i]][0]] > 0){

                cont1 = cont1 + 1;

                posRepair=i;

        }

}


        int NextMinTime;

        //Se encuentra la posición de la transición habilitada que inicia la
        //operación con míimo tiempo de procesamiento

        NextMinTime = findnextmintime2(um,pn.getPT(),pn, MinTransition);


        int MachineRestartTime = mchBreakdownInstant + p;


        if(um.size() > 1 && cont1 > 0  && (tnow + NextMinTime >=
        mchBreakdownInstant + p)){

                A = posRepair;

        }else{

                A = EDD(um,pn.getPT(),pn,jobArray, MachineRestartTime);


        }


        //Se codifica la transición


        int meme=0;

        int number=0;

        int cont3 = 0;


        number = rand() % chrm.size();

        meme = (number*(um.size())) + A;

        chrm[j] = meme;


        if(j == chrm.size()-1){

                if(pn.getp_id()[pn.getCmin()[um[0]][0]]<0){

                        break;
```

```
        }
}


u_activo.push_back(um[A]);

chrmDec.push_back(um[A]);

pt = pn.getPT()[pn.getCplus()[um[A]][0]];


int pindx = pn.getCmin()[um[A]][0];

r = j;

pn.changeStateR(u_activo, r, delta);

tnow = tnow + delta;


if(pn.getp_id()[pn.getCmin()[um[A]][0]]>=0){

        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
        ].getNumber();

        jobTimes[jobNo] = jobTimes[jobNo] + pt;

        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {

                CjRS[NoJob_tRS] = tnow;

                Wj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
                ].getWeight();

                Dj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
                ].getDue();


                if (objective == 4) {

                        sumCjRS = sumCjRS + Cj[NoJob_tRS];


                }

                else if (objective == 5) {

                        sumWjCjRS = sumWjCjRS + Wj*(CjRS[NoJob_tRS]);

                        }

                else if (objective == 6) {

                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)

                                sumTjRS = sumTjRS + (CjRS[NoJob_tRS] -
                                Dj);

                        }
```

```
                                else if (objective == 7 || objective == 9) {
                                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)
                                                sumWjTjRS = sumWjTjRS + Wj *
                                                (CjRS[NoJob_tRS] - Dj);
                                        }
                                NoJob_tRS = NoJob_tRS + 1;
                                }
                        else;
                }


                //Cuando se termina de reparar la máquina, la plaza de repaaración
                //vuelve a tomar el valor inicial
                if(pn.getp_id()[pn.getCmin()[um[A]][0]]<0){
                        int q = pn.getCmin()[um[A]][0];
                        int repairTime = 100000000;
                        pn.changePT( repairTime, q, pn);
                }


                constructU(pn,um);
                constructUmp(pn,um, ump);
}


if (objective == 4)  {
                fitness = sumCjRS;
}
else if (objective == 5)  {


                fitness = sumWjCjRS;
}
else if (objective == 6)  {
                fitness = sumTjRS;
}
else if (objective == 7)  {
                fitness = sumWjTjRS;
```

```
	}
	else if (objective == 8)  {
			fitness = tnow;
	}
	else if (objective == 9)  {
			fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);
	}

	fitness1 = tnow;
	fitness2 = sumWjTjRS;
	fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);

}

void Individual::evalObjectiveCR(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p, intvec& chrmDec,
intvec& chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, double& sumWjTj,
double& sumCj, double& sumTj, double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec&
jobTimes) {

	intvec Mf, Mp, Mr, p_id, Cmin, Cplus;
	intvec um;
	intvec ump;
	int A, delta, r;
	int remainingTime;
	int MinTransition;
	deque<int> u_activo;
	u_activo.clear();

	constructU(pn,um);
	constructUmp(pn,um, ump);
	double sumWjTjRS = sumWjTj;
```

```
double sumCjRS = sumCj;

double sumTjRS = sumTj;

double sumWjCjRS = sumWjCj;

int NoJob_tRS = NoJob_t;


int Dj ;

int Wj ;

intvec CjRS;

CjRS.clear();

for(int i=0; i<Cj.size();i++){

        CjRS.push_back(Cj[i]);

}


chrmCoded.clear();

chrmDec.clear();


for (int j = 0; j < chrm.size(); j++) {


        int temp = 0;

        int cont1 = 0;

        int posRepair;

        int pt;


        //Se encuentra la posición de la transición de rehabilitación de la máquina en
        //el vector um si aún está habilitada

        for(int i = 0; i < um.size(); i++){

                int pindx = pn.getCmin()[um[i]][0];

                pt = pn.getPT()[pn.getCplus()[um[i]][0]];

                if(pn.getp_id()[pn.getCmin()[um[i]][0]] < 0 &&
                pn.getPT()[pn.getCmin()[um[i]][0]] > 0){

                        cont1 = cont1 + 1;

                        posRepair=i;

                }

        }
```

```
int NextMinTime;

//Se encuentra la posición de la transición habilitada que inicia la
//operación con míimo tiempo de procesamiento

NextMinTime = findnextmintime2(um,pn.getPT(),pn, MinTransition);


if(um.size() > 1 && cont1 > 0  && (tnow + NextMinTime >=
mchBreakdownInstant + p)){

        A = posRepair;

}else{

        A = CR(um,pn.getPT(),pn,jobArray,pf_id, tnow, PTj, jobTimes);


}


//Se codifica la transición


int meme=0;

int number=0;

int cont3 = 0;


number = rand() % chrm.size();

meme = (number*(um.size())) + A;

chrm[j] = meme;


if(j == chrm.size()-1){

        if(pn.getp_id()[pn.getCmin()[um[0]][0]]<0){

                break;

        }

}


u_activo.push_back(um[A]);

chrmDec.push_back(um[A]);


pt = pn.getPT()[pn.getCplus()[um[A]][0]];
```

```
int pindx = pn.getCmin()[um[A]][0];

r = j;

pn.changeStateR(u_activo, r, delta);

tnow = tnow + delta;


if(pn.getp_id()[pn.getCmin()[um[A]][0]]>=0){

        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
        ].getNumber();

        jobTimes[jobNo] = jobTimes[jobNo] + pt;

        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {

                CjRS[NoJob_tRS] = tnow;

                Wj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
                ].getWeight();

                Dj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
                ].getDue();


                if (objective == 4) {

                        sumCjRS = sumCjRS + Cj[NoJob_tRS];


                }

                else if (objective == 5) {

                        sumWjCjRS = sumWjCjRS + Wj*(CjRS[NoJob_tRS]);

                        }

                else if (objective == 6) {

                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)

                                sumTjRS = sumTjRS + (CjRS[NoJob_tRS] -
                                Dj);

                        }

                else if (objective == 7 || objective == 9) {

                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)

                                sumWjTjRS = sumWjTjRS + Wj *
                                (CjRS[NoJob_tRS] - Dj);

                        }

                NoJob_tRS = NoJob_tRS + 1;

                }
```

```
                        else;
                }


                //Cuando se termina de reparar la máquina, la plaza de repaaración
                //vuelve a tomar el valor inicial
                if(pn.getp_id()[pn.getCmin()[um[A]][0]]<0){
                        int q = pn.getCmin()[um[A]][0];
                        int repairTime = 100000000;
                        pn.changePT( repairTime, q, pn);
                }


                constructU(pn,um);
                constructUmp(pn,um, ump);



}


if (objective == 4)  {
                fitness = sumCjRS;
}
else if (objective == 5)  {


                fitness = sumWjCjRS;
}
else if (objective == 6)  {
                fitness = sumTjRS;
}
else if (objective == 7)  {
                fitness = sumWjTjRS;
}
else if (objective == 8)  {
                fitness = tnow;
}
else if (objective == 9)  {
```

```
                        fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);
        }


        fitness1 = tnow;
        fitness2 = sumWjTjRS;
        fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);


}



void Individual::evalObjectiveATC(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p, intvec& chrmDec,
intvec& chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, double& sumWjTj,
double& sumCj, double& sumTj, double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec&
jobTimes) {



        intvec Mf, Mp, Mr, p_id, Cmin, Cplus;
        intvec um;
        intvec ump;
        int A, delta, r;
        int remainingTime;
        int MinTransition;
        deque<int> u_activo;
        u_activo.clear();



        constructU(pn,um);
        constructUmp(pn,um, ump);
        double sumWjTjRS = sumWjTj;
        double sumCjRS = sumCj;
        double sumTjRS = sumTj;
        double sumWjCjRS = sumWjCj;
        int NoJob_tRS = NoJob_t;
```

```
int Dj ;

int Wj ;

intvec CjRS;

CjRS.clear();

for(int i=0; i<Cj.size();i++){

        CjRS.push_back(Cj[i]);

}


chrmCoded.clear();

chrmDec.clear();


for (int j = 0; j < chrm.size(); j++) {


        int temp = 0;

        int cont1 = 0;

        int posRepair;

        int pt;


        //Se encuentra la posición de la transición de rehabilitación de la máquina en
        //el vector um si aún está habilitada

        for(int i = 0; i < um.size(); i++){

                        int pindx = pn.getCmin()[um[i]][0];

                        pt = pn.getPT()[pn.getCplus()[um[i]][0]];

                        if(pn.getp_id()[pn.getCmin()[um[i]][0]] < 0 &&
                        pn.getPT()[pn.getCmin()[um[i]][0]] > 0){

                                cont1 = cont1 + 1;

                                posRepair=i;

                        }

        }


                int NextMinTime;

                //Se encuentra la posición de la transición habilitada que inicia la
                //operación con míimo tiempo de procesamiento

                NextMinTime = findnextmintime2(um,pn.getPT(),pn, MinTransition);
```

```
if(um.size() > 1 && cont1 > 0  && (tnow + NextMinTime >=
mchBreakdownInstant + p)){

        A = posRepair;

}else{

        A = ATC(um,pn.getPT(),pn,jobArray,pf_id, tnow, PTj, jobTimes);


}


//Se codifica la transición


int meme=0;

int number=0;


int cont3 = 0;


number = rand() % chrm.size();

meme = (number*(um.size())) + A;

chrm[j] = meme;


if(j == chrm.size()-1){

        if(pn.getp_id()[pn.getCmin()[um[0]][0]]<0){

                break;

        }

}

u_activo.push_back(um[A]);

chrmDec.push_back(um[A]);



pt = pn.getPT()[pn.getCplus()[um[A]][0]];

int pindx = pn.getCmin()[um[A]][0];

r = j;

pn.changeStateR(u_activo, r, delta);

tnow = tnow + delta;
```

```
if(pn.getp_id()[pn.getCmin()[um[A]][0]]>=0){

        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
        ].getNumber();

        jobTimes[jobNo] = jobTimes[jobNo] + pt;

        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {

                CjRS[NoJob_tRS] = tnow;

                Wj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
                ].getWeight();

                Dj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
                ].getDue();


                if (objective == 4) {

                        sumCjRS = sumCjRS + Cj[NoJob_tRS];

                }

                else if (objective == 5) {

                        sumWjCjRS = sumWjCjRS + Wj*(CjRS[NoJob_tRS]);

                        }

                else if (objective == 6) {

                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)

                                sumTjRS = sumTjRS + (CjRS[NoJob_tRS] -
                                Dj);

                        }

                else if (objective == 7 || objective == 9) {

                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)

                                sumWjTjRS = sumWjTjRS + Wj *
                                (CjRS[NoJob_tRS] - Dj);

                        }

                NoJob_tRS = NoJob_tRS + 1;

                }

        else;

}


//Cuando se termina de reparar la máquina, la plaza de repaaración
//vuelve a tomar el valor inicial

if(pn.getp_id()[pn.getCmin()[um[A]][0]]<0){
```

```
                        int q = pn.getCmin()[um[A]][0];
                        int repairTime = 100000000;
                        pn.changePT( repairTime, q, pn);
                }

                constructU(pn,um);
                constructUmp(pn,um, ump);
}

if (objective == 4)  {
                fitness = sumCjRS;
}
else if (objective == 5)  {

                fitness = sumWjCjRS;
}
else if (objective == 6)  {
                fitness = sumTjRS;
}
else if (objective == 7)  {
                fitness = sumWjTjRS;
}
else if (objective == 8)  {
                fitness = tnow;
}
else if (objective == 9)  {
                fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);
}

fitness1 = tnow;
fitness2 = sumWjTjRS;
fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);
```

```
}


void Individual::evalObjectiveMSLACK(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p, intvec& chrmDec,
intvec& chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, double& sumWjTj,
double& sumCj, double& sumTj, double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec&
jobTimes) {



        intvec Mf, Mp, Mr, p_id, Cmin, Cplus;

        intvec um;

        intvec ump;

        int A, delta, r;

        int remainingTime;

        int MinTransition;

        deque<int> u_activo;

        u_activo.clear();



        constructU(pn,um);

        constructUmp(pn,um, ump);

        double sumWjTjRS = sumWjTj;

        double sumCjRS = sumCj;

        double sumTjRS = sumTj;

        double sumWjCjRS = sumWjCj;

        int NoJob_tRS = NoJob_t;


        int Dj ;

        int Wj ;

        intvec CjRS;

        CjRS.clear();

        for(int i=0; i<Cj.size();i++){

                CjRS.push_back(Cj[i]);

        }
```

```
chrmCoded.clear();

chrmDec.clear();


for (int j = 0; j < chrm.size(); j++) {


        int temp = 0;

        int cont1 = 0;

        int posRepair;

        int pt;


        //Se encuentra la posición de la transición de rehabilitación de la máquina en
        //el vector um si aún está habilitada

        for(int i = 0; i < um.size(); i++){

                int pindx = pn.getCmin()[um[i]][0];

                pt = pn.getPT()[pn.getCplus()[um[i]][0]];

                if(pn.getp_id()[pn.getCmin()[um[i]][0]] < 0 &&
                pn.getPT()[pn.getCmin()[um[i]][0]] > 0){

                        cont1 = cont1 + 1;

                        posRepair=i;

                }

        }


                int NextMinTime;

                //Se encuentra la posición de la transición habilitada que inicia la
                //operación con míimo tiempo de procesamiento

                NextMinTime = findnextmintime2(um,pn.getPT(),pn, MinTransition);


                if(um.size() > 1 && cont1 > 0  && (tnow + NextMinTime >=
                mchBreakdownInstant + p)){

                        A = posRepair;

                }else{

                        A = MSLACK(um,pn.getPT(),pn,jobArray, pf_id, tnow, PTj,
                        jobTimes);

                }
```

```
//Se codifica la transición


int meme=0;

int number=0;

int cont3 = 0;


number = rand() % chrm.size();

meme = (number*(um.size())) + A;

chrm[j] = meme;


if(j == chrm.size()-1){

        if(pn.getp_id()[pn.getCmin()[um[0]][0]]<0){

                break;

        }

}


u_activo.push_back(um[A]);

chrmDec.push_back(um[A]);


pt = pn.getPT()[pn.getCplus()[um[A]][0]];

int pindx = pn.getCmin()[um[A]][0];

r = j;

pn.changeStateR(u_activo, r, delta);

tnow = tnow + delta;


if(pn.getp_id()[pn.getCmin()[um[A]][0]]>=0){

        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
        ].getNumber();

        jobTimes[jobNo] = jobTimes[jobNo] + pt;

        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {

                CjRS[NoJob_tRS] = tnow;

                Wj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
                ].getWeight();
```

```
                Dj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]
                ].getDue();


                if (objective == 4) {
                        sumCjRS = sumCjRS + Cj[NoJob_tRS];
                }
                else if (objective == 5) {
                        sumWjCjRS = sumWjCjRS + Wj*(CjRS[NoJob_tRS]);
                        }
                else if (objective == 6) {
                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)
                                sumTjRS = sumTjRS + (CjRS[NoJob_tRS] -
                                Dj);
                        }
                else if (objective == 7 || objective == 9) {
                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)
                                sumWjTjRS = sumWjTjRS + Wj *
                                (CjRS[NoJob_tRS] - Dj);
                        }
                NoJob_tRS = NoJob_tRS + 1;
                }
        else;
}


//Cuando se termina de reparar la máquina, la plaza de repaaración
//vuelve a tomar el valor inicial

if(pn.getp_id()[pn.getCmin()[um[A]][0]]<0){
        int q = pn.getCmin()[um[A]][0];
        int repairTime = 100000000;
        pn.changePT( repairTime, q, pn);
}


constructU(pn,um);
constructUmp(pn,um, ump);
```

```
        }


        if (objective == 4)  {
                        fitness = sumCjRS;
        }
        else if (objective == 5)  {


                        fitness = sumWjCjRS;
        }
        else if (objective == 6)  {
                        fitness = sumTjRS;
        }
        else if (objective == 7)  {
                        fitness = sumWjTjRS;
        }
        else if (objective == 8)  {
                        fitness = tnow;
        }
        else if (objective == 9)  {
                        fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);
        }


        fitness1 = tnow;
        fitness2 = sumWjTjRS;
        fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);
}



void Individual::evalObjective(PetriNet pn, int tnow, intvec& pf_id, int ta, const int&
objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, int p, intvec& chrmDec,
intvec& chrmCoded, double& PrMakespan, double& PrtotalWeightedTardiness, deque<int>& ur,
deque<int>& td, double& sumWjTj, double& sumCj, double& sumTj, double& sumWjCj, int& NoJob_t,
intvec& Cj) {
```

```
intvec Mf, Mp, Mr, p_id, Cmin, Cplus;
intvec um, ump;
int A, delta, r;
int MinTransition;
deque<int> u_activo;
u_activo.clear();

intvec shootTimes;
shootTimes.clear();
chrmDec.clear();

constructU(pn,um);
constructUmp(pn, um, ump);
double sumWjTjRS = sumWjTj;
double sumCjRS = sumCj;
double sumTjRS = sumTj;
double sumWjCjRS = sumWjCj;
int NoJob_tRS = NoJob_t;
int Dj ;
int Wj ;
intvec CjRS;
CjRS.clear();
for(int i=0; i<Cj.size();i++){
        CjRS.push_back(Cj[i]);
}
int remainingTime;
int temp = 0;
int n = 0;
int pt;

for (int j = 0; j < chrm.size(); j++) {
        int temp = 0;
```

```
int minMr = 1000000;

int cont1 = 0;

int meme;

int number;

number = rand() % chrm.size();



int posRepair;

intvec inputTransitions;

intvec outputTransitions;


for(int i = 0; i < um.size(); i++){

                int pindx = pn.getCmin()[um[i]][0];

                if(pn.getp_id()[pn.getCmin()[um[i]][0]] < 0 &&
                pn.getPT()[pn.getCmin()[um[i]][0]] > 0){

                        cont1 = cont1 + 1;

                        posRepair=i;

                }


                //Se guardan dos vectores, uno con las transiciones que activan
                //las operaciones (Entradas) y y el otro con las transiciones
                //que las finalizan (Salidas)

                if(pn.getCplus()[um[i]].size() > 1 &&
                pn.getPT()[pn.getCplus()[um[i]][0]] == 0){

                        outputTransitions.push_back(um[i]);

                }else{

                        inputTransitions.push_back(um[i]);

                }

}


A = chrm[j] % um.size();


int tempA;

int minimalRestingTime=100000000;

int cnt = 0;
```

```
if(outputTransitions.size()>0){
        //Si hay transiciones de salida
        for(int i= 0; i<outputTransitions.size(); i++){
        //Se encuentra la posición correspondiente en el vector
                if(outputTransitions[i] == um[A]){
                        for(int m= 0; m<outputTransitions.size(); m++){
                                int cnt1 = 0;
                                for(int k=0; k<u_activo.size(); k++){
                                        if((outputTransitions[m]-1) ==
                                        u_activo[k]){
                                        //Si la transición es de salida, se busca
                                        //si la transición de entrada de la
                                        //operación ha sido disparada en la
                                        //reprogramación y se calcula el tiempo
                                        //de producción restante
                                                pt = pn.getPT()[pn.getCplus()
                                                [u_activo[k]][0]];
                                                int NextTime = (pt +
                                                shootTimes[k]) - tnow;
                                                if(NextTime <
                                                minimalRestingTime){
                                                        tempA = m;
                                                        minimalRestingTime =
                                                        NextTime;
                                                        cnt = cnt +1;
                                                        cnt1 = cnt1 +1;
                                                }
                                        }
                                }
                                if(cnt1 == 0){
                                        for(int k=0; k<ur.size(); k++){
                                                if((outputTransitions[m]-1) ==
                                                ur[k]){
                                                //Si la transición es de salida,
                                                //se busca si la transición de
                                                //entrada de la operación ha sido
                                                //disparada antes de la falla y
                                                //se calcula el tiempo de
                                                //producción restante
```

```
                                                pt = pn.getPT()
                                                [pn.getCplus()[ur[k]][0]];

                                                int NextTime = (pt +
                                                td[k]) - tnow;

                                                if(NextTime <
                                                minimalRestingTime){

                                                        tempA = m;

                                                        minimalRestingTime
                                                        = NextTime;

                                                        cnt = cnt +1;

                                                }

                                        }

                                }

                        }

                }

        }

}


if(cnt > 0){//Si la transición de entrada ha sido disparada

                for(int i=0; i<um.size(); i++){

                        if(um[i] == outputTransitions[tempA]){

                                A = i;

                                meme = (number*(um.size())) + A;

                                chrm[j] = meme;

                        }

                }


                if(cont1>0){

                //Si la reparación de la máquina que presentó la falla
                //finalizará antes del tiempo restante de la operación
                //de la transición de salida que se disparará, se
                //dispara la transición que la habilita nuevamente

                        if(tnow + minimalRestingTime >=
                        mchBreakdownInstant + p){

                                A = posRepair;

                                meme = (number*(um.size())) + A;

                                chrm[j] = meme;
```

```
                        }
                    }
        }else{
                    if(cont1>0){
                    //Si ha finalizado la reparación de la máquina que
                    //presentó la falla, se dispara la transición que la
                    //habilita nuevamente
                        if(tnow >= mchBreakdownInstant + p){
                            A = posRepair;
                            meme = (number*(um.size())) + A;
                            chrm[j] = meme;
                        }
                    }

        }

}




if(j == chrm.size()-1){
        if(pn.getp_id()[pn.getCmin()[um[0]][0]]<0){
            break;
        }
}

u_activo.push_back(um[A]);
chrmDec.push_back(um[A]);

int pt = pn.getPT()[pn.getCplus()[um[A]][0]];
int pindx = pn.getCmin()[um[A]][0];

r = j;
pn.changeStateR(u_activo, r, delta);
```

```
tnow = tnow + delta;
shootTimes.push_back(tnow);

if(pn.getp_id()[pn.getCmin()[um[A]][0]]>=0){
        int jobNo = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]].getNumber();

        if (pn.getCplus()[u_activo[j]][0] == pf_id[jobNo]) {
                CjRS[NoJob_tRS] = tnow;
                Wj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]].getWeight();
                Dj = jobArray[pn.getp_id()[pn.getCmin()[um[A]][0]]].getDue();
                if (objective == 4) {
                        sumCjRS = sumCjRS + CjRS[NoJob_tRS];
                }
                else if (objective == 5) {
                        sumWjCjRS = sumWjCjRS + Wj*(CjRS[NoJob_tRS]);
                        }
                else if (objective == 6) {
                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)
                                sumTjRS = sumTjRS + (CjRS[NoJob_tRS] - Dj);
                        }
                else if (objective == 7 || objective == 9) {
                        if ( (CjRS[NoJob_tRS] - Dj) >= 0)
                                sumWjTjRS = sumWjTjRS + Wj * (CjRS[NoJob_tRS] -
                                Dj);
                        }
                NoJob_tRS = NoJob_tRS + 1;
                }
        else;
}

if(pn.getp_id()[pn.getCmin()[um[A]][0]]<0){
        int q = pn.getCmin()[um[A]][0];
        int repairTime = 100000000;
        pn.changePT( repairTime, q, pn);
```

```
                }


                constructU(pn,um);
                constructUmp(pn,um, ump);
        }


        if (objective == 4)  {
                        fitness = sumCjRS;
        }
        else if (objective == 5)  {
                        fitness = sumWjCjRS;
        }
        else if (objective == 6)  {
                        fitness = sumTjRS;
        }
        else if (objective == 7)  {
                        fitness = sumWjTjRS;
        }
        else if (objective == 8)  {
                        fitness = tnow;
        }
        else if (objective == 9)  {
                        fitness = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);

        }


        fitness1 = tnow;
        fitness2 = sumWjTjRS;
        fitness3 = (PrMakespan*tnow) + (PrtotalWeightedTardiness*sumWjTjRS);

}
```

```
void Individual::generateScheduleIS(PetriNet pn, const vector<Tjob>& jobArray, int tnow,
intvec& pf_id, int& NoJob_t, vec2dim& sched, deque<int>& u, intvec& tr, intvec& winnerDec,
vec2dim& transitions, intvec& Cj, double& PrMakespan, double& PrtotalWeightedTardiness, int&
TTPAGIS) {



	sched.clear();

	sched.resize(1);

	copy (chrm.begin(), chrm.end(), back_inserter(sched[0]));


	tr.clear();


	intvec um, ump;

	deque<int> u_activo;

	u_activo.clear();

	u.clear();

	constructU(pn,um);

	constructUmp(pn, um, ump);

	int A, r, delta;

	int NextMinTime;

	int cont1 = 0;

	int index, pt;

	int oper, machine, numJob;

	int NoJob_tIS=NoJob_t;

	int sumWjTj = 0;

	intvec CjIS;

	CjIS.clear();

	for(int i=0; i<Cj.size();i++){

		CjIS.push_back(Cj[i]);

	}


	for (int i = 0; i < sched.size(); i++) {


		for (int j=0; j< sched[i].size(); j++) {
```

```
A = sched[0][j] % ump.size();
u_activo.push_back(ump[A]);
u.push_back(ump[A]);
int tran = ump[A];
r = j;
pn.changeStateR(u_activo, r, delta);
tnow = tnow + delta;

tr.push_back(tnow);

numJob = jobNo(pn,ump[A]);

        index = pn.getCplus()[ump[A]][0];
        pt = pn.getPT()[index];

        if(pt>0){
                machine = allMchNo(pn,ump[A]);
                jobOperationIndex(ump, transitions, oper, A, numJob);
        }else if(pt == 0){
                machine = 0;
                if(pn.getp_id()[pn.getCmin()[ump[A]][0]]<0){
                        oper = -1;
                        numJob = -1;
                }else{
                        jobOperationIndex(ump, transitions, oper, A,
numJob);
                }
        }

        int Dj ;//Fecha de entrega del trabajo j
        int Wj ;//Peso del trabajo j
```

```
                                if(pn.getp_id()[pn.getCmin()[ump[A]][0]]>=0){
                                        int jobNo =
jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]].getNumber();


                                        if (pn.getCplus()[ump[A]][0] == pf_id[jobNo]) {
                                                CjIS[NoJob_tIS] = tnow;
                                                Wj =
jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]].getWeight();
                                                Dj =
jobArray[pn.getp_id()[pn.getCmin()[ump[A]][0]]].getDue();
                                                if ( (CjIS[NoJob_tIS] - Dj) >= 0) {
                                                        sumWjTj = sumWjTj + Wj * (CjIS[NoJob_tIS]
- Dj);
                                                }
                                        NoJob_tIS = NoJob_tIS + 1;
                                        }
                                }


                                TTPAGIS = sumWjTj;


                        constructU(pn,um);
                        constructUmp(pn, um, ump);
                }
        }


}




void Individual::generateSchedule(PetriNet pn, const vector<Tjob>& jobArray, int& tnow,
intvec& pf_id, int& NoJob_t, vec2dim& sched, deque<int>& u, intvec& tr, intvec& jobsR, intvec&
operationsR, intvec& machinesR, intvec& ptR, intvec& winnerDec, vec2dim& transitions, intvec&
Cj, int sumWjTj, double& PrMakespan, double& PrtotalWeightedTardiness, double& TTPRS) {


                sched.clear();
        sched.resize(1);
```

```
copy (chrm.begin(), chrm.end(), back_inserter(sched[0]));

        tr.clear();

        intvec um, ump;
        deque<int> u_activo;
        u_activo.clear();
        u.clear();
        constructU(pn,um);
        constructUmp(pn, um, ump);
        int A, r, delta;
        int NextMinTime;
        int cont1 = 0;
        int oper, machine, numJob;
        int index, pt;
        int NoJob_tRS=NoJob_t;

        intvec CjRS;
        CjRS.clear();
        for(int i=0; i<Cj.size();i++){
                CjRS.push_back(Cj[i]);
        }

        for (int i = 0; i < sched.size(); i++) {
                for (int j=0; j< sched[i].size(); j++) {
                        if(j == sched[i].size()-1){
                                if(pn.getp_id()[pn.getCmin()[um[0]][0]]<0){
                                        break;
                                }
                        }

                        A = sched[0][j] % um.size();
                        u_activo.push_back(um[A]);
```

```
u.push_back(um[A]);
r = j;
pn.changeStateR(u_activo, r, delta);
tnow = tnow + delta;

tr.push_back(tnow);

numJob = jobNo(pn,um[A]);

index = pn.getCplus()[um[A]][0];
pt = pn.getPT()[index];

if(pt>0){
        machine = allMchNo(pn,um[A]);
        machinesR.push_back(machine);
        jobOperationIndex(um, transitions, oper, A, numJob);
}else if(pt == 0){
        machinesR.push_back(0);
        machine = 0;
        if(pn.getp_id()[pn.getCmin()[um[A]][0]]<0){
                oper = -1;
                numJob = -1;
        }else{
                jobOperationIndex(um, transitions, oper, A,
numJob);

        }
}

jobsR.push_back(numJob+1);
operationsR.push_back(oper+1);
ptR.push_back(pt);

int Dj ;//Fecha de entrega del trabajo j
int Wj ;//Peso del trabajo j
```

```
                        if(pn.getp_id()[pn.getCmin()[um[A]][0]]>=0){
                                int jobNo = jobArray[pn.getp_id()[pn.getCmin()[um[A]]
                                [0]]].getNumber();
                                if (pn.getCplus()[um[A]][0] == pf_id[jobNo]) {
                                        CjRS[NoJob_tRS] = tnow;
                                        Wj = jobArray[pn.getp_id()[pn.getCmin()[um[A]]
                                        [0]]].getWeight();
                                        Dj = jobArray[pn.getp_id()[pn.getCmin()[um[A]]
                                        [0]]].getDue();
                                        if ( (CjRS[NoJob_tRS] - Dj) >= 0) {
                                                sumWjTj = sumWjTj + Wj * (CjRS[NoJob_tRS]
                                                - Dj);
                                        }
                                NoJob_tRS = NoJob_tRS + 1;
                                }

                        }

                        TTPRS = sumWjTj;
                        constructU(pn,um);

                }
        }
}
```

Machine.h: interface for the Machine class.

```
class Machine
{
```

```
friend class WkCenter;
friend class Schedule;

friend bool operator < (const Machine&, const Machine&);

public:
        Machine();
        virtual ~Machine();

        //Set methods
        void setMchId(const char*);
        void setRelease(const int);
        void setStatus(const char*);

        // Get methods
        const char* getMchId() const;
        int getRelease() const;
        const char* getStatus() const;

private:
        char mchId[80];  //Identifies the machine with a string of characters
        int release;   // Availability date of the machine
        char status[2]; //Initial status of the machine. Must be a letter (A, B, C, etc)
                                        //See Lekin documentation for definition of status

};
```

---------- Machine.cpp: implementation of the Machine class. ----------

```
#include "Machine.h"
```

```
#include <stddef.h>
#include <string.h>

////////////////////////////////////////////////////////////////////////
// Construction/Destruction
////////////////////////////////////////////////////////////////////////

bool operator < (const Machine& m1, const Machine& m2){

        return m1.release < m2.release;
}

Machine::Machine()
{
}

Machine::~Machine()
{
}

void Machine::setMchId(const char* idn){
        strcpy(mchId,idn);
}

void Machine::setRelease(const int rls)
{ release = rls; }


void Machine::setStatus(const char* stts)
        {strcpy(status,stts); }

const char* Machine::getMchId() const {
    return mchId;
```

```
}

int Machine::getRelease() const {
    return release;
}

const char* Machine::getStatus() const {
    return status;

}
```

---

matrix.h: interface for the class matrix.

```
#ifndef MATRIX_H
#define MATRIX_H

#include <functional>

#include <algorithm>
#include <cassert>
#include <iterator>
#include <vector>
using namespace std;


#include <math.h>
#include <iostream>
#include <fstream>

typedef vector<int> intvec;
```

```
typedef vector<intvec> vec2dim;


class matrix{


////////////////////////////////////////////////////////////////

//Friend functions

////////////////////////////////////////////////////////////////


friend ostream& operator << (ostream&, const matrix&);

friend void extract_col (const matrix&, matrix&, const int&); //extracts a column from a
matrix

friend void mmult (const matrix&, const matrix&, matrix&, bool&);

friend void trans (const matrix&, matrix&); // transposes a matrix

friend void msum (const matrix&, const matrix&, matrix&, bool&); //sums two matrices

friend void stackmatrix(matrix&, matrix&, matrix&); //stacks matrix b under matrix a. result
stored in c

friend void adjoinmatrix(matrix&, matrix&, matrix&);//adjoins matrix b to the right of matrix
a. result stored in c

friend void insert_row(matrix&, matrix&, const int); //inserts row in position numrow

friend void insert_col(matrix&, matrix&, const int); //inserts col in position numcol*/

friend bool operator == (const matrix&, const matrix&); //determines if two matrices are
equal.


////////////////////////////////////////////////////////////////

//Public functions and members

////////////////////////////////////////////////////////////////


public:


        matrix();                                   //default constructor

        ~matrix();                                  //destructor



        matrix (int, int);                  //creates a matrix with n rows and m columns

        matrix (int);                       //creates an array of n elements
```

```
        int getcols () const;         //gets number of columns
    int getrows () const;             //gets number of rows
        vec2dim getmat() const;
        int mmax ();                          //gets the max of the matrix
        int mmin ();                          //gets the min of the matrix

        void readMatrix (istream&);           //reads a matrix from a file
        void readMatrix ();                           //reads from screen
        void readMatrix (istream&, const int&, const int&);  //reads a matrix from a file with
fixed number of cols and rows
        void readMatrix (istream&, const int&);
        void clearmatrix ();          //clears a matrix
        void destroymatrix();      //uses the destructor ~vector
        void resize_matrix(const int&, const int&);  // resizes a matrix
        void resize_sqmatrix (const int& n, const int& m);   // resizes a matrix to be square
with the maximum of the two dimensions n or m

        matrix& erase_row(const int num);             //erases the num-th row
        matrix& erase_col(const int num);             //erases the num-th column
        matrix& operator = (const matrix&);           // the assignment operator

        const int& operator () (const int i, const int j) const;//returns position i,j of the
matrix r-value
        int& operator() (int i, int j); //l-value for assingment;
        matrix operator+ (const matrix&);
        matrix operator- (const matrix&);
        matrix operator* (const matrix&); // multiplies two matrices

private:
        int rows, cols;
        vec2dim mat;


};
```

```
#endif
```

---

matrix.cpp: implementation of the matrix class.

```
#include "matrix.h"
//#include <vector>

//#include <functional>

//*********************************************************************
matrix::matrix () {
//default constructor
        cols =0;
        rows=0;
        mat.resize(0);
}
//*********************************************************************
matrix::matrix (int numrows, int numcols)
//constructor that allocates numrows*numcols spaces and initalizes matrix


{ assert(numrows>0 && numcols>0);
  rows = numrows;
  cols = numcols;

  mat.resize(numrows);

        for (int i=0; i< numrows; i++)
```

```
        {       mat[i].resize(numcols);}
}

//*********************************************************************
matrix::matrix (int numelements)
//constructor that allocates numelements spaces and initalizes a 1-dimensional array

{       assert(numelements >=0);
        rows = numelements;
        cols = 1;
        mat.resize(numelements);

        for (int i=0; i< numelements; i++)

        {mat[i].resize(1);}
}

//*********************************************************************
matrix::~matrix()
{mat.~vector();}

//*********************************************************************
int matrix::getcols() const
// returns number of columns
{ return cols; }


//*********************************************************************
int matrix::getrows() const
// returns number of rows
{ return rows; }
```

```
vec2dim matrix::getmat() const
//returns mat

{return mat;}



//*********************************************************************
void matrix::readMatrix ()
// void that reads from screen a matrix of integers
{

        vec2dim::iterator matPtr=mat.begin();
        intvec::iterator vecPtr = mat[0].begin();

        //alternative form
        //cout << "Enter matrix elements by rows " << endl;
        istream_iterator <int> inputint(cin);

        int i=0;

        for (matPtr = mat.begin(); matPtr != mat.end();matPtr++)

                for(vecPtr = mat[i].begin() ; vecPtr <= mat[i].end() ; vecPtr++)
        {
                *vecPtr = *inputint;

                if (vecPtr <mat[i].end() - 1)
                        inputint++;
                i++;
        }
}
```

```
//*********************************************************************
void matrix::readMatrix (istream& infile)
// void that reads from 'is' a matrix of integers.Reads all elements of the file
{
        mat.clear();

        //vec2dim::iterator matPtr= mat.begin();
        //intvec::iterator vecPtr = mat[0].begin();

        int i=0;
        istream_iterator <int> inputint(infile);

        while (!infile.eof()) {
                copy (istream_iterator<int> (infile), istream_iterator<int> (),
back_inserter(mat[i]));
                i++;
        }

        //alternative form

        /*
        mat.push_back(*inputint);

        while (!infile.eof()) {
                ++ inputint;
                mat.push_back(*inputint);

        }*/
}

//*********************************************************************

void matrix::readMatrix (istream& infile, const int& row, const int& col)
// void that reads from 'is' a matrix of integers
```

```
{
        mat.clear();
        exception e;

        try {

        rows = row;
        cols = col;
        if (rows <= 0 || cols <= 0)

                throw e;

        mat.resize(rows);
        for (int i=0; i< rows;i++) {

                mat[i].resize(cols);

                for (int j=0; j< cols;j++) {

                        infile >> mat[i][j];

                        if (mat[i][j] != 0 && mat[i][j] != 1 && mat[i][j] != -1)

                                throw e;

                }
        }

        }
        catch (exception e) {

                //cout << "Invalid Incidence Matrix" << endl;
                system("PAUSE");
```

```
                exit(1);
        }

}

//*************************************************************************
void matrix::readMatrix (istream& infile, const int& row)
// void that reads from 'is' a matrix of integers
{
        mat.clear();

        rows = row;
        cols = 1;
        for (int i=0; i< rows;i++) {

                copy (istream_iterator<int> (infile), istream_iterator<int> (),
back_inserter(mat[i]));
                mat[i].resize(cols);

        }

        //alternative form*/

        /*istream_iterator_int inputint(infile);
        mat.push_back(*inputint);
        i= 1;

        while (i<rows) {
                ++ inputint;
                mat.push_back(*inputint);
                ++i;
        }*/
}
```

```
//*********************************************************************
void matrix::destroymatrix()
{mat.~vector();
cols=0;
rows=0;}

//*********************************************************************
void matrix::clearmatrix()
{mat.clear();}

//*********************************************************************
void matrix::resize_matrix (const int& n, const int& m) {

        rows = n;
        cols = m;
        mat.resize(n);

        for (int i=0; i<(n); i++){
                mat[i].resize(m);
        }

}

void matrix::resize_sqmatrix (const int& n, const int& m){

        if (n > rows || m > cols) {

                if (n >= m)
                        resize_matrix(n,n);
                else

                        resize_matrix(m,m);
        }
```

```
}

//*********************************************************************
int matrix::mmax(){;

int i,j,max=mat[0][0];

        for (i=0; i<rows;i++) {
                for (j=0; j<cols;j++) {
                        if (max < mat[i][j])
                                max= mat[i][j];
                }
        }
        return max;
}

//*********************************************************************
int matrix::mmin(){;

int i,j,min=mat[0][0];

        for (i=0; i<rows;i++) {
                for (j=0; j<cols;j++) {
                        if (min < mat[i][j])
                                min= mat[i][j];
                }
        }
        return min;
}

matrix& matrix::operator= (const matrix& a) {

        rows=a.rows;
```

```
        cols=a.cols;
        mat= a.mat;
        return *this;
}

matrix& matrix::erase_row(const int num) {

        assert (num <= rows);
        for (int j=0;j<cols;j++)
                mat.erase(mat.begin() + num*cols);
        rows--;

        return *this;
}

matrix& matrix::erase_col(const int num) {

        assert (num <= cols);
        for (int i=0; i<rows; i++)
                mat.erase(mat.begin() + i*cols + num - i);
        cols--;
        return *this;
}

matrix matrix::operator + (const matrix &m1) {

        int i,j;
        assert(cols==m1.cols);
        assert(rows==m1.rows);
        matrix m2(rows,cols);

        for (i=0; i<m1.rows;i++)      {
```

```
                for (j=0; j<m1.cols;j++) {
                        m2(i,j)= mat[i][j] + m1(i,j);}
        }

        return m2;
}

//*********************************************************************
matrix matrix::operator - (const matrix &m1) {

        int i,j;
        assert(cols==m1.cols);
        assert(rows==m1.rows);
        matrix m2(rows,cols);



        for (i=0; i<m1.rows;i++)      {
                for (j=0; j<m1.cols;j++) {
                        m2(i,j)= mat[i][j] - m1(i,j);}
        }

        return m2;
}

//*********************************************************************

matrix matrix::operator* (const matrix &m1) {

                int val,i,j,k;
                matrix m3;

                assert (rows == m1.cols);
```

```
                m3.resize_matrix(m1.rows,m1.cols);


                for (i=0; i<m3.rows;i++)      {
                        for (j=0; j<m3.cols;j++) {
                                val = 0;
                                for (k = 0; k < m1.cols; k++)
                                        val += mat[i][k]* m1(k,j);
                                m3.mat[i][j]= val;
                        } //for j
                } //for i



                return m3;
        }

//*****************************************************************

//FRIEND FUNCTIONS



// return_type  context :: operator/function

const int& matrix::operator () (const int i, const int j) const {
//returns position i,j of the matrix which is in position (i, j) in a linear array

        return mat[i][j];
} //returns the reference const

int& matrix::operator () (int i, int j)  {
//returns position i,j of the matrix which is in position (i, j)

        return mat[i][j];} //returns the reference
```

```
//*********************************************************************
ostream& operator<<(ostream& os, const matrix& v)
// operator <<  outputs a matrix of integers
 {
   /*matrix m(v.getrows(),v.getcols());
   m = v;
           = v.getmat(); */
   for (int i=0;i<v.getrows();i++)     {
           for (int j=0;j<v.getcols();j++) {
                os << v(i,j) << " ";
                                   }
           os << endl;
   }

   return os;
 }
//*********************************************************************




void extract_col (const matrix& matr, matrix& column, const int& col_number) {

//extracts the col_number-th column from matrix matr.


        //assert conditions must be true for the program to go on
        assert (column.rows== matr.rows || column.cols==1 || col_number<=matr.cols);

                for (int i=0; i<matr.rows; i++)
                        column(i,0) = matr(i,col_number);
}

//*********************************************************************
```

```
void mmult (const matrix& m1, const matrix& m2, matrix& m3, bool& error) {
        int i,j,k,val;
        error = false;

        if (m1.cols != m2.rows)
        {//cout << "Error: Dimensions do not agree " << endl;
        error = true;}

        else
        {
                m3.resize_matrix(m1.rows,m2.cols);

                for (i=0; i<m3.rows;i++)      {
                        for (j=0; j<m3.cols;j++) {
                                val = 0;
                                for (k = 0; k < m1.cols; k++)
                                        val += m1(i,k)* m2(k,j);
                                m3.mat[i][j]= val;
                        } //for j
                } //for i

        }//else

}

//*****************************************************************
void msum (const matrix& m1, const matrix& m2, matrix& m3, bool& error) {
int i,j;

        if (m1.rows != m2.rows || m1.cols != m2.cols)
                {//cout << "error dimensions do not agree " << endl;
                error = true;}
```

```
        else
        {
                m3.resize_matrix(m1.rows,m1.cols);

                for (i=0; i<m1.rows;i++)      {
                        for (j=0; j<m1.cols;j++) {
                                m3(i,j) = m1(i,j)+ m2(i,j);}
                }
        }
}

void trans(const matrix& m1, matrix& m2) {
        int i,j;

        m2.resize_matrix(m1.rows,m1.cols);

        for (i=0; i<m1.cols;i++) {
                for (j=0; j<m1.rows;j++) {
                        m2.mat[i][j] = m1(j,i);
                }
        }
}

bool operator == (const matrix& mat1, const matrix& mat2) {

        if (mat1.getcols()!=mat2.getcols() || mat1.getrows()!=mat2.getrows() ||
mat1.getmat()!= mat2.getmat())
        return false;

        else
        return true;

}
```

```
void stackmatrix(matrix& a, matrix& b, matrix& c) {

        //stacks matrix b below matrix a and the result goes to c.
        if (a.mat.empty()) {

                /*c.mat=b.mat;
                c.cols=b.cols;
                c.rows=b.rows;*/
                c = b;
        }

        else {

                assert (a.cols==b.cols);
                c.mat= a.mat;
        //inserting rows

                c.mat.insert(c.mat.end(),b.mat.begin(), b.mat.end());
                /*for (int i=0;i<b.rows*b.cols;i++) {

                        c.mat.push_back(b.mat[i]);

                }*/

                c.rows=a.rows+b.rows;
                c.cols = a.cols;
        }

}

void adjoinmatrix(matrix& a, matrix& b , matrix& c) {
```

```
        //adjoins matrices a and b. Result goes to c;
        //a and b are the left and right matrices respectively
        if (a.mat.empty()) {

                c = b;
        }

        else {

                assert (a.rows==b.rows);
                //matrix d=b;
                c = a;

                for (int i=0;i<c.rows;i++) {

                        for (int j=0; j<b.cols;j++)

                                {c.mat[i].push_back(b(i,j));}
                }
                c.cols=a.cols+b.cols;
        }
}

void insert_row(matrix& a, matrix& b, const int numrow) {

        //inserts a row vector b in matrix a in position numrow
        if (a.mat.empty() && b.cols > 0 && b.rows==1) {

                a.mat=b.mat;
                a.cols=b.cols;
                a.rows=1;
        }
```

```
        else {

                assert (a.cols ==  b.cols && b.rows == 1 && numrow <= a.rows);

        //inserting rows
                intvec bb;
                a.mat.insert(a.mat.begin()+numrow,b.mat[0]);
                a.rows++;
        }

}

void insert_col(matrix& a, matrix& b , const int numcol) {
        //inserts a column matrix b in matrix a in position numcol

        if (a.mat.empty() && b.rows > 0 && b.cols==1) {

                a = b;
        }

        else {

                assert (a.rows==b.rows && b.cols == 1 && numcol <= a.cols);

                for (int i=0;i<a.rows;i++) {

                        a.mat[i].insert(a.mat[i].begin()+numcol,b(i,0));

                }

                a.cols++;
        }
}
```

```
void init(intvec& v1, const int& num) {
        for (int i=0; i<v1.size(); i++)
                v1[i]= num;
}
```

---

memetic.h: interface for the memetic class.

---

memetic.cpp: implementation of the memetic class.

```
#include "Memetic.h"
#include "Tjob.h"
#include "WkCenter.h"
#include "PetriNet.h"
#include "Schedule.h"
#include "Math.h"
#include <fstream>


//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

Memetic::Memetic() {

        prMutation = 0.0;
```

```
	prCrossover = 0.0;
		mutateIntensity = 1;
		elitism = false;
	parentSelection = 0; //ruleta, torneo
	crossoverType  = false; //1 punto, 2 puntos
	chrSelection = 0; // mu + lambda, mu, lambda
	maxGenerations = 0; //max number of iterations
	populationSize = 0;
		percent = 0;
}

Memetic::Memetic(double prMut , double prCross , int mutInt, int maxGen , bool selType , bool
crossType , int gama,bool applyLS , bool lSearch, bool lSType , bool iRZt) {

	prMutation = prMut;
	prCrossover = prCross;
	mutateIntensity = mutInt;
	elitism = selType;
	crossoverType = crossType;
	applyLocalSearch = applyLS;
	localSearch = lSearch;
	localSearchType1 = lSType;
	iRZtype = iRZt;
	Gamma = gama;
	parentSelection = 0;
	chrSelection = 0;
	maxGenerations = maxGen;
	populationSize = 0;
	percent = 0;
}

Memetic::~Memetic()
{
```

```
}

void Memetic::initPopulation(PetriNet pn, int tnow, intvec& pf_id, intvec& allFitnessGA, int
ta, const int& psize, const int& chrmSize, const int& objective, const vector<Tjob>& jobArray,
int mchBreakdownInstant, intvec& um, int p, double& PrMakespan, double&
PrtotalWeightedTardiness, deque<int>& ur, deque<int>& td, double& sumWjTj, double& sumCj,
double& sumTj, double& sumWjCj, int& NoJob_t, intvec& Cj, intvec& PTj, intvec& jobTimes) {


        population.clear();

        population.resize(psize);

        populationSize = psize;


               decodedPopulation.clear();

        decodedPopulation.resize(psize);


               int tejeci, tejecf, tejec;

               int cont1 = 0;

               allFitnessGA.clear();

               tejeci = time(0);


        intvec tempChrm(chrmSize);


            for (int j = 0; j < chrmSize; j++) {

                 tempChrm [j] = j;

        }


        intvec dummyChrm = tempChrm;


        for (int i = 0; i < psize ; i++) {

                population[i].chrm.resize(chrmSize);

                              for (int j = 0; j < chrmSize; j++) {

                                            int pos = rand() % dummyChrm.size();


                        population[i].chrm[j] = dummyChrm[pos];


                                            dummyChrm.erase(dummyChrm.begin() + pos);

                }
```

```
deque<int> uRS;

vec2dim arrivals;

vec2dim transitions;//nuevo

vec2dim operEndTime;

vector<intvec> marking;

intvec timeU;

intvec chrmCoded;

intvec chrmDec;


if(i > 2 && cont1==0 ){

        cont1 = cont1 +1;

}


uRS.clear();


if (i==0) {


        population[i].evalObjectiveSPT(pn, tnow, pf_id, ta,
        objective,jobArray, mchBreakdownInstant, p, chrmDec,
        chrmCoded, PrMakespan, PrtotalWeightedTardiness,
        sumWjTj, sumCj, sumTj, sumWjCj, NoJob_t, Cj);

        decodedPopulation[i] = chrmDec;


        allFitnessGA.push_back(population[i].fitness);


        dummyChrm = tempChrm;

        tejecf = time(0);

        tejec = tejecf - tejeci;

        }

else if (i==1) {

        population[i].evalObjectiveLPT(pn, tnow, pf_id, ta,
        objective,jobArray, mchBreakdownInstant, p, chrmDec,
        chrmCoded, PrMakespan, PrtotalWeightedTardiness,
        sumWjTj, sumCj, sumTj, sumWjCj, NoJob_t, Cj);

        decodedPopulation[i] = chrmDec;
```

```
        dummyChrm = tempChrm;

        allFitnessGA.push_back(population[i].fitness);

        tejecf = time(0);

        tejec = tejecf - tejeci;


        }
    else if (i==2) {

        population[i].evalObjectiveMRWT(pn, tnow, pf_id, ta,
        objective,jobArray, mchBreakdownInstant, p, chrmDec,
        chrmCoded, PrMakespan, PrtotalWeightedTardiness,
        sumWjTj, sumCj, sumTj, sumWjCj, NoJob_t, Cj);

        decodedPopulation[i] = chrmDec;

        allFitnessGA.push_back(population[i].fitness);

        dummyChrm = tempChrm;

        tejecf = time(0);

        tejec = tejecf - tejeci;

        }
    else if (i==3) {

        population[i].evalObjectiveATC(pn, tnow, pf_id, ta,
        objective,jobArray, mchBreakdownInstant, p, chrmDec,
        chrmCoded, PrMakespan, PrtotalWeightedTardiness,
        sumWjTj, sumCj, sumTj, sumWjCj, NoJob_t, Cj, PTj,
        jobTimes);

        population[i].chrmDec.resize(chrmDec.size());

        decodedPopulation[i] = chrmDec;

        allFitnessGA.push_back(population[i].fitness);

        dummyChrm = tempChrm;

    }else if (i==4) {

        population[i].evalObjectiveCR(pn, tnow, pf_id, ta,
        objective,jobArray, mchBreakdownInstant, p, chrmDec,
        chrmCoded, PrMakespan, PrtotalWeightedTardiness,
        sumWjTj, sumCj, sumTj, sumWjCj, NoJob_t, Cj, PTj,
        jobTimes);

        population[i].chrmDec.resize(chrmDec.size());

        decodedPopulation[i] = chrmDec;

        allFitnessGA.push_back(population[i].fitness);

        dummyChrm = tempChrm;

    }else if (i==5) {
```

```
                                        population[i].evalObjectiveEDD(pn, tnow, pf_id, ta,
                                        objective,jobArray, mchBreakdownInstant, p, chrmDec,
                                        chrmCoded, PrMakespan, PrtotalWeightedTardiness,
                                        sumWjTj, sumCj, sumTj, sumWjCj, NoJob_t, Cj, PTj,
                                        jobTimes);

                                        population[i].chrmDec.resize(chrmDec.size());

                                        decodedPopulation[i] = chrmDec;

                                        allFitnessGA.push_back(population[i].fitness);

                                         dummyChrm = tempChrm;

                                    }else if (i==6) {

                                        population[i].evalObjectiveMSLACK(pn, tnow, pf_id, ta,
                                        objective,jobArray, mchBreakdownInstant, p, chrmDec,
                                        chrmCoded, PrMakespan, PrtotalWeightedTardiness,
                                        sumWjTj, sumCj, sumTj, sumWjCj, NoJob_t, Cj, PTj,
                                        jobTimes);

                                        population[i].chrmDec.resize(chrmDec.size());

                                        decodedPopulation[i] = chrmDec;

                                        allFitnessGA.push_back(population[i].fitness);

                                        dummyChrm = tempChrm;

                                    }else {

                                        population[i].evalObjective(pn, tnow, pf_id, ta,
                                        objective,jobArray, mchBreakdownInstant, p, chrmDec,
                                        chrmCoded, PrMakespan, PrtotalWeightedTardiness, ur, td,
                                        sumWjTj, sumCj, sumTj, sumWjCj, NoJob_t, Cj);

                                        population[i].chrmDec.resize(chrmDec.size());

                                        decodedPopulation[i] = chrmDec;

                                        allFitnessGA.push_back(population[i].fitness);

                                        dummyChrm = tempChrm;



                                    }

                }


        }


void Memetic::initPopulationIS(PetriNet pn, int tnow, intvec& pf_id, intvec& allFitnessGA,
intvec& chrmCoded, intvec& chrmCodedSPT, intvec& chrmCodedLPT, intvec& chrmCodedMRWT, intvec&
chrmCodedEDD, intvec& chrmCodedCR, intvec& chrmCodedATC, int ta, const int& psize, const int&
chrmSize, const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, intvec&
um, double& PrMakespan, double& PrtotalWeightedTardiness, deque<int>& ur, deque<int>& td,
intvec& PTj) {
```

```
population.clear();
population.resize(psize);
populationSize = psize;

        decodedPopulation.clear();
        decodedPopulation.resize(psize);

        int tejeci, tejecf, tejec;
        int cont1 = 0;
        int p=0;
        allFitnessGA.clear();
        tejeci = time(0);

        intvec tempChrm(chrmSize);

    for (int j = 0; j < chrmSize; j++) {
         tempChrm [j] = j;
    }

    intvec dummyChrm = tempChrm;

for (int i = 0; i < psize ; i++) {
         population[i].chrm.resize(chrmSize);
                        for (int j = 0; j < chrmSize; j++) {
                                int pos = rand() % dummyChrm.size();
                                population[i].chrm[j] = dummyChrm[pos];
                                dummyChrm.erase(dummyChrm.begin() + pos);
                         }

                        deque<int> uRS;
                        vec2dim arrivals;
                        vec2dim transitions;
                        vec2dim operEndTime;
```

```
                vector<intvec> marking;

                intvec timeU;

                intvec chrmDec;



                if(i > 2 && cont1==0 ){

                        cont1 = cont1 +1;

                }


                uRS.clear();



if (i==0) {

        population[i].evalObjectiveSPTIS(pn, tnow, pf_id, ta,
        objective,jobArray, mchBreakdownInstant, chrmDec, chrmCoded,
        PrMakespan, PrtotalWeightedTardiness);

        population[i].chrm.clear();

        population[i].chrm = chrmCoded;

        decodedPopulation[i] = chrmDec;

         dummyChrm = tempChrm;

        allFitnessGA.push_back(population[i].fitness);
        }else if (i==1) {

                population[i].evalObjectiveLPTIS(pn, tnow, pf_id, ta, objective,
                jobArray, mchBreakdownInstant, chrmDec, chrmCoded, PrMakespan,
                PrtotalWeightedTardiness);

                population[i].chrm.clear();

                population[i].chrm = chrmCoded;

                decodedPopulation[i] = chrmDec;

                dummyChrm = tempChrm;

                allFitnessGA.push_back(population[i].fitness);
        }else if (i==2) {

                population[i].evalObjectiveMRWTIS(pn, tnow, pf_id, ta,
                objective, jobArray, mchBreakdownInstant, chrmDec, chrmCoded,
                PrMakespan, PrtotalWeightedTardiness);

                population[i].chrm.clear();

                population[i].chrm = chrmCoded;

                decodedPopulation[i] = chrmDec;
```

```
                allFitnessGA.push_back(population[i].fitness);

                dummyChrm = tempChrm;

            }else if (i==3) {

                population[i].evalObjectiveEDDIS(pn, tnow, pf_id, ta, objective,
                jobArray, mchBreakdownInstant, chrmDec, chrmCoded, PrMakespan,
                PrtotalWeightedTardiness, PTj);

                population[i].chrm.clear();

                population[i].chrm = chrmCoded;

                decodedPopulation[i] = chrmDec;

                allFitnessGA.push_back(population[i].fitness);

                dummyChrm = tempChrm;

            }else if (i==4) {

                population[i].evalObjectiveCRIS(pn, tnow, pf_id, ta,
                objective,jobArray, mchBreakdownInstant, chrmDec, chrmCoded,
                PrMakespan, PrtotalWeightedTardiness, PTj);

                population[i].chrm.clear();

                population[i].chrm = chrmCoded;

                decodedPopulation[i] = chrmDec;

                allFitnessGA.push_back(population[i].fitness);

                dummyChrm = tempChrm;

            }else if (i==5) {

                population[i].evalObjectiveATCIS(pn, tnow, pf_id, ta,
                objective,jobArray, mchBreakdownInstant, chrmDec, chrmCoded,
                PrMakespan, PrtotalWeightedTardiness, PTj);

                population[i].chrm.clear();

                population[i].chrm = chrmCoded;

                decodedPopulation[i] = chrmDec;

                allFitnessGA.push_back(population[i].fitness);

                dummyChrm = tempChrm;

            }else if (i==6) {

                population[i].evalObjectiveMSLACKIS(pn, tnow, pf_id, ta,
                objective,jobArray, mchBreakdownInstant, chrmDec, chrmCoded,
                PrMakespan, PrtotalWeightedTardiness, PTj);

                population[i].chrm.clear();

                population[i].chrm = chrmCoded;

                decodedPopulation[i] = chrmDec;

                allFitnessGA.push_back(population[i].fitness);
```

```
                        dummyChrm = tempChrm;
                    }else {
                        population[i].evalObjectiveIS(pn, tnow, pf_id, ta,
                        objective,jobArray, mchBreakdownInstant, chrmDec, chrmCoded,
                        PrMakespan, PrtotalWeightedTardiness, ur, td);
                        population[i].chrmDec.resize(chrmDec.size());
                        decodedPopulation[i] = chrmDec;
                        allFitnessGA.push_back(population[i].fitness);
                        dummyChrm = tempChrm;
                    }
        }


}


void Memetic::tournament(const vector<Individual>& bestPopulation, const int& psize,
vector<Individual>& parents){


        int chrmL = population[0].chrm.size();
        parents.clear();
        parents.resize(2);


       int numParents = bestPopulation.size();


        for (int i= 0; i < 2; i++) {
                int p1 = rand() % numParents;
                int p2 = rand() % numParents;
                parents[i].chrm.resize(chrmL);


                while (p1 == p2){
                       p2 = rand() % numParents;
                       }


                if (bestPopulation[p1].fitness <= bestPopulation[p2].fitness){
                        parents[i] = bestPopulation[p1];
```

```
                }else{
                        parents[i] = bestPopulation[p2];
                }
        }

}

void Memetic::bestPopulationSelection(const vector<Individual>& population,
vector<Individual>& bestPopulation, int bestPopulationPercentage){

                vector<Individual> tempPopulation;
                tempPopulation = population;
                bestPopulation.clear();

                int numParents = (bestPopulationPercentage * population.size()) / 100;
                std::vector<Individual>::iterator best;

                for(int i = 0 ; i < numParents ; i++ ){
                        best = min_element(tempPopulation.begin(),
                        tempPopulation.end(),compareObjective);
                        bestPopulation.push_back(*best);
                        tempPopulation.erase(best);
                }
}

void Memetic::roulleteScaleFunction(const vector<Individual>& bestPopulation, const int&
psize, vector<Individual>& parents, vector<double>& roulleteScale, double& sumFitness){

                double sumDesviation = 0;
                double number;
                double bestFitness = bestPopulation[0].fitness;
                sumFitness = 0;
                const int tamaño =bestPopulation.size();
```

```
		roulleteScale.clear();
		roulleteScale.resize(bestPopulation.size());

		for(int i=0; i<bestPopulation.size(); i++){
			sumFitness = sumFitness + bestPopulation[i].fitness;
		}

		double suma=0;
		double desviation=0;

		for(int i=0; i<bestPopulation.size(); i++){
			desviation = 1-(bestPopulation[i].fitness/sumFitness);
			suma = suma + (1-desviation);
			roulleteScale[i] = suma;

			if(i==bestPopulation.size()-1){
				roulleteScale[i] = 1;
			}
		}

}

void Memetic::roulleteSelection(const vector<Individual>& bestPopulation, const int& psize,
vector<Individual>& parents, vector<double> roulleteScale){

		double number;
		int chrmL = bestPopulation[0].chrm.size();
		parents.clear();
		parents.resize(2);

		int p1, p2;
		int i=0;

		do{
```

```
number = ((double)rand() / (double)RAND_MAX + 0.0);


for(int j=0; j<(roulleteScale.size()); j++){
                if(j==0){
                if(number <= roulleteScale[j]){
                        p1 = j;
                        break;
                }
        }else if(number > roulleteScale[j-1] && number <=
        roulleteScale[j]){
                        p1 = j;
                        break;
        }
}


parents[i].chrm.resize(chrmL);
parents[i] = bestPopulation[p1];
i++;
number = ((double)rand() / (double)RAND_MAX + 0.0);


for(int j=0; j<(roulleteScale.size()); j++){
        if(j==0){
                if(number <= roulleteScale[j]){
                        p2 = j;
                        break;
                }
        }else if(number > roulleteScale[j-1] && number <=
        roulleteScale[j]){
                        p2 = j;
                        break;
        }
}
```

```
                        while (p1 == p2){
                                number = ((double)rand() / (double)RAND_MAX + 0.0);
                                for(int j=0; j<(roulleteScale.size()); j++){
                                        if(j==0){
                                                if(number <= roulleteScale[j]){
                                                        p2 = j;
                                                        break;
                                                }
                                        }else if(number > roulleteScale[j-1] && number <=
                                        roulleteScale[j]){
                                                p2 = j;
                                                break;
                                        }
                                }
                        }

                        parents[i].chrm.resize(chrmL);
                        parents[i] = bestPopulation[p2];

                        i++;
                }while(i < 2);

}

void Memetic::selection(vector<Individual> population, vector<Individual>& elitismGroup, int
elitismPercentage){
        int numParents = (elitismPercentage * population.size()) / 100;
        std::vector<Individual>::iterator best;
        elitismGroup.clear();
        for(int i = 0 ; i < numParents ; i++ ){
                best = min_element(population.begin(), population.end(),compareObjective);
                elitismGroup.push_back(*best);
                population.erase(best);
        }
```

```
}


void Memetic::onePointCrossover(const vector<Individual>& parents, vector<Individual>&
offspring){


        offspring.clear();

        offspring.resize(2);


        int point = rand() % parents[0].chrm.size(); // crossover point


        copy (parents[0].chrm.begin(), parents[0].chrm.begin()+point,
       back_inserter(offspring[0].chrm));

        copy (parents[1].chrm.begin()+point, parents[1].chrm.end(),
       back_inserter(offspring[0].chrm));


        copy (parents[1].chrm.begin(), parents[1].chrm.begin()+point,
        back_inserter(offspring[1].chrm));

        copy (parents[0].chrm.begin()+point, parents[0].chrm.end(),
       back_inserter(offspring[1].chrm));


}



void Memetic::twoPointCrossover(const vector<Individual>& parents, vector<Individual>&
offspring){


        offspring.clear();

        offspring.resize(2);


        int point = rand() % (parents[0].chrm.size()-2); // crossover point 1


               int point2 = rand() % (parents[0].chrm.size()-1); // crossover point 2

               while(point2 <= point){

                       point2 = rand() % (parents[0].chrm.size()-1);

               }
```

```
        copy(parents[0].chrm.begin(),parents[0].chrm.begin()+point,
        back_inserter(offspring[0].chrm));

        copy (parents[1].chrm.begin() + point,parents[1].chrm.begin()+point2,
        back_inserter(offspring[0].chrm));

        copy (parents[0].chrm.begin()+point2, parents[0].chrm.end(),
        back_inserter(offspring[0].chrm));


        copy (parents[1].chrm.begin(),parents[1].chrm.begin()+point,
        back_inserter(offspring[1].chrm));

        copy (parents[0].chrm.begin() + point,parents[0].chrm.begin()+point2,
        back_inserter(offspring[1].chrm));

        copy (parents[1].chrm.begin()+point2, parents[1].chrm.end(),
        back_inserter(offspring[1].chrm));

}



void Memetic::mutate(Individual& child, int mutateIntensity){


                int cont = 0;


                do{
                        int point1 = rand() % child.chrm.size();

                        int point2 = rand() % child.chrm.size();


                        while (point2 == point1) {

                                point2 = rand() % child.chrm.size();

                        }


                        int temp;

                        temp = child.chrm[point1];

                        child.chrm[point1] = child.chrm[point2];

                        child.chrm[point2] = temp;

                        cont++;


                }while(cont < mutateIntensity);
```

```
}


void Memetic::repair(Individual& i1){

        int size = i1.chrm.size();
        intvec vecPosition;
        intvec tempChrm(size);

                        for (int j = 0; j < size; j++) {
                                tempChrm [j] = j;
                        }
                        vector<int>:: iterator it1 = i1.chrm.begin();
                        vector<int>:: iterator it2;

                        int k = 0;

                        while (it1!=i1.chrm.end()) {

                                it2 = find(tempChrm.begin(), tempChrm.end(), *it1);

                                if (it2 == tempChrm.end())
                                        vecPosition.push_back((it1-i1.chrm.begin()));
                                else
                                        tempChrm.erase(tempChrm.begin() + (it2-
                                        tempChrm.begin()));
                                        it1++;
                        }
                        k = 0;
                        while (k != vecPosition.size()){
                                i1.chrm[vecPosition[k]] = tempChrm[k];
                                k++;
                        }
}
```

```
void Memetic::runIS(PetriNet pn, int tnow, intvec& pf_id, intvec& allFitnessGA, int ta, int
elitismPercentage, int bestPopulationPercentage, Individual& winner, Individual& bestSolution,
const int& objective, const vector<Tjob>& jobArray, int mchBreakdownInstant, double&
PrMakespan, double& PrtotalWeightedTardiness, double& initialObjective, deque<int>& ur,
deque<int>& td, int& generationIS){



                vector<double> roulleteScale;

                vector<Individual> bestPopulation;

                vector<Individual> elitismGroup;

                vector<Individual> newPopulation;

                vector<Individual> parents;

                vector<Individual> offspring;

                vector<Individual> generationBest;

                vector<Individual> localOptimumList;

                vector<Individual>survivors;


                std::vector<Individual>::iterator iterBest;

                //Iterador que apunta al individuo con mejor objetivo ponderado


                int n = 1;

                generationIS;

                double sumFitness=0;

                intvec chrmDec;

                intvec chrmCoded;


                iterBest = min_element(population.begin(), population.end(),compareObjective);

                generationBest.push_back(*iterBest);

                winner = *iterBest;


                int pos;

                for(int i=0; i<allFitnessGA.size(); i++){

                        if(allFitnessGA[i] == winner.fitness){

                                pos = i;
```

```
                }
        }
        if(applyLocalSearch == false){
                if(elitism == false){
                        newPopulation.push_back(winner);
                }
        }

while (n < maxGenerations){

                        if(elitism == true){
                                selection(population,elitismGroup, elitismPercentage);
                                newPopulation.insert(newPopulation.begin(),
                                elitismGroup.begin(), elitismGroup.end());
                        }

                        bestPopulationSelection(population, bestPopulation,
                        bestPopulationPercentage);

                        roulleteScaleFunction(bestPopulation,bestPopulation.size(),
                        parents, roulleteScale, sumFitness);

                        allFitnessGA.clear();

                        if(sumFitness == 0){
                                break;
                        };
         while (newPopulation.size() < population.size()) {
        //Se escogen los individuos a los que se le aplican los operadores genéticos
        //mediante el torneo}

roulleteSelection(bestPopulation,bestPopulation.size(), parents, roulleteScale);

                double unif = rand()/32767.0 ;
```

```
if (unif < prCrossover) {

      if(crossoverType == true){
             onePointCrossover(parents,offspring);

      }else if(crossoverType == false){

             twoPointCrossover(parents,offspring);

      }


       unif = rand()/32767.0;


       if (unif < prMutation){

              mutate(offspring[0], mutateIntensity);

            }


       unif = rand()/32767.0;


       if (unif < prMutation){

             mutate(offspring[1], mutateIntensity);

       }


      offspring[0].evalObjectiveIS(pn, tnow, pf_id, ta, objective,
      jobArray, mchBreakdownInstant, chrmDec, chrmCoded, PrMakespan,
      PrtotalWeightedTardiness, ur, td);


      newPopulation.push_back(offspring[0]);


      allFitnessGA.push_back(offspring[0].fitness);


      if( population.size() - newPopulation.size() > 0){


             offspring[1].evalObjectiveIS(pn, tnow, pf_id, ta,
             objective, jobArray, mchBreakdownInstant, chrmDec,
             chrmCoded, PrMakespan, PrtotalWeightedTardiness, ur,
             td);


             newPopulation.push_back(offspring[1]);
```

```
                        allFitnessGA.push_back(offspring[1].fitness);

                }
         }

        parents.clear();
 }

population.clear();
population = newPopulation;

iterBest = min_element(population.begin(), population.end(),compareObjective);
generationBest.push_back(*iterBest);

if (winner.fitness > (*iterBest).fitness) {
        winner = *iterBest;
        for(int i=0; i<allFitnessGA.size(); i++){
                if(allFitnessGA[i] == winner.fitness){
                        pos = i;
                }
        }

        generationIS = n;
}

newPopulation.clear();

if(elitism == false){
        newPopulation.push_back(winner);
}

n++;
```

```
	}


		if(bestSolution.chrm.size()>0){

			if(bestSolution.fitness<winner.fitness){

				winner = bestSolution;

			}

		}


		iterBest = min_element(generationBest.begin(),
		generationBest.end(),compareObjective);


}



void Memetic::run(PetriNet pn, int tnow, intvec& pf_id, intvec& allFitnessGA, int ta, int
elitismPercentage, int bestPopulationPercentage, Individual& winner, const int& chrmSize,
Individual& bestSolution, const int& objective, const vector<Tjob>& jobArray, int
mchBreakdownInstant, int p, double& PrMakespan, double& PrtotalWeightedTardiness, double&
initialObjective, deque<int>& ur, deque<int>& td, double& sumWjTj, double& sumCj, double&
sumTj, double& sumWjCj, int& NoJob_t, intvec& Cj, double& decrease){



		vector<double> roulleteScale;

		vector<Individual> bestPopulation;

		vector<Individual> elitismGroup;

		vector<Individual> newPopulation;

		vector<Individual> parents;

		vector<Individual> offspring;

		vector<Individual> generationBest;

		vector<Individual> localOptimumList;

		vector<Individual>survivors;


		std::vector<Individual>::iterator iterBest;

		//Iterador que apunta al individuo con mejor objetivo ponderado


		int n = 1;
```

```
int generation = 0;
double sumFitness;
intvec chrmDec;
intvec chrmCoded;

//Se obtiene el individuo con menor valor objetivo de la población inicial
iterBest = min_element(population.begin(), population.end(),compareObjective);

generationBest.push_back(*iterBest);
winner = *iterBest;

//Se escoge el menor fitness del vector allfitness y se almacena la posición
int pos;
for(int i=0; i<allFitnessGA.size(); i++){
        if(allFitnessGA[i] == winner.fitness){
                pos = i;
        }
}

//Se guarda el vector decodificado del mejor individuo

if(applyLocalSearch == false){
        if(elitism == false){
                newPopulation.push_back(winner);
        }
}

//Se aplica la técnica de búsqueda local
if(applyLocalSearch == true){
        SimulatedAnnealing(pn, tnow, pf_id, ta, winner, bestSolution,
        objective, jobArray, mchBreakdownInstant, p, PrMakespan,
        PrtotalWeightedTardiness, initialObjective, ur, td, sumWjTj, sumCj,
        sumTj, sumWjCj, NoJob_t, Cj, decrease);
        localOptimumList.push_back(bestSolution);
        newPopulation.push_back(bestSolution);
```

```
        }// ApplyLocal Search


while (n < maxGenerations){


                if(elitism == true){

                        selection(population,elitismGroup, elitismPercentage);


newPopulation.insert(newPopulation.begin(),elitismGroup.begin(),elitismGroup.end());

                }


                bestPopulationSelection(population, bestPopulation,
                bestPopulationPercentage);

                roulleteScaleFunction(bestPopulation,bestPopulation.size(), parents,
                roulleteScale, sumFitness);


                decodedPopulation.clear();

                allFitnessGA.clear();


                if(sumFitness == 0){

                        break;

                }


                while (newPopulation.size() < population.size()) {

                        //Se escogen los individuos a los que se le aplican los
                        //operadores genéticos mediante el torneo


                        roulleteSelection(bestPopulation,bestPopulation.size(), parents,
                        roulleteScale);


                        double unif = rand()/32767.0 ;


                        if (unif < prCrossover) {

                        //Se realiza el cruce de los individuos escogidos

                                if(crossoverType == true){

                                        onePointCrossover(parents,offspring);
```

```
            }else if(crossoverType == false){
                    twoPointCrossover(parents,offspring);
            }

 unif = rand()/32767.0;

 //Se realiza la mutación de los individuos escogidos
 if (unif < prMutation){
         mutate(offspring[0], mutateIntensity);
 }

 unif = rand()/32767.0;
 if (unif < prMutation){
         mutate(offspring[1], mutateIntensity);
 }

//Se obtiene el fitness del primer hijo
    offspring[0].evalObjective(pn, tnow, pf_id, ta,
    objective,jobArray, mchBreakdownInstant, p, chrmDec, chrmCoded,
    PrMakespan, PrtotalWeightedTardiness, ur, td, sumWjTj, sumCj,
    sumTj, sumWjCj, NoJob_t, Cj);

    newPopulation.push_back(offspring[0]);

    decodedPopulation.push_back(chrmDec);

    allFitnessGA.push_back(offspring[0].fitness);

    if( population.size() - newPopulation.size() > 0){
    //Se obtiene el fitness del segundo hijo

    offspring[1].evalObjective(pn, tnow, pf_id, ta, objective,
    jobArray, mchBreakdownInstant, p,  chrmDec, chrmCoded,
    PrMakespan, PrtotalWeightedTardiness, ur, td, sumWjTj, sumCj,
    sumTj, sumWjCj, NoJob_t, Cj);
```

```
                newPopulation.push_back(offspring[1]);


                decodedPopulation.push_back(chrmDec);


                allFitnessGA.push_back(offspring[1].fitness);


                }


        }
        parents.clear();
}


   population.clear();
   population = newPopulation;


   iterBest = min_element(population.begin(), population.end(),
   compareObjective);
   generationBest.push_back(*iterBest);


   if (winner.fitness > (*iterBest).fitness) {
           winner = *iterBest;
           for(int i=0; i<allFitnessGA.size(); i++){
                   if(allFitnessGA[i] == winner.fitness){
                           pos = i;
                   }
           }
           generation = n+1;
   }


   newPopulation.clear();


   if(elitism == false){
           newPopulation.push_back(winner);
           if(applyLocalSearch == false){
```

```
                                        newPopulation.push_back(winner);
                                }
                        }

                        if(applyLocalSearch == true){
                                SimulatedAnnealing(pn, tnow, pf_id, ta, winner, bestSolution,
                                objective, jobArray, mchBreakdownInstant, p, PrMakespan,
                                PrtotalWeightedTardiness, initialObjective, ur, td, sumWjTj,
                                sumCj, sumTj, sumWjCj, NoJob_t, Cj, decrease);

                                localOptimumList.push_back(bestSolution);

                                newPopulation.push_back(bestSolution);

                                if(bestSolution.fitness<winner.fitness){
                                        winner = bestSolution;
                                        generation = n+1;
                                }
                        }
                        n++;
        }

                iterBest = min_element(generationBest.begin(), generationBest.end(),
                compareObjective);

}

bool compareObjective (Individual ind1, Individual ind2){
        return ind1.fitness < ind2.fitness;
}

bool exist (intvec vector, int num){
        bool exist = false;
                for(int i = 0 ; i < vector.size() ; i++ ){
```

```
                        if(vector[i] == num){
                                exist == true;
                                break;
                        }
                }
        return exist;
}
```

```
void Memetic::SimulatedAnnealing(PetriNet pn, int tnow, intvec& pf_id, int ta, Individual&
winner, Individual& neighbour, const int& objective, const vector<Tjob>& jobArray, int
mchBreakdownInstant, int p, double& PrMakespan, double& PrtotalWeightedTardiness, double&
initialObjective, deque<int>& ur, deque<int>& td, double& sumWjTj, double& sumCj, double&
sumTj, double& sumWjCj, int& NoJob_t, intvec& Cj, double& decrease){
```

```
                Individual winnerWinner;
                Individual OriginalWinner = winner;

                intvec chrmDec;
                intvec chrmCoded;
                int cont1 = 0;
                int cont2 = 0;

                double winnerFitness;
                double initialSolution = winner.fitness;
                double Tinitial = 1000;
                double Tfinal = 0.01;
                double deltaE;

                winnerWinner = winner;

                srand(time(0));

                while(Tinitial>Tfinal){
```

```
        winnerFitness = winner.fitness;


        FindNeighbour(pn, tnow, pf_id, ta, winner, neighbour, objective,
        jobArray, mchBreakdownInstant, p);


        neighbour.evalObjective(pn, tnow, pf_id, ta, objective,jobArray,
        mchBreakdownInstant, p, chrmDec, chrmCoded, PrMakespan,
        PrtotalWeightedTardiness, ur, td, sumWjTj, sumCj, sumTj, sumWjCj,
        NoJob_t, Cj);


        deltaE = neighbour.fitness - winnerFitness;


        if(deltaE <= 0){

                winner = neighbour;

                cont1++;

        }else{

                if(pow(2.71828183,(deltaE/Tinitial)) >= rand()){

                        winner = neighbour;

                        cont2++;

                }

        }


        if(winner.fitness < winnerWinner.fitness){

                winnerWinner = winner;


                winnerWinner.evalObjective(pn, tnow, pf_id, ta,
                objective,jobArray, mchBreakdownInstant, p, chrmDec, chrmCoded,
                PrMakespan, PrtotalWeightedTardiness, ur, td, sumWjTj, sumCj,
                sumTj, sumWjCj, NoJob_t, Cj);

        }

        Tinitial=(1-(decrease/100))*Tinitial;

}


neighbour = winnerWinner;

winner = OriginalWinner;
```

```
}


void Memetic::FindNeighbour(PetriNet pn, int tnow, intvec& pf_id, int ta, Individual& winner,
Individual& neighbour, const int& objective, const vector<Tjob>& jobArray, int
mchBreakdownInstant, int p){


		int n = 1;
		neighbour.chrm.resize(winner.chrm.size());

		for(int i = 0; i<winner.chrm.size(); i++){
			neighbour.chrm[i] = winner.chrm[i];
		}

		int cont = 0;

		do{

		int temp;
		int pos1 = rand() % neighbour.chrm.size();
		int pos2 = rand() % neighbour.chrm.size();

		while (pos1 == pos2) {
			pos2 = rand() % neighbour.chrm.size();
		}

		temp = neighbour.chrm[pos1];
		neighbour.chrm[pos1] = neighbour.chrm[pos2];
		neighbour.chrm[pos2] = temp;
		cont++;
		}while(cont < n);

}
```

Operation.h: interface for the Operation class.

```
#ifndef OPERATION_H_
#define OPERATION_H_

#include <fstream>
#include <iostream>
using namespace std;

/**
* @file    Operation.h
* @brief   Clase Operation
* @version 1.0
* @date    16/02/2005
* @author  gmejia
* @par     Descripción de la clase
*                  Los miembros de la clase son: <br>
*                  resNo: El recurso usado en la operación. <br>
*                  jobNo: El trabajo que se procesa en el recurso.<br>
*                  startTime: El tiempo de inicio de la operación (-1 si no ha sido
asignado).<br>
*                  endTime: El tiempo de finalización de la operación (-1 si no ha sido
asignado).<br>
*                  trans: La transición que se disparó para iniciar la operación.<br>
*
*
*
* @todo
```

```
*/


class Operation

{


public:

        /**


        * @brief    Constructor por default

        *

        * @param   No

        * @pre     True

        * @post    Se tiene un objeto de la clase Operation con todos los miembros
inicializados en -1

        */

        Operation();


        /**


        * @brief    Destructor por default

        *

        * @param   No

        * @pre     Se tiene un objeto de la clase Operation

        * @post    True

        */

        virtual ~Operation();


        /**


        * @brief    Devuelve el # del recurso usado en la operación

        *

        * @param   No

        * @pre     Se tiene un objeto de la clase Operation
```

```
* @post    Se tiene el # del recurso
*/
const int getRes();

/**

* @brief    Devuelve el tiempo de inicio de la operación
*
* @param   No
* @pre     Se tiene un objeto de la clase Operation
* @post    Se tiene el tiempo de inicio
*/
const int getStart();

/**

* @brief    Devuelve el tiempo de finalización de la operación
*
* @param   No
* @pre     Se tiene un objeto de la clase Operation
* @post    Se tiene el tiempo de finalización
*/
const int getEnd();

/**

* @brief    Devuelve el # del trabajo usado en la operación
*
* @param   No
* @pre     Se tiene un objeto de la clase Operation
* @post    Se tiene el # del trabajo
*/
const int getJob();
```

```
/**

* @brief    Devuelve el # de la transición que dio inicio a la operación
*
* @param   No
* @pre     Se tiene un objeto de la clase Operation
* @post    Se obtiene el # de la transición
*/
const int getTrans();

/**

* @brief   Asigna el # del trabajo (jobNo) al objeto de tipo Operation
* @param   Un número válido de trabajo. Debe ser un número entero
*/
Operation& setJob(const int& jobn);

/**

* @brief   Asigna el tiempo de inicio al objeto de tipo Operation
* @param   Un tiempo de inicio válido. Debe ser un número entero
*/
Operation& setStart(const int& start);

/**

* @brief   Asigna el tiempo de finalización al objeto de tipo Operation
* @param   Un tiempo de inicio válido. Debe ser un número entero
*/
Operation& setEnd(const int& end);

/**
```

```
        * @brief   Asigna el # de recurso al objeto de tipo Operation
        * @param   Un # de recurso válido. Debe ser un número entero
        */
        Operation& setRes(const int& res);

        /**

        * @brief   Asigna el # de la transición en la red de Petri que dio inicio <br>
        * al objeto de tipo Operation
        * @param   Una transición válida. Debe ser un número entero
        */
        Operation& setTrans(const int& trans);

        /**

        * @brief   Operador de asignación. Permite operacion2 = operacion1
        * @param   Un objeto válido del tipo Operation
        */
        Operation& operator = (const Operation&);

private:

        int resNo;
        int jobNo;
        int startTime;
        int endTime;
        int trans;


        /**
        * @brief   Operador de salida. Permite os << operacion1 <br>
        *                  Donde os es un objeto ostream válido y operacion1 es un objeto
Operation
```

```
        * @param   Un objeto del tipo ostream y un objeto Operation válidos
        * @bug     No funciona; bug no encontrado
        */
friend ostream& operator << (ostream&, const Operation&);

};

#endif // !defined(AFX_OPERATION_H__190F90B6_71D1_431C_B47F_A65ACC5FF24E__INCLUDED_)
```

Operation.cpp: implementation of the Operation class.

```
#include "Operation.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

Operation::Operation(){
        jobNo = -1;
        endTime =-1;
        startTime=-1;
        resNo=-1;
}

Operation::~Operation(){}

const int Operation::getRes(){
        return resNo;}
```

```
const int Operation::getStart(){
        return startTime;}

const int Operation::getEnd(){
        return endTime;}

const int Operation::getJob(){
        return jobNo;}

const int Operation::getTrans(){
        return trans;}

Operation& Operation::setJob(const int& jobNo1){
        jobNo = jobNo1;
        return *this;}

Operation& Operation::setStart(const int& start1){
        startTime = start1;
        return *this;}

Operation& Operation::setEnd(const int& end1){
        endTime = end1;
        return *this;}

Operation& Operation::setRes(const int& res1){
        resNo = res1;
        return *this;}

Operation& Operation::setTrans(const int& tr1){
        trans = tr1;
        return *this;}

Operation& Operation::operator = (const Operation& op2) {
```

```
 jobNo = op2.jobNo;
 startTime = op2.startTime;
 endTime = op2.endTime;
 resNo = op2.resNo;
 trans = op2.trans;

 return *this;

}
```

PetriNet.h: interface for the PetriNet class.

```
#ifndef PetriNet_H
#define PetriNet_H
#pragma warning (disable : 4786)

#include "matrix.h"
#include "Tjob.h"
#include "funvectors.h"
#include <deque>
#include <list>
#include <stdlib.h>
#include <ctime>
#include "Tjob.h"
#include "WkCenter.h"

/**
* \file    PetriNet.h
* \brief   Clase PetriNet
```

```
* \version 1.0

* \date    16/02/2005

* \author  gmejia

* \par     Proyecto Investigaci&oacute;n

*

* \par     Descripcion

*                 Clase que describe una red de Petri Ordinaria Temporizada que describe <br>

*                 un sistema de manufactura. La red tiene n plazas y m transitions. Las n
plazas pueden <br>

*                 ser operacionales (describen una operaci&oacute;n o condici&oacute;n de un
trabajo) o de recurso <br>

*                 (describen la disponibilidad de un recurso).<br>

*                 La red est&aacute; definida por el conjunto de plazas, la matriz de
incidencia positiva C+ <br>

*                 la matriz de incidencia negativa C- y el conjunto de tiempos asociados con
las plazas. <br>

*                 Los tiempos de las plazas de recurso son 0.

*                 Si hay "o" plazas operacionales y "r" plazas de recursos, entonces "n = o +
r"

*                 Los miembros son:

*                 Marcaje: Vector (1..n) de enteros nonegativos

*

* \todo

*

*/


class PetriNet {


public:


        /**


        * \brief   Constructor de la Clase PetriNet

        *

        * \pre   True
```

```
        * \post  Se tiene un nuevo objeto de la clase PetriNet
        */
        PetriNet() {}

        /**

        * \brief   Destructor de la Clase PetriNet
        *
        * \pre   Se tiene un objeto de la clase PetriNet
        * \post  True
        */
        virtual ~PetriNet();

        /**

        * \brief   Obtiene las dimensiones de la matriz de incidencia de un objeto PetriNet
        *
        * \par     Devuelve el n&uacute;mero de filas en rows y el n&uacute;mero de columnas
en cols
        */
        void getDimensions(int& rows, int& cols);

        /**

        * \brief   Define el estado de un objeto PetriNet como el marcaje inicial definido
(M0)
        y el tiempo de proceso remanente (Mr0 = 0)
        *
        * \pre     Se tiene un objeto de la clase PetriNet con un estado [Mp Mr]
        * \post    Se tiene un objeto de la clase PetriNet con un estado [M0 0]
        */
        void reset();

        /**
```

```
        * \brief   Define como vacíos (dimensi&oacute;n 0), los miembros de un objeto PetriNet
        *
        * \pre     Se tiene un objeto de la clase PetriNet
        * \post    Se tiene un objeto de la clase PetriNet con todos miembros vacíos
        */
        void clear();


        /**

        * \brief    Define el estado de un objeto PetriNet con Mp1 y Mr1
                                Mp1 y Mr1 son un vector de marcaje y un vector de tiempos
remanentes
                                respectivamente. Valores negativos son colocados en 0
        *
        * \param   V&aacute;lidos vectores Mp1 (marcaje) y Mr1 (tiempos de proceso remanente)
        * \pre     Se tiene un objeto de la clase PetriNet
        * \post    Se tiene un objeto de la clase PetriNet con un nuevo estado Mp = Mp1 y Mr=
Mr1
        * Estos vectores deben tener la misma dimensi&oacute;n del marcaje inicial
        */
        void setState(const intvec& Mp1, const intvec& Mr1);

        /**

        * \brief    Imprime los miembros de un objeto PetriNet a un stream os
        *
        * \param   Un v&aacute;lido ostream os
        */
        void printPetrinet(ostream& os);

        /**
```

```
        * \brief    Cambia el estado de una red de Petri disparando la z-&eacute;sima
transici&oacute;n
        * del vector de transitions activas um. Devuelve tambi&eacute;n el tiempo transcurrido
        * para el disparo de la transici&oacute;n (delta)
    * \param    um: Es un vector de transitions activas tipo intvec con entradas en el rango
(0 a m-1). m es el n&uacute;mero * de columnas de la matriz de incidencia.
        * z: Una posici&oacute;n v&aacute;lida z en el vector um. z est&aacute; en el rango 0-
um.size()-1
        * \par     Nota adicional
        * El m&eacute;todo verifica si la transici&oacute;n que se dispara est&aacute; activa.
        * Debe usarse con el m&eacute;todo constructU
        * \pre      Se tiene un objeto de la clase PetriNet
        * \post     Se tiene un objeto de la clase PetriNet con un nuevo estado
        */
        void changeState(const intvec& um, const int& z, int& delta);

        /**

        * \brief    Agrega un trabajo que llega
        *  al sistema de forma aleatoria
        */
        void changearrivals(int w, int z, int tnow, deque<int> t, PetriNet& pn);

        /**

        * \brief    Cambia el estado de la red de petri
        *  después de la falla de una máquina
        */

        void changeMachineBreakDown(const intvec& um, const int& z, int& delta);

        /**

        * \brief    Asigna el tiempo de reparación a
```

```
*  la plaza de la máquina correspondiente
*/

void changePT(int p, int q, PetriNet& pn);
/**

* \brief    Cambia el estado de la red de petri
*  en la programación reactiva después del disparo
*  de una transición cuando no ha fallado ninguna máquina
*/
void changeStateR(deque<int>& u_activo, const int& r, int& delta);

/**

* \brief    Recupera el marcaje inicial de la Clase Petri Net.
* M0 es un vector de enteros (intvec)
*/
const intvec& getM0() const;

/**

* \brief    Recupera el marcaje actual de la Clase Petri Net.
* Mp es un vector de enteros (intvec)
*/
const intvec& getMp() const;

/**

* \brief    Recupera el vector de tiempos remanentes de la Clase Petri Net.
* Mr es un vector de enteros (intvec)
*/
const intvec& getMr() const;
```

```
/**

* \brief    Recupera el vector de tiempos (time delays) de la Clase Petri Net.
* PT es un vector de enteros (intvec)
*/
const intvec& getPT() const;

/**

* \brief    Recupera el vector de tiempos de trabajo remanentes de la Clase Petri Net.
* rwt es un vector de enteros (intvec)
*/
const intvec& getrwt() const;

/**

* \brief    Recupera el vector de identificadores de las plazas la Clase Petri Net.
* pl_id es un vector de enteros (intvec)
*/
const intvec& getp_id() const;

/**

* \brief    Recupera la matriz negativa de la Clase Petri Net.
* Cmin es un vector de vectores (vec2dim)
*/
const vec2dim& getCmin() const;

/**

* \brief    Recupera la matriz positiva de la Clase Petri Net.
* Cmin es un vector de vectores (vec2dim)
*/
```

```
        const vec2dim& getCplus() const;

        /**

        * \brief    Imprime los miembros de la PetriNet pn en el stream os
        * \param    Un stream de salida os
        *
        */
        void printState (ostream& os);

        vec2dim Cmin; /*incidence matrix C-.
        This is a condensed matrix. This vec2dim contains as many positions as columns
        there are columns in the incidence matrix;
        The j-th (j = 1 to number of transitions) vector of Cmin contains the numbers of the
places
        which are INPUTS to the j-th transition. The position 0 of the j-th vector is always
an
        operational place and the remaining positions are resource places*/

        vec2dim Cplus; /*incidence matrix C+
        This is a condensed matrix. This vec2dim contains as many positions as columns
        there are columns in the incidence matrix;
        The j-th (j = 1 to number of transitions) vector of Cmin contains the numbers of the
places
        which are OUTPUTS to the j-th transition. The position 0 of the j-th vector is always
an
        operational place and the remaining positions are resource places*/


private:

        intvec M0; //initial marking;

        intvec Mp; //marking # of tokens in place
```

```
        intvec Mr; //remaining processing time vector


        intvec PT; //process time vector


        //vec2dim Cmin; /*incidence matrix C-.
        //This is a condensed matrix. This vec2dim contains as many positions as columns
        //there are columns in the incidence matrix;
        //The j-th (j = 1 to number of transitions) vector of Cmin contains the numbers of the
places
        //which are INPUTS to the j-th transition. The position 0 of the j-th vector is always
an
        //operational place and the remaining positions are resource places*/


        //vec2dim Cplus; /*incidence matrix C+
        //This is a condensed matrix. This vec2dim contains as many positions as columns
        //there are columns in the incidence matrix;
        //The j-th (j = 1 to number of transitions) vector of Cmin contains the numbers of the
places
        //which are OUTPUTS to the j-th transition. The position 0 of the j-th vector is
always an
        //operational place and the remaining positions are resource places*/



        intvec p_id;
        /*place_identifier: identifies the job or resource number of each place
        If p_id is non negative then it means that the place is a operational place;
        if p_id is negative then it means that the corresponding place is a resource place
        Example: If p_id[3] = 2 means that place # 3 belongs to job # 2
                        If p_id[7] = -5 means that place # 7 is an resource place
corresponding to machine #4
                        Note: Machines in Lekin are numbered from 0 to nummachines - 1


        */


        intvec rwt;
```

```
        //remaining work time: remaining work time to completion for each place

/*****************************************************************************************
*/

/*****************************************************************************************
*/

friend ostream& operator << (ostream&, const PetriNet&);

// No funciona todavía

/**

* \brief    Toma como entrada la matriz C y devuelve las matrices C- y C+

* \param    Una v&aacute;lida matriz de incidencia C del objeto de la clase matrix;

*

*/

friend void convertMatrix (const matrix&, vec2dim&, vec2dim&);

/**

* \brief    Devuelve un vector de transitions activas (um) a partir de una red de Petri

* \param    Una v&aacute;lida red de Petri

* \par         Descripci&oacute;n Adicional

* um es un vector con los índices de las transitions activas:

* por ejemplo: si um = [ 3 5 9] significa que las transitions 3, 5 y 9 est&aacute;n activas

* dado el estado actual de la red.

*

*/

friend void constructU (const PetriNet& pn, intvec& um);

/**

* \brief    Lee los miembros de la PetriNet pn del stream is
```

```
* \param    Una v&aacute;lida red de Petri y un stream de entrada is

*

*/

friend void readPn (istream& is, PetriNet& pn);


/**


/**


* \brief    Dispara las transitions de una Red de Petri (PetriNet pn) de acuerdo a el
Algoritmo para Programas Activos

* Se finaliza cuando no hay transitions activas y se devuelve

* marcaje en ese momento.

* \param    Una v&aacute;lida red de Petri (objeto PetriNet)

* Un v&aacute;lido marcaje final (objeto intvec)

*

*/

friend int ActSched(PetriNet pn, deque<int>& u, const vector<Tjob>& jobArray, vec2dim&
transitions, vector<intvec>& marking, intvec& timeU, intvec& jobsU, intvec& operationsU,
intvec& machinesU, intvec& ptU, vec2dim& operEndTime, intvec& PTj, double& initialObjective,
intvec& pf_id, intvec& chrmCoded, const int& objective, double& PrMakespan, double&
PrtotalWeightedTardiness, int& rule, int& ActTTP);


/**


* \brief    Dispara las transiciones de una Red de Petri (PetriNet pn)

* de acuerdo a el mejor programa inicial obtenido.

*

*/

friend int InitialSched(PetriNet pn, deque<int>& u, const vector<Tjob>& jobArray, vec2dim&
transitions, vector<intvec>& marking, intvec& timeU, intvec& jobsU, intvec& operationsU,
intvec& machinesU, intvec& ptU, vec2dim& operEndTime, double& InitialObjectiveIS, intvec&
pf_id, intvec& uTemporal, const int& objective, double& PrMakespan, double&
PrtotalWeightedTardiness, int& TTPAGIS);



/**
```

```
* \brief    Dispara las transitions de una Red de Petri (PetriNet pn) de acuerdo a la regla SPT
* (Shortest Processing Time). Se finaliza cuando no hay transitions activas y se devuelve
* marcaje en ese momento.
* \param    Una v&aacute;lida red de Petri (objeto PetriNet)
* Un v&aacute;lido marcaje final (objeto intvec)
*
*/
friend int sptTime(PetriNet pn, deque<int>& u);


/**



/**

        * \brief    Convierte un sistema (framework) en formato de Lekin a una Red de Petri.
        * Devuelve un objeto PetriNet con un marcaje inicial M0 corrspondiente al estado
        * "todos los trabajos listos para iniciarse" y un marcaje final
        * correspondiente al estado "todos los trabajos terminados"
        *
        *
        *
        * \param    Entradas:
        * vector de trabajos (clase Tjob) (proveniente de _user.job) jobArray
        * vector de estaciones de trabajo (clase WkCenter) (proveniente de _user.mch) wkcArray
        *
        * Salidas: Objeto PetriNet pn
        *  Mf (Marcaje Final): Un vector de tipo intvec


        * \par         Nota adicional.
        * Usar los m&eacute;todos de lectura para leer jobArray y wkcArray. Ver clase Tjob y wkcArray
```

```
*/

friend void convertToPN(vector<Tjob>& jobArray, vector<WkCenter>& wkcArray, PetriNet& pn,
intvec& Mf, vec2dim& transitions , int& Pavg, intvec& PTj );


/**


* \brief    Devuelve el n&uacute;mero del trabajo al que pertenece

                    la transici&oacute;n uz en una PetriNet pn

* \param    Un objeto PetriNet pn

                    Una transici&oacute;n v&aacute;lida de la red

*

*/

friend int jobNo(const PetriNet& pn, const int& uz);


/**

*

* \brief    Devuelve el n&uacute;mero de la m&aacute;quina que se libera (release)

*                    al disparar la transici&oacute;n uz en una PetriNet pn

* \param    Un objeto PetriNet pn

*                    Una transici&oacute;n v&aacute;lida de la red

*\par          Nota

*                    Devuelve -1 si no se libera ninguna m&aacute;quina

*

*/

friend int rlsMchNo (const PetriNet& pn, const int& uz);


/**

*

* \brief    Devuelve el n&uacute;mero de la m&aacute;quina que se ocupa

*                    al disparar la transici&oacute;n uz en una PetriNet pn

* \param    Un objeto PetriNet pn

*                    Una transici&oacute;n v&aacute;lida de la red

*\par          Nota

*                    Devuelve -1 si no se ocupa ninguna m&aacute;quina
```

```
*
*/
friend int allMchNo (const PetriNet& pn, const int& uz);


/*friend void activeUm(const PetriNet& pn, intvec& um);
        Devuelve una deque<int> (vector) um de transitions que generan programas activos
   Entradas:
        Objeto PetriNet pn
        # de m&aacute;quinas: nmachines
        # de trabajos: njobs


  Salida
   deque<int> um;
  */


/**
*
* \brief    Devuelve el n&uacute;mero de trabajos o procesos dado el vector de id's de una
PetriNet pn
* \param    Vector de id's de un objeto PetriNet p_id
*
*/
friend int nJobs(const intvec& p_id);


/**
*
* \brief    Devuelve el n&uacute;mero de recursos o m&aacute;quinas dado el vector de id's de
una PetriNet pn
* \param    Vector de id's de un objeto PetriNet p_id
*
*/
friend int nResources (const intvec& p_id);


/**
```

```
* \brief    Devuelve un vector de transitions activas (um) m&aacute;s pr&oacute;ximas a
disparar a partir de una red de Petri

* \param    Una v&aacute;lida red de Petri

* \par          Descripci&oacute;n Adicional

*                      um es un vector con los índices de las transitions activas:

*                      Por ejemplo: si um = [ 3 5 9] significa que las transitions 3, 5 y 9
est&aacute;n activas

*                      dado el estado actual de la red.

*

*/

friend void constructUmp (const PetriNet& pn, const intvec& um, intvec& ump);


friend void cons_pe_id (const PetriNet& pn, const intvec& p_id, intvec& pe_id); //OJO

friend void cons_pf_id (const PetriNet& pn, const intvec& p_id, intvec& pf_id); //OJO

friend bool existU (const deque<int>& u, int trans);

};


#endif
```

PetriNet.cpp: implementation of the PetriNet class.

```
#include "PetriNet.h"

#include "Tjob.h"

#include <iomanip>

#include <exception>

#include <fstream>

#include <math.h>

#define NULL 0
```

```
int findsumremtime (const intvec& Mp, const intvec& Mr, const intvec& rwt, const intvec&
busy_places);

int findmaxremtime ( const intvec& Mr, const intvec& rwt, const intvec& Mp, const intvec&
busy_places);

int findminremtime (const intvec& Mr, const intvec& rwt, const intvec& busy_places);

int findnextmaxtime(const intvec& um, const intvec& iv, const PetriNet& pn);

int findpresmaxtime(const intvec& um, const intvec& iv, const PetriNet& pn);

int findnextmintime(const intvec& um, const intvec& iv, const PetriNet& pn);

int EDD(const intvec& um, const intvec& iv, const PetriNet& pn, const vector<Tjob>& jobArray,
int MachineRestartTime);

int CR(const intvec& um, const intvec& iv, const PetriNet& pn, const vector<Tjob>& jobArray,
intvec& pf_id, int& time, intvec& PTj, intvec& jobTimes);

int ATC(const intvec& um, const intvec& iv, const PetriNet& pn, const vector<Tjob>& jobArray,
intvec& pf_id, int& time, intvec& PTj, intvec& jobTimes);

int MSLACK(const intvec& um, const intvec& iv, const PetriNet& pn, const vector<Tjob>&
jobArray, intvec& pf_id, int& time, intvec& PTj, intvec& jobTimes);

int findnextmintime2(const intvec& um, const intvec& iv, const PetriNet& pn, int&
MinTransition);

int findnextmintimeH(intvec& hebilitadas, const intvec& iv, const PetriNet& pn);

intvec findNextMinTimeVector(const intvec& um, const intvec& iv, const PetriNet& pn);

int max(int n, int m);


//Nuevas Funciones


void jobOperationIndex(const intvec& um ,const vec2dim& transitions, int& oper, int i, int
numJob);

void jobOperationIndex1(const intvec& um ,const vec2dim& transitions, int& oper, int i, int
numJob);


#pragma warning (disable : 4786)


PetriNet::~PetriNet() {}



const intvec& PetriNet::getM0() const

        {return M0;}
```

```
const intvec& PetriNet::getMp() const
        {return Mp;}

const intvec& PetriNet::getMr() const
        {return Mr;}

const intvec& PetriNet::getPT() const
        {return PT;}

const intvec& PetriNet::getrwt() const
        {return rwt;}

const intvec& PetriNet::getp_id() const
        {return p_id;}

const vec2dim& PetriNet::getCmin() const
        {return Cmin;}

const vec2dim& PetriNet::getCplus() const
        {return Cplus;}

/*ostream& operator << (ostream& os, const PetriNet& pn) {

        //os <<  pn.M0 << endl;
        os <<"Mp " <<  pn.Mp << endl;
        os <<"Mr " << pn.Mr << endl;
        os <<"PT " <<  pn.PT << endl;
        os <<"rwt " <<  pn.rwt << endl;
        os <<"p_id " <<  pn.p_id << endl;

        return os;
}*/
```

```
void PetriNet::printPetrinet(ostream& os) {

        if (!os)

                cout << "This is not a valid output stream " << endl;

        else {

        os << Mp << endl;
        os << Mr << endl;
        os << PT << endl;
        os << rwt << endl;
        os << p_id << endl;

        }
}

void PetriNet::getDimensions(int& rows, int& cols) {

        rows = Mp.size();
        cols = Cmin.size();
}

void PetriNet::reset() {

        Mr.clear();
        Mr.resize(M0.size());
        Mp=M0;
}

void PetriNet::clear() {
```

```
        M0.clear();
        Mp.clear();
        Mr.clear();
        PT.clear();
        p_id.clear();
        rwt.clear();
        Cmin.clear();
        Cplus.clear();
}

void PetriNet::setState(const intvec& Mp1, const intvec& Mr1) {
        if (Mp1.size() != Mp.size() || Mr1.size() != Mr.size())
                cout << "Invalid assignment of states " << endl;
        else {
                for (int i= 0; i < Mp.size(); i++) {
                        if (Mp[i] != Mp1[i] || Mr[i] != Mr1[i]) {
                                Mp[i] = Mp1[i];
                                if (Mp1[i] < 0)
                                 Mp[i] = 0;

                                if (Mp[i] > 0 && Mr1[i]>=0)
                                        Mr[i] = Mr1[i];
```

```
                        else
                                Mr[i] = 0;

                }

        }

        //return *this;
}

void PetriNet::changeState(const intvec& um, const int& z, int& delta) {

        /* changes the state of the Petri Net by firing the zth-1 transition
        of the vector of active transitions um. Also returns the incremental time "delta"
        resulting from the transition firing*/

                intvec itk;  //vector of tokens added by a transition. incoming tokens to place
                intvec otk;    //vector of tokens removed by a transition
                itk = Cmin[um[z]];
                otk = Cplus[um[z]];


                int i=0;

                bool active = true;

                for (i=0;i<Cmin[um[z]].size();i++) {

                        if (Mp[Cmin[um[z]][i]] < 1) {

                        active = false;
```

```
        break;
        }
}

if (active == true) {

        delta = 0;
        i=0;

        while (i<itk.size() || i<otk.size()) {

                if (i < itk.size()) {
                        Mp[itk[i]] = Mp [itk[i]] - 1;
                        //removes one token

                        if (Mr[itk[i]] > delta)
                                delta = Mr[itk[i]];
                        //picks the maximum of all input places of transition z
                }

        if (i < otk.size())
                Mp[otk[i]] = Mp [otk[i]] + 1;
                //adds one token

        i++;
        }

        //*************************************************
        //Procedure to find delta = next transition firing
        //************************************************* delta found

        //Ecuation for Mr: Mr(k+1) = Mr(k) - Delta (k) + TC+u(k)
```

```
                        //s.t. Mr(k) >=0
                        // T diagonal matrix which Tii = PT[i] if otk[i]!=0
                        // This matrix brings in the process times of the outgoing
                        //tokens from the last transition firing

                        for (i=0; i< Mr.size();i++) { //removes delta from Mr

                                Mr[i] = Mr[i] - delta;
                                if (Mr[i] < 0) {
                                        Mr[i]=0;

                                }
                        }

                        for (i=0; i< otk.size();i++)
                                Mr[otk[i]]= Mr[otk[i]] + PT[otk[i]];
                                //adds process times

                } //active = true

                else {

                        //cout << "Invalid Transition firing \n";

                }


}

void PetriNet::changearrivals(int w, int z, int tnow, deque<int> t, PetriNet& pn) {

        Mp[w] = Mp[w] + 1;

        for (int j=0; j< Mr.size();j++) {
```

```
                Mr[j] = Mr[j] + t[z-1] - tnow;
                if (Mr[j] < 0) {
                        Mr[j]=0;
                }
        }
}

void PetriNet::changeMachineBreakDown(const intvec& um, const int& z, int& delta) {
        /* changes the state of the Petri Net by firing the zth-1 transition
        of the vector of active transitions um. Also returns the incremental time "delta"
        resulting from the transition firing*/

                intvec itk;  //vector of tokens added by a transition. incoming tokens to place
                intvec otk;    //vector of tokens removed by a transition
                itk = Cmin[um[z]];
                otk = Cplus[um[z]];

                int i=0;

                bool active = true;

                for (i=0;i<Cmin[um[z]].size();i++) {

                        if (Mp[Cmin[um[z]][i]] < 1) {

                        active = false;
                        break;
                        }
                }

                if (active == true) {

                        i=0;
```

```
while (i<itk.size() || i<otk.size()) {

        if (i < itk.size()) {
                Mp[itk[i]] = Mp [itk[i]] - 1; //removes one token
                Mr[itk[i]] = 0;
                                        }

if (i < otk.size())
        Mp[otk[i]] = Mp [otk[i]] + 1; //adds one token
i++;
}


//*************************************************
//Procedure to find delta = next transition firing
//************************************************* delta found

//Ecuation for Mr: Mr(k+1) = Mr(k) - Delta (k) + TC+u(k)

        //s.t. Mr(k) >=0
        // T diagonal matrix which Tii = PT[i] if otk[i]!=0
        // This matrix brings in the process times of the outgoing
        //tokens from the last transition firing

        for (i=0; i< Mr.size();i++) { //removes delta from Mr

                        Mr[i] = Mr[i] - delta;
                        if (Mr[i] < 0) {
                                Mr[i]=0;

                        }
        }
```

```
                        for (i=0; i< otk.size();i++){
                                Mr[otk[i]]= Mr[otk[i]] + PT[otk[i]];//adds process times
                        }
                } //active = true

                else {

                        //cout << "Invalid Transition firing \n";

                }
                fclose(a);

}

void PetriNet::changePT(int p, int q, PetriNet& pn){
        PT[q] = p;

}

void PetriNet::changeStateR(deque<int>& u_activo, const int& r, int& delta) {


        /* changes the state of the Petri Net by firing the Rth-1 transition
        of the vector of active transitions um. Also returns the incremental time "delta"
        resulting from the transition firing*/

                intvec itk;  //vector of tokens added by a transition. incoming tokens to place
                intvec otk;    //vector of tokens removed by a transition
                itk = Cmin[u_activo[r]];
                otk = Cplus[u_activo[r]];

                int i=0;
```

```
bool active = true;


for (i=0;i<Cmin[u_activo[r]].size();i++) {


        if (Mp[Cmin[u_activo[r]][i]] < 1) {


        active = false;

        break;

        }

}


if (active == true) {


        delta = 0;

        i=0;


        while (i<itk.size() || i<otk.size()) {


                if (i < itk.size()) {

                        if(Mp [itk[i]]>0){

                                Mp[itk[i]] = Mp [itk[i]] - 1; //removes one token

                        }

                        if (Mr[itk[i]] > delta)

                                delta = Mr[itk[i]];

                                //picks the maximum of all input places of
                                //transition z

                        }



                if (i < otk.size()) {

                        if(Mp [otk[i]]<1){

                        Mp[otk[i]] = /*Mp [otk[i]] */+ 1; //adds one token

                        }

                }
```

```
                i++;
            }


            //*************************************************
            //Procedure to find delta = next transition firing
            //************************************************* delta found

            //Ecuation for Mr: Mr(k+1) = Mr(k) - Delta (k) + TC+u(k)

                //s.t. Mr(k) >=0
                // T diagonal matrix which Tii = PT[i] if otk[i]!=0
                // This matrix brings in the process times of the outgoing
                //tokens from the last transition firing

                for (i=0; i< Mr.size();i++) { //removes delta from Mr

                        Mr[i] = Mr[i] - delta;
                        if (Mr[i] < 0) {
                                Mr[i]=0;
                        }
                }

                for (i=0; i< otk.size();i++)
                        Mr[otk[i]]= Mr[otk[i]] + PT[otk[i]];//adds process times

        } //active = true

        else {
                //cout << "Invalid Transition firing \n";
        }

}
```

```
//*****************************************************************************************
//Friend Functions
//*****************************************************************************************
void readPn (istream& is, PetriNet& pn) {

        vector<int>::iterator it;

        int r,c,i=0 ;

        pn.clear();
        matrix C;
        exception e;

        try {

                is >> r >> c;
                C.readMatrix(is,r,c);

                pn.M0.resize(r);

                pn.Mr.resize(r);
                pn.PT.resize(r);
                pn.rwt.resize(r);
                pn.p_id.resize(r);
                read_vector(is, pn.M0);

                it = min_element(pn.M0.begin(), pn.M0.end());
                if (*it < 0)
                        throw e;
                pn.Mp=pn.M0;
```

```
        read_vector(is, pn.Mr);
        for (i= 0; i < pn.Mp.size(); i++) {

                if (pn.Mp[i] = 0 && pn.Mr[i] >0)

                        pn.Mr[i] = 0;
                else if (pn.Mr[i] < 0)
                        throw e;
        }

        read_vector(is, pn.PT);
        it = min_element(pn.PT.begin(), pn.PT.end());
        if (*it < 0)
                throw e;


        read_vector(is, pn.rwt);

        it = min_element(pn.rwt.begin(), pn.rwt.end());
        if (*it < 0)
                throw e;

        read_vector(is,pn.p_id);

}

        catch (exception& e) {
                //cout << e.what() << endl;
                //System ("pause");
                exit(1);

}
```

```
        convertMatrix(C,pn.Cmin,pn.Cplus);

}

void PetriNet::printState (ostream& os) {

        if (os ){
                os << "Mp " << Mp << endl;
                os << "Mr " << Mr << endl;
                os << endl;
        }
        else
                cout << "Invalid Output stream" << endl;
}

void convertMatrix (const matrix& C, vec2dim& subs_Cmin, vec2dim& subs_Cplus) {

        //converts the incidence matrix C
        //into two sparse-vectors Cmin and Cplus with the positions on the array
        //each vector of the vec2dim has the indices of a COCUMN

        int Crows = C.getrows();
        int Ccols = C.getcols();
        assert(Crows>0 && Ccols>0);
        subs_Cmin.resize(0);
        subs_Cplus.resize(0);

        for (int j=0;j<Ccols;j++) {

                subs_Cmin.resize(j+1);
                subs_Cplus.resize(j+1);

                for (int i=0;i<Crows;i++)       {
```

```
                        if (C(i,j) == -1)

                                subs_Cmin[j].push_back(i);

                        else if (C(i,j) == 1)

                                subs_Cplus[j].push_back(i);
                }

                if (subs_Cmin[j].size() == 0)
                subs_Cmin.resize(j-1);
                if (subs_Cmin[j].size() == 0)
                subs_Cplus.resize(j-1);
        }

}

void constructU (const PetriNet& pn, intvec& um){

        /*detects active transitions
        um is a vector with the subscipts of the active transitions
        Cmin = C- in compact form
        Mp current marking*/

        int i,j,utemp;
        int cols =pn.Cmin.size();
        ////cout<<"\ncols: "<<cols;
        um.resize(0);
        for (j=0;j<cols;j++) {
                utemp = 1;
                for (i=0;i<pn.Cmin[j].size();i++) {
```

```
                        if (pn.Mp[pn.Cmin[j][i]] < 1) {

                        utemp=0;
                        break;
                        }
                        if(pn.Cplus[j].size() == 2){
                                if(pn.PT[pn.Cplus[j][1]]>0 && pn.p_id[pn.Cplus[j][1]]<0){
                                        utemp=0;
                                        break;
                                }
                        }

                }

                if (utemp == 1)
                        um.push_back(j);
                }

        return ;

}

int sptTime(PetriNet pn, deque<int>& u){

        int delta;
        int time;
        time =0;// time +delta;
        u.clear();

        intvec um; //vector of enabled transitions

        constructU(pn, um);//find enabled transitions
        for (int i = 0 ; i < um.size() ; i++ ){
```

```
        }
        int z;

        while( !um.empty() )  {

                z = findnextmintimeH(um,pn.getPT(),pn);
                u.push_back(um[z]);
                pn.changeState(um,z,delta);
                time = time +delta;
                constructU(pn,um);
                for (int i = 0 ; i < um.size() ; i++ ){
                }
        }

        return time;

}

void convertToPN(vector<Tjob>& jobArray, vector<WkCenter>& wkcArray, PetriNet& pn, intvec& Mf,
vec2dim& transitions , int& Pavg, intvec& PTj) {

        //Converts the a Lekin representation of Lekin into a Petri Net

        int njobs = jobArray.size();
        int nStations = wkcArray.size();
        int ind = 0;
        int sumPT = 0;
        int operationCounter = 0;

        vec2dim ptm, route, rwtm;
        vec2dim endTransition;

        //vec2dim indices;
```

```
        /*ptm: matriz de tiempos
        route: matriz de rutas
        rwtm: matriz de tiempos acumulados
        */
        // cambia las dimensiones de las matrices
        ptm.resize(njobs);
        route.resize(njobs);
        rwtm.resize(njobs);
        transitions.resize(njobs);
        endTransition.resize(nStations);
        PTj.resize(njobs);
        for (int i=0; i< njobs; i++) {
        // el número de filas va a ser = al # de operaciones del trabajo i

                int numOp = jobArray[i].getNOp();
                int sumPTj = 0;
                ptm[i].resize(numOp);
                route[i].resize(numOp);
                rwtm[i].resize(numOp);
                transitions[i].resize(numOp);

                for (int j = 0; j < numOp; j++) {
            //llena la matriz de ruta y de tiempos con sus respectivos datos
                        route[i][j] = jobArray[i].getRoute_i(j);// getRoute_i retorna la
MAQUINA que preocesa la operacion i-esima
                        // una ruta es unas secuencia ordenada de maquinas
                        ptm[i][j] = jobArray[i].getOpTimes_i(j);
                        sumPTj = sumPTj + ptm[i][j];
                }
                PTj[i] = sumPTj;//Tiempo de procesamiento del trabajo J
        }

        for (int i=0; i< njobs; i++) {
                int numOp = jobArray[i].getNOp();
```

```
        int ptacum = 0;
        for (int j = 0; j < numOp; j++) {
                //llena la matriz de tiempo acumulado
                ptacum = ptacum + ptm[i][numOp - j - 1];
                sumPT += ptm[i][numOp - j - 1];
                rwtm[i][numOp - j - 1] = ptacum;
                operationCounter += 1;
        }
}

Pavg = sumPT / operationCounter;

int n = 0;
int m = 0;
int machines;

for (int i=0; i < njobs; i++) {

        int numOp = jobArray[i].getNOp(); //number of operations for job i
        int numMchSt;
        // number of machines in workstation
        machines = 0;
        for (int j = 0; j < numOp; j++) {
        //halla el numero total de plazas y transitions pero sin contar las plazas de
        //trabajo finalizado
                numMchSt = wkcArray[route[i][j]].getNumMch();
                n = n + (1 + numMchSt);
                m = m + 3 * numMchSt;//Número de transiciones
                machines = machines + numMchSt;
        }
}

n = n + machines + njobs;  //add end places
```

```
int nOP = n; //number of operational places //esta variable parece que no la usa

intvec wkmach; //vector of indices of resource places in marking vector


m = m+(machines);


for (int j =0; j< nStations; j++){


        wkmach.push_back(n);

        n = n + wkcArray[j].getNumMch();


}//add resource places


m = m - machines;


//Se generan las transitions correspondientes a la habilitación de las máquinas luego
//de la falla

for(int j=0; j<nStations; j++){

        int numMach = wkcArray[j].getNumMch();

        endTransition[j].resize(numMach);

        for(int k=0;k<numMach;k++){

                endTransition[j][k] = m;

                m = m+1;

        }

}


pn.clear();

Mf.clear();

 //Cmin y Cplus son vectores de vectores = matriz

 /*The j-th (j = 1 to number of transitions) vector of Cmin contains the numbers of
the places which are INPUTS to the j-th transition. The position 0 of the j-th vector
is always an operational place and the remaining positions are resource places*/

pn.Cmin.resize(m);

pn.Cplus.resize(m);

pn.Mp.resize(n);
```

```
pn.M0.resize(n);
pn.Mr.resize(n);
pn.p_id.resize(n);
pn.PT.resize(n);
pn.rwt.resize(n);
Mf.resize(n);

Mf.resize(n);

m = m - machines;


int trIndex =0; // transition index
int plIndex =0; // place index

for (int i = 0; i < njobs; i++) {
        int numOp = jobArray[i].getNOp(); //number of operations for job i
        pn.M0[plIndex] = 1;// la plaza 0 le asigna un token

        for (int j = 0; j < numOp; j++) {
                int numMchSt = wkcArray[route[i][j]].getNumMch();// # mch en wc
                int wcenter = wkmach[route[i][j]];
                int plBuf0 = plIndex;                 //initial buffer place
                int plBuf1 = plIndex + numMchSt + 1; //end buffer place

                pn.PT[plIndex] = 0;// PT = vector de tiempo de proceso
                pn.rwt[plIndex] = rwtm[i][j];

                for (int k= 0; k < numMchSt; k++) {

                        int trbuf0 = endTransition[route[i][j]][k];

                        pn.Cmin[trIndex].resize(2);
```

```
        pn.Cplus[trIndex].resize(1);
        pn.Cmin[trIndex][0] = plBuf0;
        pn.Cmin[trIndex][1] = wkmach[route[i][j]] + k- machines;
        pn.p_id[plIndex] = i;

        plIndex ++;

        pn.PT[plIndex] = ptm[i][j];
        pn.p_id[plIndex] = i;
        pn.rwt[plIndex] = rwtm[i][j] - ptm[i][j];

        pn.Cplus[trIndex][0] =  plIndex;
        trIndex++;

        pn.Cmin[trIndex].resize(1);
        pn.Cplus[trIndex].resize(2);
        pn.Cmin[trIndex][0] = plIndex;
        pn.Cplus[trIndex][0] = plBuf1;
        pn.Cplus[trIndex][1] = wkmach[route[i][j]] + k-machines;

        trIndex++;

        pn.Cmin[trIndex].push_back(plIndex);
        pn.Cplus[trIndex].push_back(plBuf0);
        pn.Cplus[trIndex].push_back(wkmach[route[i][j]] + k);
        pn.Cmin[trbuf0].push_back(wkmach[route[i][j]] + k);
        pn.Cplus[trbuf0].push_back(wkmach[route[i][j]] + k-machines);

        trIndex++;

}
transitions[i][j] = trIndex;
plIndex = plBuf1;
```

```
            }

            pn.PT[plIndex] = 0;
            pn.p_id[plIndex] = i;
            pn.rwt[plIndex] = 0;
            Mf[plIndex] = 1;
            plIndex++;

        }

        int machindex = plIndex;

        for (int i=machindex; i< n-machines; i++){
            pn.M0[i]= 1;
            pn.p_id[i]= machindex-i -1;
            Mf[i] = 1;
            plIndex++;
        }

        machindex = machindex + machines;

        for (int i=machindex; i< n; i++){
            pn.M0[i]= 0;
            pn.PT[plIndex] = 100000;
            pn.p_id[i]= machindex-i -1 - machines;
            Mf[i] = 0; //Marcación final ????
            plIndex++;

        }

        return;
}
```

```
int ActSched(PetriNet pn, deque<int>& u, const vector<Tjob>& jobArray, vec2dim& transitions,
vector<intvec>& marking, intvec& timeU, intvec& jobsU, intvec& operationsU, intvec& machinesU,
intvec& ptU, vec2dim& operEndTime, intvec& PTj, double& initialObjective, intvec& pf_id,
intvec& chrmCoded, const int& objective, double& PrMakespan, double& PrtotalWeightedTardiness,
int& rule, int& ActTTP) {


        u.clear();

        timeU.clear();


        int delta=0;

        int time = 0;

        int nJobs = jobArray.size();

        int z= 0;

        int endtime;

        int sumWjTj = 0;

        int Dj;

        int Wj;

        int NoJob_t = 0;

        intvec Cj;

        intvec jobTimes;


        Cj.resize(nJobs);

        jobTimes.resize(nJobs);


        intvec um, ump;

        //vector of enabled transitions

        intvec endingTransitions;

        //Transiciones de finalización de operaciones


        vec2dim assignments;


        chrmCoded.clear();
```

```
operEndTime.resize(nJobs);
//Matriz con los tiempos de finalización de las operaciones

for(int i = 0 ; i< nJobs ; i++){
        int numOp = jobArray[i].getNOp();
        operEndTime[i].resize(numOp);
        for(int j = 0; j <numOp; j++){
                operEndTime[i][j] = -1;
        }
}

constructU(pn, um);//find enabled transitions

constructUmp(pn, um, ump);

while( !um.empty() ) {
        int index;
        int pt;
        int oper;
        int numJob;
        int endingOperationTime;
        //Tiempo de finalización de la operación
        int maxiumMinimalEndingOperationTime = 10000000;
        //Mínimo tiempo de finalización
        int machine;
        int cont = 0;

        intvec enabledTransitions;
        //Transiciones que habilitan operaciones disponibles
        intvec endingOperationTimes;
        //Tiempos de finalización de las operaciones disponibles
        intvec enabledTransitions2;
        intvec endingOperationTimes2;
```

```
intvec allMachines;
//Máquinas disponibles
intvec Machines;
//Máquinas por asignar
intvec MachinesEndingTimes;
intvec enabledTransitionsMachine;
//Transiciones que habilitan la máquina
intvec G;
//Transiciones que inicializan operaciones con tiempo de inicio menor al mínimo
//tiempo de finalización
intvec startTimes;//Tiempos de inicio de G


endingTransitions.clear();
assignments.clear();
allMachines.clear();
Machines.clear();
MachinesEndingTimes.clear();


        for(int i = 0; i < um.size(); i++){
                index = pn.getCplus()[um[i]][0];
                pt = pn.getPT()[index];
                numJob = jobNo(pn,um[i]);
                jobOperationIndex(um, transitions, oper, i, numJob);
                if(pt == 0){
                        endingTransitions.push_back(um[i]);
                }

                if(time >= operEndTime[numJob][oper]){
                        enabledTransitions.push_back(um[i]);
                        if(oper>0){// oper 1 y 2
                                if(pt == 0){ //transitions de salida
                                        cont++;

                endingOperationTime = operEndTime[numJob][oper];
```

```
                        }else{
                                endingOperationTime =
                                operEndTime[numJob][oper-1] + pt;
                        }
                }else{// oper 0
                        if(pt == 0){
                                cont++;
                                        endingOperationTime =
operEndTime[numJob][oper];
                        }else{
                                endingOperationTime = time + pt;
                        }//end else
                }//end else
                endingOperationTimes.push_back(endingOperationTime);
        }
}//end for

for(int i = 0; i < enabledTransitions.size(); i++){
         enabledTransitions2.push_back(enabledTransitions[i]);
         endingOperationTimes2.push_back(endingOperationTimes[i]);
}

if(enabledTransitions.size()>0){
        //Se organizan las transiciones de acuerdo al mayor tiempo de
        //teminación
        int cont1 =0;
        int pos;
        do{
                maxiumMinimalEndingOperationTime = 0;
                for(int i = 0 ; i < enabledTransitions2.size() ; i++ ){
                        if(endingOperationTimes2[i] >=
                        maxiumMinimalEndingOperationTime){
                        pos = i;
                        maxiumMinimalEndingOperationTime =
                        endingOperationTimes2[i];
                        }
```

```
                }


                enabledTransitions[cont1] = enabledTransitions2[pos];


                endingOperationTimes[cont1] =
                endingOperationTimes2[pos];


                enabledTransitions2.erase(enabledTransitions2.begin() +
                pos);

                endingOperationTimes2.erase(
                endingOperationTimes2.begin() + pos);


                cont1 = cont1 +1;

        }while(cont1 < enabledTransitions.size());


        maxiumMinimalEndingOperationTime = endingOperationTimes[0];

}



if(enabledTransitions.size()>0  && cont == 0){

//No hay transiciones de salida

                for(int i = 0; i < enabledTransitions.size(); i++){


                        machine = allMchNo(pn,enabledTransitions[i]);

                        allMachines.push_back(machine);

                }


                Machines.push_back(allMachines[0]);

                MachinesEndingTimes.push_back(endingOperationTimes[0]);

                allMachines.erase(allMachines.begin() + 0);

                endingOperationTimes.erase(endingOperationTimes.begin()
                + 0);

                if(allMachines.size() > 0){

                //Se borran las máquinas repetidas

                        int cont2 = 0;
```

```
                do{
                        machine = allMachines[0];
                        endingOperationTime =
                        endingOperationTimes[0];
                        for(int i= 0; i<Machines.size(); i++){
                                if(Machines[i]==machine){
                                        cont2++;
                                }
                        }

                        if(cont2==0){
                                Machines.push_back(machine);
MachinesEndingTimes.push_back(endingOperationTime);
                        }

                        allMachines.erase(allMachines.begin() +
                        0);
endingOperationTimes.erase(endingOperationTimes.begin() + 0);

                        }while(allMachines.size() > 0);
                }

                assignments.resize(Machines.size());

                for(int i=0; i<Machines.size();i++){
                        assignments[i].resize(6);
                }

                for(int i=0; i<assignments.size(); i++){
                        assignments[i][0] = Machines[i];
```

```
        //Se asigna un lugar de la matriz para cada
        //máquina disponible para ser asignada

        assignments[i][5] = -1;

        //Se le asigna el valor -1 para indicar que no ha
        //sido asignada

}


for(int i=0; i<Machines.size(); i++){

        enabledTransitionsMachine.clear();

        for(int j =0; j < pn.getCmin().size(); j++){

                if(pn.getCmin()[j].size()>1 &&
                Machines[i] == pn.getCmin()[j][1]){

                enabledTransitionsMachine.push_back(j);

                }

        }



        G.clear();

        startTimes.clear();


        int Mini1=100000;

        int TransMini1;

        int startTime1;

        int startTime2;

        int endingTime1;

        int pos1;


                for(int j =0;
                j<enabledTransitionsMachine.size(); j++){

                        index = pn.getCplus()
                        [enabledTransitionsMachine[j]][0]
                        ;

                        pt = pn.getPT()[index];

                        numJob =
                        jobNo(pn,enabledTransitionsMachin
                        e[j]);
```

```
        jobOperationIndex(enabledTransitionsMachine, transitions, oper, j, numJob);

                                                        int OperaciónReservada = 0;

                                                        for(int k=0;
k<assignments.size(); k++){
                                                                if(assignments[k][5] ==
1){
        if(assignments[k][2] == numJob){
        if(assignments[k][3] == oper){
        OperaciónReservada = OperaciónReservada + 1;
                                                                                        }
                                                                        }
                                                                }
                                                        }

                                                        if(OperaciónReservada == 0){
                                                                if(oper>0){


        if(operEndTime[numJob][oper]<0 || operEndTime[numJob][oper]==0){
                                                                                        startTime1
= operEndTime[numJob][oper-1];
                                                                                        endingTime1
= startTime1 + pt;
                                                                        }


        if(operEndTime[numJob][oper]<0 && startTime1 < maxiumMinimalEndingOperationTime &&
operEndTime[numJob][oper-1]>=0){

        G.push_back(enabledTransitionsMachine[j]);
```

```
        startTimes.push_back(startTime1);
                                                                        }

                                                                        //El tiempo de
finalización es -1 para las operaciones anteriores que no han sido inicializadas
                                                                    }else{//Operaciones
iniciales


        if(operEndTime[numJob][oper]>0){
                                                                        }else{
                                                                                startTime1
= 0;
                                                                                endingTime1
= startTime1 + pt;
        if(operEndTime[numJob][oper]<0 && startTime1 < maxiumMinimalEndingOperationTime){
        G.push_back(enabledTransitionsMachine[j]);
        startTimes.push_back(startTime1);
                                                                                }
                                                                        }
                                                                    }
                                                            }
                                                        }
                                                    int MachineRestartTime = 0;


                                                    if(G.size()>0){
                                                    int TransMini2;


                                                    if(rule == 1){
                                                            TransMini2 =
                                                            G[findnextmintime(G,pn.getPT(),pn)];
                                                    }else if(rule == 2){
                                                            TransMini2 =
                                                            G[findnextmaxtime(G,pn.getPT(),pn)];
```

```
}else if(rule == 3){

        TransMini2 =
        G[findnextmaxtime(G,pn.getrwt(),pn)];

}else if(rule == 4){

        TransMini2 =
        G[EDD(G,pn.getPT(),pn,jobArray,MachineRes
        tartTime)];

}else if(rule == 5){

        TransMini2 =
        G[CR(G,pn.getPT(),pn,jobArray, pf_id,
        time, PTj, jobTimes)];

}else if(rule == 6){

        TransMini2 =
        G[ATC(G,pn.getPT(),pn,jobArray, pf_id,
        time, PTj, jobTimes)];//ATCCCCC

}else if(rule == 7){

        TransMini2 =
        G[MSLACK(G,pn.getPT(),pn,jobArray, pf_id,
        time, PTj, jobTimes)];//ATCCCCC

}



int startTime3;


for(int j =0; j<G.size(); j++){

//Se busca el tiempo de inicio de la transición u
//operación en startTimes

        if(TransMini2==G[j]){

                startTime3 = startTimes[j];

        }

}


assignments[i][1] = TransMini2;

assignments[i][4] = startTime3;

assignments[i][5] = 1;

numJob = jobNo(pn,TransMini2);

assignments[i][2] = numJob;
```

```
                int pos2;

                for(int k=0; k<enabledTransitionsMachine.size();
k++){
                        if(TransMini2 ==
enabledTransitionsMachine[k]){
                                pos2 = k;
                        }
                }
                jobOperationIndex(enabledTransitionsMachine,
transitions, oper, pos2, numJob);
                assignments[i][3] = oper;
                }
        }

        int cont8=0;
        int cont9=0;
        int pos3;

        for(int i =0; i<assignments.size(); i++){
                if(assignments[i][5]==1){
                        cont9+=1;
                        pos3 = i;
                }
        }

        if(assignments.size() == 1){
                for(int i=0; i<um.size(); i++){
                        if(assignments[0][1] == um[i]){
                                cont8 +=1;
                                z = i;
                        }
                }
```

```
if(cont8 == 0){//Si la transición no está habilitada
	for(int j =0; j<pn.getCplus().size(); j++){
		if(pn.getCplus()[j][0] ==
		pn.getCmin()[assignments[0][1]][0]){
				for(int k=0; k<um.size();
				k++){
					if(j == um[k]){
						z = k;
					}
				}
			}
		}
}

}else if(cont9 ==1){
//Solo hay una máquina (Tamaño de assignments es 1)
	for(int i=0; i<um.size(); i++){
		if(assignments[pos3][1] == um[i]){
			cont8 +=1;
			z = i;
		}
	}

	if(cont8 == 0){
	//La transición no está habilitada
		for(int j =0; j<pn.getCplus().size();
		j++){
			if(pn.getCplus()[j][0] ==
		pn.getCmin()[assignments[pos3][1]][0]){
				for(int k=0; k<um.size();
				k++){
					if(j == um[k]){
						z = k;
					}
```

```
                                        }
                                    }
                                }
                            }

                            }else{// Hay más de una máquina para asignar

                            int minimotiempodeinicio = 100000;
                            int minimo2;

                            for(int i=0; i<assignments.size(); i++){
                                if(assignments[i][5] == 1){
        if(assignments[i][4]<minimotiempodeinicio){
                                        minimotiempodeinicio =
assignments[i][4];
                                        minimo2 = i;
                                    }
                                }

                            }

                        int cont7=0;

                            for(int i=0; i<um.size(); i++){
                                if(assignments[minimo2][1] == um[i]){
                                    cont7 +=1;
                                    z = i;
                                }
                            }

                            if(cont7 == 0){
                                for(int j =0; j<pn.getCplus().size(); j++){
```

```
                                                        if(pn.getCplus()[j][0] ==
pn.getCmin()[assignments[minimo2][1]][0]){
                                                                for(int k=0; k<um.size();
k++){
                                                                        if(j == um[k]){
                                                                                z = k;
                                                                        }
                                                                }
                                                        }
                                                }
                                        }
                                }//Fin assignments


                        }else{
                                z = findnextmintime(um,pn.getPT(),pn);


                        }



                int machine1;
                intvec otherTransitions;
                intvec otherTransitionsEndingTimes;
                intvec otherTransitionsMachines;
                for (int i = 0 ; i < um.size() ; i++ ){
                        index = pn.getCplus()[um[i]][0];
                        pt = pn.getPT()[index];
                        if(pt != 0){
                                machine1 = allMchNo(pn,um[i]);
                                int cont2 =0;
                                for(int j=0; j<assignments.size(); j++){
                                        if(machine1 == assignments[j][0]){
                                                cont2 = cont2 +1;
                                        }
                                }
```

```
                        if(cont2 == 0){
                                otherTransitions.push_back(um[i]);
                                otherTransitionsEndingTimes.push_back(time + pt);
                                otherTransitionsMachines.push_back(machine1);
                        }
                }

        }//Impresión transitions habilitada

        int newMinTime= 100000;
        int tempPos;

        if(otherTransitions.size()>0){
                for(int i=0; i<otherTransitions.size(); i++){
                        if(otherTransitionsEndingTimes[i] < newMinTime){
                                newMinTime = otherTransitionsEndingTimes[i];
                                tempPos = i;
                        }
                }

        }

        //Si de las operaciones que se encuentran en proceso alguna termina antes de la
transición seleccionada, se dispara primero la transición de finalización, se enceuentra a
continuación:
        int cont15 =0;
        int FinalTransition;
        index = pn.getCplus()[um[z]][0];
        pt = pn.getPT()[index];
        if(pt == 0){
                for(int j=0; j<u.size(); j++){
                        if(u[j] == um[z]-1){
                                index = pn.getCplus()[u[j]][0];
                                pt = pn.getPT()[index];
```

```
                                endtime = timeU[j] + pt;
                                for(int i = 0 ; i < endingTransitions.size() ; i++ ){
                                        for(int k=0; k<u.size(); k++){
                                                if(u[k] == endingTransitions[i]-1){
                                                        index = pn.getCplus()[u[k]][0];
                                                        pt = pn.getPT()[index];
                                                        if(timeU[k]+pt<endtime){
                                                                endtime = timeU[k]+pt;
                                                                FinalTransition =
                                                                endingTransitions[i];
                                                                cont15 = cont15 +1;
                                                        }
                                                }
                                        }
                                }
                        }
                }

                if(cont15 >0){//Se encuentra la posición en um
                        for(int j=0; j<um.size(); j++){
                                if(um[j] == FinalTransition){
                                        z = j;
                                }
                        }
                }
        }

        for (int i = 0 ; i < endingTransitions.size() ; i++ ){
                for(int j=0; j<u.size(); j++){
                        if(u[j] == endingTransitions[i]-1){
                                index = pn.getCplus()[u[j]][0];
                                pt = pn.getPT()[index];
                                endtime = timeU[j] + pt;
                        }
```

```
        }
}


        u.push_back(um[z]);
        int pl_id;
        for (int i=0;i<pn.getCmin()[um[z]].size();i++) {
                pl_id = pn.getp_id()[pn.getCmin()[um[z]][i]];
                if(pl_id<0){
                        pl_id = -pl_id;
                }
        }



        for (int i=0;i<pn.getCplus()[um[z]].size();i++) {
                pl_id=pn.getp_id()[pn.getCplus()[um[z]][i]];
                if (pl_id < 0){
                        pl_id=-pl_id;
                }
        }


        int number = rand() % pn.getCmin().size();


        int meme = (number*um.size()) + z;


        chrmCoded.push_back(meme);


        pn.changeState(um,z,delta);
        numJob = jobNo(pn,um[z]);
        jobOperationIndex(um, transitions, oper, z, numJob);
        index = pn.getCplus()[um[z]][0];
        pt = pn.getPT()[index];
        time = time + delta;
```

```
jobTimes[numJob] = jobTimes[numJob] + pt;

jobsU.push_back(numJob+1);
operationsU.push_back(oper+1);
ptU.push_back(pt);


if(pt>0){
        machine = allMchNo(pn,um[z]);
        machinesU.push_back(machine);
}else{
        machinesU.push_back(0);
}



intvec marking1;
marking1 = pn.getMp();
marking1.push_back(time);
marking.push_back(marking1);

timeU.push_back(time);

operEndTime[numJob][oper] = time + pt;

if (pn.getCplus()[um[z]][0] == pf_id[numJob]) {//Evaluar objetivo
        Cj[numJob] = time;
        Wj = jobArray[pn.getp_id()[pn.getCmin()[um[z]][0]]].getWeight();
        Dj = jobArray[pn.getp_id()[pn.getCmin()[um[z]][0]]].getDue();
        if ( (Cj[numJob] - Dj) >= 0){
                sumWjTj = sumWjTj + Wj * (Cj[numJob] - Dj);
                        NoJob_t = NoJob_t + 1;
```

```
                        }
                }//Fin evaluar objetivo



                constructU(pn,um);

                constructUmp(pn, um, ump);
        }//End while principal


        //Se guardan los valores objetivos
        if(objective == 8){
                initialObjective = time;
        }
        else if(objective == 7){
                initialObjective = sumWjTj;
        }
        else if(objective == 9){
                initialObjective = (PrMakespan*time) + (PrtotalWeightedTardiness*sumWjTj);
        }


        ActTTP = sumWjTj;


        return time;


}



int InitialSched(PetriNet pn, deque<int>& u, const vector<Tjob>& jobArray, vec2dim&
transitions, vector<intvec>& marking, intvec& timeU, intvec& jobsU, intvec& operationsU,
intvec& machinesU, intvec& ptU, vec2dim& operEndTime, double& initialObjectiveIS, intvec&
pf_id, intvec& uTemporal, const int& objective, double& PrMakespan, double&
PrtotalWeightedTardiness, int& TTPAGIS) {


        //En esta función se actualiza la red de petri de acuerdo con el vector generado en el
AG de la programación inicial
```

```
    timeU.clear();
    marking.clear();

    int delta;
    int time = 0;
int nJobs = jobArray.size();
    int z= 0;
    int endtime;//Tiempo de finalización de la operación
    int sumWjTj = 0;
    int Dj ;
int Wj ;
    int NoJob_t = 0;
    intvec Cj;
    Cj.resize(nJobs);
    intvec um, ump; //vector of enabled transitions



    operEndTime.resize(nJobs);

    for(int i = 0 ; i< nJobs ; i++){
            int numOp = jobArray[i].getNOp();
            operEndTime[i].resize(numOp);
            for(int j = 0; j <numOp; j++){
                    operEndTime[i][j] = -1;
            }
    }

    constructU(pn, um);//find enabled transitions

    constructUmp(pn, um, ump);
```

```
for(int h=0; h<uTemporal.size(); h++ ) {
        int index;
        int pt;
        int oper;
        int numJob;

                for(int i = 0; i < um.size(); i++){
                        if(um[i]==uTemporal[h]){
                                z=i;
                        }
                }

        int TranTemp = um[z];

        u.push_back(um[z]);
        pn.changeState(um,z,delta);

        numJob = jobNo(pn,um[z]);
        jobOperationIndex(um, transitions, oper, z, numJob);
        index = pn.getCplus()[um[z]][0];
        pt = pn.getPT()[index];
        time = time + delta;

        intvec marking1;
        marking1 = pn.getMp();
        marking1.push_back(time);
        marking.push_back(marking1);

        timeU.push_back(time);
        operEndTime[numJob][oper] = time + pt;
        endtime = time + pt;

        if (pn.getCplus()[TranTemp][0] == pf_id[numJob]) {//Evaular objetivo
```

```
                Cj[numJob] = time;
                Wj = jobArray[pn.getp_id()[pn.getCmin()[TranTemp][0]]].getWeight();
                Dj = jobArray[pn.getp_id()[pn.getCmin()[TranTemp][0]]].getDue();

                if ( (Cj[numJob] - Dj) >= 0){
                        sumWjTj = sumWjTj + Wj * (Cj[numJob] - Dj);
                        NoJob_t = NoJob_t + 1;
                }
        }

        jobsU.push_back(numJob+1);
        operationsU.push_back(oper+1);
        ptU.push_back(pt);

        int machine;
        if(pt>0){
                machine = allMchNo(pn,um[z]);
                machinesU.push_back(machine);
        }else{
                machinesU.push_back(0);
        }

        constructU(pn,um);
        constructUmp(pn, um, ump);
}


if(objective == 8){
        initialObjectiveIS = time;
}
else if(objective == 7){
        initialObjectiveIS = sumWjTj;
}
```

```
        else if(objective == 9){
                initialObjectiveIS = (PrMakespan*time) + (PrtotalWeightedTardiness*sumWjTj);
        }

        TTPAGIS = sumWjTj;

        return time;


}

int jobNo(const PetriNet& pn, const int& uz) {
/* Devuelve el número del trabajo al que pertenece
   la transición uz en una PetriNet pn*/

        return pn.p_id[pn.Cmin[uz][0]];
}


int rlsMchNo (const PetriNet& pn, const int& uz){
/* Devuelve el número de la máquina que se libera
al disparar la transición uz en una PetriNet pn.
Si no se libera ninguna máquina entonces, devuelve 0*/

        if (pn.Cplus.size() < 2)

                return 0;
        else
          return pn.Cplus[uz][1];
}

int allMchNo (const PetriNet& pn, const int& uz){
/* Devuelve el número de la máquina que se ocupa
```

```
al disparar la transición uz en una PetriNet pn
Si no se ocupa ninguna máquina entonces, devuelve 0*/


        if (pn.Cmin.size() < 2){

                return 0;
        }else{


        if(pn.Cmin[uz][1] == NULL){
                return 0;
        }else{

          return pn.Cmin[uz][1];

        }
        }
}

int nJobs(const intvec& p_id){

        //determines the number of jobs using the p_id vector that contains the index of the
        //jobs.

        int nj=p_id[0];
        for (int i=0; i< p_id.size();i++) {

                if (p_id[i] > nj)
                        nj = p_id[i];
        }

        nj++;
```

```
        return nj;
}


int nResources(const intvec& p_id) {

        //finds the number of resources in the system.

        int nr=0;
        intvec::const_iterator it1=find(p_id.begin()+1,p_id.end(), -1);
        int i=it1-p_id.begin();
        while (it1!=p_id.end() ) {

                nr ++;
                it1++;

        }
        return nr;
}

void constructUmp (const PetriNet& pn, const intvec& um, intvec& ump) {

        int min = 10000;
        int pindex;
        ump.clear();

        for (int j= 0; j < um.size(); j++) {

                pindex = pn.Cmin[um[j]][0];

                if (min > pn.Mr[pindex]) {
```

```
                        min = pn.Mr[pindex];
                        ump.clear(); //borra
                        ump.push_back(um[j]);
                }

                else if (min == pn.Mr[pindex]) {

                        ump.push_back(um[j]);

                }

        }

}

void cons_pe_id (const PetriNet& pn, const intvec& p_id, intvec& pe_id) {

        pe_id.clear();
        int i=0;
        int temp=0;

        while (pn.getp_id()[i] >= 0) {

                if (pn.getp_id()[i] == temp) {
                temp =temp + 1;
                pe_id.push_back(i);
                }
        i=i+1;
        }
}

void cons_pf_id (const PetriNet& pn, const intvec& p_id, intvec& pf_id) {
```

```
        pf_id.clear();
        int k=0;
        int tempo=1;

        while (pn.getp_id()[k] >= 0) {

                if (pn.getp_id()[k+1] == tempo || pn.getp_id()[k+1] < 0) {
                tempo =tempo + 1;
                pf_id.push_back(k);
                }
                k=k+1;
        }
}

/*****************************************************************************************
*
Funciones Accesorias
/*****************************************************************************************
**/

void jobOperationIndex(const intvec& um ,const vec2dim& transitions, int& oper, int i, int
numJob){

        int j = 0 ;
        bool cont = false;

        do{//encontramos a que operación pertenece la transición

                if(um[i]< transitions[numJob][j]){
                        oper = j;//Inidice de la operacion del trabajo
                        cont = true;
                }

                j++;
```

```
        }while(cont == false  );


}


void jobOperationIndex1(const intvec& um ,const vec2dim& transitions, int& oper, int i, int
numJob){


        int j = 0 ;

        bool cont = false;


}

int findsumremtime (const intvec& Mp, const intvec& Mr, const intvec& rwt,

                                const intvec& busy_places) {


        int ans =0;//

        for (int i=0;i<busy_places.size();i++)


                ans = ans  + (Mr[busy_places[i]] + rwt[busy_places[i]])*Mp[busy_places[i]];


        return ans;

}


int findmaxremtime (const intvec& Mr, const intvec& rwt, const intvec& Mp,

                                const intvec& busy_places) {


        int ans =0;

        for (int i=0;i<busy_places.size();i++) {


                 if ((rwt[busy_places[i]] + Mr[busy_places[i]])> ans)


                 ans = rwt[busy_places[i]] + Mr[busy_places[i]];

        }

        return ans;

}
```

```
int findminremtime (const intvec& Mr, const intvec& rwt,
                                const intvec& busy_places) {

        int ans =rwt[busy_places[0]] + Mr[busy_places[0]];
        for (int i=1;i<busy_places.size();i++) {

                if ((rwt[busy_places[i]] + Mr[busy_places[i]]) < ans)

                ans= (rwt[busy_places[i]] + Mr[busy_places[i]]);
        }
        return ans;
}

int findnextmintime(const intvec& um, const intvec& iv, const PetriNet& pn) {

//Rule to select a transition
//finds the minimum of the remaining job times of output places of the enabled //transitions
//The rule applies as follows:
//(a)If a token in a place has exhausted its process time, it has priority so it releases the
//resources it seized.
//(b) If there are tokens in buffer places (Process Time=0), pick the transition whose output
//place (PT>0 by definition) is minimum.
//(c) If all tokens are in operational places (PT>0) fire the transition whose input place has
//the minimum Mr (Remaining Process Time).

        assert(um.size()>0);
        assert(iv.size()>0);
        assert (pn.getCplus().size()>0);
        assert (pn.getCmin().size()>0);
        assert (pn.getPT().size()>0);

        int z;
        int j=0;
```

```
int pindx;// = pn.Cmin[um[0]][0];

int pnext;

int min=100000;


bool notokeninbuf= false;

//checks if there is at least one token in a buffer place

//bool exhausted = false; //checks if there is at least one token that exhausted its
remaining time to enable its output transition


for(int k=0; k<um.size();k++){


}


while (j<um.size()) {
        pindx = pn.getCmin()[um[j]][0];

        int t2= pn.getp_id()[pindx];


        if (pn.getPT()[pindx] > 0 && pn.getMr()[pindx] == 0) {

                //checks which places exhausted their process time. give priority to
                //those.

                        z=j;

                        break;

        }




        else if (pn.getPT()[pindx] == 0) {

                //from all marked places which PTi=0, pick the one whose next
                //operational place minimizes a criterion

                //given by iv


                pnext = pn.getCplus()[um[j]][0];
```

```
                        if (iv[pnext] < min || notokeninbuf==false) {
                                min = iv[pnext];
                                z=j;
                        }

                        notokeninbuf = true;
                }

                else if ( pn.getPT()[pindx] > 0 && notokeninbuf==false) {
                //from all marked places which PTi>0 and Mri>0, pick
                //the one whose next PT is minimum
                //only works if there are no tokens in buffer places (PTi=0);

                        if (pn.getMr()[pindx] < min) {
                                min = pn.getMr()[pindx];
                                z=j;
                        }
                }

                j++;

        }

        return z;
}

int EDD(const intvec& um, const intvec& iv, const PetriNet& pn, const vector<Tjob>& jobArray,
int MachineRestartTime) {

//Rule to select a transition
//finds the minimum of the remaining job times of output places of the enabled transitions
//The rule applies as follows:
//(a)If a token in a place has exhausted its process time, it has priority so it releases the
//resources it seized.
```

```
//(b) If there are tokens in buffer places (Process Time=0), pick the transition whose output
//place (PT>0 by definition) is minimum.

(c) If all tokens are in operational places (PT>0) fire the transition whose input place has
//the minimum Mr (Remaining Process Time).


        assert(um.size()>0);

        assert(iv.size()>0);

        assert (pn.getCplus().size()>0);

        assert (pn.getCmin().size()>0);

        assert (pn.getPT().size()>0);


        int z;


        int j=0;

        int pindx;// = pn.Cmin[um[0]][0];

        int pnext;

        int min=100000;


        int numJob;

        int Dj;



        bool notokeninbuf= false;

        //checks if there is at least one token in a buffer place

        //bool exhausted = false; //checks if there is at least one token that exhausted its
        //remaining time to enable its output transition


        int MinimalDD = 10000000;


        while (j<um.size()) {

                pindx = pn.getCmin()[um[j]][0];

                numJob = jobNo(pn,um[j]);

                if(numJob >= 0){

                        Dj = jobArray[pn.getp_id()[pn.getCmin()[um[j]][0]]].getDue();
```

```
                }else{
                        Dj = MachineRestartTime;
                }
                if(Dj < MinimalDD){
                                MinimalDD = Dj;
                                z = j;
                        }
                j++;
        }

        return z;

}

int CR(const intvec& um, const intvec& iv, const PetriNet& pn, const vector<Tjob>& jobArray,
intvec& pf_id, int& time, intvec& PTj, intvec& jobTimes) {

//Rule to select a transition
//finds the minimum of the remaining job times of output places of the enabled transitions
//The rule applies as follows:
//(a)If a token in a place has exhausted its process time, it has priority so it releases the
//resources it seized.
//(b) If there are tokens in buffer places (Process Time=0), pick the transition whose output
//place (PT>0 by definition) is minimum.
//(c) If all tokens are in operational places (PT>0) fire the transition whose input place has
//the minimum Mr (Remaining Process Time).

        assert(um.size()>0);
        assert(iv.size()>0);
        assert (pn.getCplus().size()>0);
        assert (pn.getCmin().size()>0);
        assert (pn.getPT().size()>0);

        int z=0;
```

```
int j=0;
int pindx;
int pnext;
int min=100000;


int numJob;
int Dj;



bool notokeninbuf= false;
//checks if there is at least one token in a buffer place
//bool exhausted = false; //checks if there is at least one token that exhausted its
//remaining time to enable its output transition


int MinimalCR = 10000000;
int CRj;
while (j<um.size()) {
        pindx = pn.getCmin()[um[j]][0];
        numJob = jobNo(pn,um[j]);
        if(numJob >= 0){
                Dj = jobArray[pn.getp_id()[pn.getCmin()[um[j]][0]]].getDue();
                if(PTj[numJob] - jobTimes[numJob] == 0){
                        CRj = 0;
                }else{
                        CRj = (Dj - time) / (PTj[numJob] - jobTimes[numJob]);
                }
        }else{
                CRj = 10000000000000;
        }



        if(CRj<MinimalCR){
                MinimalCR = CRj;
                z = j;
```

```
                }
                j++;
        }

                return z;

}

int ATC(const intvec& um, const intvec& iv, const PetriNet& pn, const vector<Tjob>& jobArray,
intvec& pf_id, int& time, intvec& PTj, intvec& jobTimes) {

//Rule to select a transition
//finds the minimum of the remaining job times of output places of the enabled transitions
//The rule applies as follows:
//(a)If a token in a place has exhausted its process time, it has priority so it releases the
//resources it seized.
//(b) If there are tokens in buffer places (Process Time=0), pick the transition whose output
//place (PT>0 by definition) is minimum.
//(c) If all tokens are in operational places (PT>0) fire the transition whose input place has
//the minimum Mr (Remaining Process Time).

        assert(um.size()>0);
        assert(iv.size()>0);
        assert (pn.getCplus().size()>0);
        assert (pn.getCmin().size()>0);
        assert (pn.getPT().size()>0);

        int z=0;
        int j=0;
        int pindx;
        int pnext;
        int min=100000;
        int averagePT;
        int sumPT=0;
```

```
int numJob;

int Dj;



bool notokeninbuf= false;

//checks if there is at least one token in a buffer place

//bool exhausted = false;

//checks if there is at least one token that exhausted its remaining time to enable
//its output transition


int MaximumIj = -1000000;

double Ij;

int k=1;


while (j<um.size()) {

        pindx = pn.getCmin()[um[j]][0];

        numJob = jobNo(pn,um[j]);

        if(numJob >= 0){

                Dj = jobArray[pn.getp_id()[pn.getCmin()[um[j]][0]]].getDue();

                for(int i = 0; i<jobTimes.size(); i++){

                        sumPT = sumPT + (PTj[i] - jobTimes[i]);

                }

                averagePT = sumPT/jobTimes.size();

                if(averagePT == 0){


                        Ij = 0;

                        } else {

                                float value = - max(Dj-(PTj[numJob]-jobTimes[numJob])-
                                time,0)/(k*averagePT);

                                if(PTj[numJob] - jobTimes[numJob] == 0){

                                        Ij = 0;

                                } else {
```

```
                                        Ij = (1/(PTj[numJob] - jobTimes[numJob]))
                                        *exp(value);
                                }
                        }
                }else{
                Ij = 0;
                }
                if(MaximumIj < Ij){
                        MaximumIj = Ij;
                        z = j;
                }
                j++;
        }

        return z;

}

int MSLACK(const intvec& um, const intvec& iv, const PetriNet& pn, const vector<Tjob>&
jobArray, intvec& pf_id, int& time, intvec& PTj, intvec& jobTimes) {

//Rule to select a transition
//finds the minimum of the remaining job times of output places of the enabled transitions
//The rule applies as follows:
//(a)If a token in a place has exhausted its process time, it has priority so it releases the
//resources it seized.
//(b) If there are tokens in buffer places (Process Time=0), pick the transition whose output
//place (PT>0 by definition) is minimum.
//(c) If all tokens are in operational places (PT>0) fire the transition whose input place has
//the minimum Mr (Remaining Process Time).

        assert(um.size()>0);
        assert(iv.size()>0);
        assert (pn.getCplus().size()>0);
        assert (pn.getCmin().size()>0);
```

```
assert (pn.getPT().size()>0);


int z=0;

int j=0;

int pindx;

int pnext;

int min=100000;

int sumPT=0;

int numJob;

int Dj;



bool notokeninbuf= false;

//checks if there is at least one token in a buffer place

//bool exhausted = false; //checks if there is at least one token that exhausted its
//remaining time to enable its output transition


int Minimum = 1000000000;

float MSLACK;

while (j<um.size()) {

        pindx = pn.getCmin()[um[j]][0];

        numJob = jobNo(pn,um[j]);

        if(numJob >= 0){

                Dj = jobArray[pn.getp_id()[pn.getCmin()[um[j]][0]]].getDue();

                MSLACK = max(Dj-(PTj[numJob] - jobTimes[numJob])-time,0);

        }else{

                MSLACK = 1000000000000;

        }

        if(Minimum >= MSLACK){

                Minimum = MSLACK;

                z = j;

        }

        j++;

}
```

```
        return z;


}


int findnextmintime2(const intvec& um, const intvec& iv, const PetriNet& pn, int&
MinTransition) {


//Rule to select a transition

//finds the minimum of the remaining job times of output places of the enabled transitions

//The rule applies as follows:

//(a)If a token in a place has exhausted its process time, it has priority so it releases the
//resources it seized.

//(b) If there are tokens in buffer places (Process Time=0), pick the transition whose output
//place (PT>0 by definition) is minimum.

//(c) If all tokens are in operational places (PT>0) fire the transition whose input place has
//the minimum Mr (Remaining Process Time).


        assert(um.size()>0);

        assert(iv.size()>0);

        assert (pn.getCplus().size()>0);

        assert (pn.getCmin().size()>0);

        assert (pn.getPT().size()>0);



        int z;


        int j=0;

        int pindx;

        int pnext;

        int min=100000;



        bool notokeninbuf= false;

        //checks if there is at least one token in a buffer place
```

```
//bool exhausted = false; //checks if there is at least one token that exhausted its
//remaining time to enable its output transition



while (j<um.size()) {
        pindx = pn.getCmin()[um[j]][0];
        int t2= pn.getp_id()[pindx];

        if (pn.getPT()[pindx] > 0 && pn.getMr()[pindx] == 0) {
        //checks which places exhausted their process time. give priority to those.

                z=pn.getMr()[pindx];
                MinTransition = j;
                break;

        }

        else if (pn.getPT()[pindx] == 0) {
                //from all marked places which PTi=0, pick the one whose next
                //operational place minimizes a criterion given by iv

                pnext = pn.getCplus()[um[j]][0];
                min = 0;
                z = min;
                MinTransition = j;

                notokeninbuf = true;
        }

        else if ( pn.getPT()[pindx] > 0 && notokeninbuf==false) {
        //from all marked places which PTi>0 and Mri>0, pick
        //the one whose next PT is minimum
        //only works if there are no tokens in buffer places (PTi=0);
```

```
                        if (pn.getMr()[pindx] < min) {
                                min = pn.getMr()[pindx];
                                z=min;
                                MinTransition = j;
                        }
                }
                j++;
        }

        return z;

}


int findnextmintimeH(intvec& enabledTransitions, const intvec& iv, const PetriNet& pn) {

//Rule to select a transition
//finds the minimum of the remaining job times of output places of the enabled transitions
//The rule applies as follows:
//(a)If a token in a place has exhausted its process time, it has priority so it releases the
//resources it seized.
//(b) If there are tokens in buffer places (Process Time=0), pick the transition whose output
//place (PT>0 by definition) is minimum.
//(c) If all tokens are in operational places (PT>0) fire the transition whose input place has
//the minimum Mr (Remaining Process Time).

        assert(enabledTransitions.size()>0);
        assert(iv.size()>0);
        assert (pn.getCplus().size()>0);
        assert (pn.getCmin().size()>0);
        assert (pn.getPT().size()>0);

        int z;
        int j=0;
```

```
int pindx;
int pnext;
int min=100000;

bool notokeninbuf= false;
//checks if there is at least one token in a buffer place

while (j<enabledTransitions.size()) {

        pindx = pn.getCmin()[enabledTransitions[j]][0];

        if (pn.getPT()[pindx] > 0 && pn.getMr()[pindx] == 0) {
        //checks which places exhausted their process time. give priority to those.
                z=j;
                break;
        }
        else if (pn.getPT()[pindx] == 0) {
        //from all marked places which PTi=0, pick the one whose next operational place
        //minimizes a criterion given by iv

                pnext = pn.getCplus()[enabledTransitions[j]][0];

                if (iv[pnext] < min || notokeninbuf==false) {
                        min = iv[pnext];
                        z=j;
                }
                notokeninbuf = true;
        }

        else if ( pn.getPT()[pindx] > 0 && notokeninbuf==false) {
        //from all marked places which PTi>0 and Mri>0, pick
        //the one whose next PT is minimum
        //only works if there are no tokens in buffer places (PTi=0);
```

```
                        if (pn.getMr()[pindx] < min) {
                                min = pn.getMr()[pindx];
                                z=j;
                        }
                }
                j++;
        }

        return z;

}

intvec findNextMinTimeVector(const intvec& um, const intvec& iv, const PetriNet& pn) {

//Rule to select a transition
//finds the minimum of the remaining job times of output places of the enabled transitions
//The rule applies as follows:
//(a)If a token in a place has exhausted its process time, it has priority so it releases the
//resources it seized.
//(b) If there are tokens in buffer places (Process Time=0), pick the transition whose output
//place (PT>0 by definition) is minimum.
//(c) If all tokens are in operational places (PT>0) fire the transition whose input place has
//the minimum Mr (Remaining Process Time).

        assert(um.size()>0);
        assert(iv.size()>0);
        assert (pn.getCplus().size()>0);
        assert (pn.getCmin().size()>0);
        assert (pn.getPT().size()>0);

        intvec z;
        int j=0;
        int pindx;
        int pnext;
```

```
int min=100000;


bool notokeninbuf= false;
//checks if there is at least one token in a buffer place


while (j<um.size()) {


        pindx = pn.getCmin()[um[j]][0];


        if (pn.getPT()[pindx] > 0 && pn.getMr()[pindx] == 0) {
        //checks which places exhausted their process time. give priority to those.
                z.push_back(j);
                z.push_back( pn.getPT()[pindx]);
                z.push_back(pn.getp_id()[pindx]);
                break;
        }
        else if (pn.getPT()[pindx] == 0) {
        //from all marked places which PTi=0, pick the one whose next operational place
        //minimizes a criterion given by iv


                pnext = pn.getCplus()[um[j]][0];


                if (iv[pnext] < min || notokeninbuf==false) {
                        min = iv[pnext];
                        z.push_back(j);
                    z.push_back( pn.getPT()[pindx]);
                    z.push_back(pn.getp_id()[pindx]);
                }
                notokeninbuf = true;
        }


        else if ( pn.getPT()[pindx] > 0 && notokeninbuf==false) {
        //from all marked places which PTi>0 and Mri>0, pick
        //the one whose next PT is minimum
```

```
                //only works if there are no tokens in buffer places (PTi=0);

                        if (pn.getMr()[pindx] < min) {
                                min = pn.getMr()[pindx];
                                z.push_back(j);
                            z.push_back( pn.getPT()[pindx]);
                            z.push_back(pn.getp_id()[pindx]);
                        }
                }
                j++;
        }

        return z;

}

int findnextmaxtime(const intvec& um, const intvec& iv, const PetriNet& pn) {

//Rule to select a transition
//finds the maximum of the remaining job times of output places of the enabled transitions
//The rule applies as follows:
//(a)If a token in a place has exhausted its process time, it has priority so it releases the
//resources it seized.
//(b) If there are tokens in buffer places (Process Time=0), pick the transition whose output
//place (PT>0 by definition) is minimum.
//(c) If all tokens are in operational places (PT>0) fire the transition whose input place has
//the minimum Mr (Remaining Process Time).

        assert(um.size()>0);
        assert(iv.size()>0);
        assert (pn.getCplus().size()>0);
        assert (pn.getCmin().size()>0);
        assert (pn.getPT().size()>0);
```

```
int z;
int j=0;
int pindx;
int pnext;
int max=0;
int min=100000;
int tiempoRestanteMin = 1000000;
int ztemp;
int cont =0;

bool notokeninbuf= false;
//checks if there is at least one token in a buffer place

int t1, t2, t3;

while (j<um.size()) {
        pindx = pn.getCmin()[um[j]][0];
        t1 = pindx;
        t2= pn.getp_id()[pindx];

        if (pn.getPT()[pindx] > 0 && pn.getMr()[pindx] == 0) {
        //checks which places exhausted their process time. give priority to those.
                        z=j;
                        break;
        }

        else if (pn.getPT()[pindx] == 0) {
        //from all marked places which PTi=0, pick the one whose next operational place
        //minimizes a criterion given by iv

                pnext = pn.getCplus()[um[j]][0];
                t3= iv[pnext];
```

```
                        if (iv[pnext] > max || notokeninbuf==false) {
                                max = iv[pnext];
                                z=j;
                        }
                        notokeninbuf = true;
                }

                else if ( pn.getPT()[pindx] > 0 && notokeninbuf==false) {
                //from all marked places which PTi>0 and Mri>0, pick
                //the one whose next PT is minimum
                //only works if there are no tokens in buffer places (PTi=0);

                        if (pn.getMr()[pindx] < min) {
                                min = pn.getMr()[pindx];
                                z=j;
                        }
                }

                j++;
        }

        if(cont == um.size()){
                z = ztemp;
        }

        return z;
}

int findpresmaxtime(const intvec um, const intvec& iv, const PetriNet& pn) {

//Rule to select a transition
//finds the maximum of the remaining job times of input places of the enabled transitions
//The rule applies as follows:
```

```
//(a)If a token in a place has exhausted its process time, it has priority so

//it releases the resources it seized.

// (b) If there are tokens in buffer places (Process Time=0), pick the transition whose output
//place (PT>0 by definition) is maximum.

// (c) If all tokens are in operational places (PT>0) fire the transition whose input place
//has the minimum Mr (Remaining Process Time).


        assert(um.size()>0);

        assert(iv.size()>0);

        assert (pn.getCplus().size()>0);

        assert (pn.getCmin().size()>0);

        assert (pn.getPT().size()>0);


        int z;

        int j=0;

        int pindx;

        int min=10000;

        int max =0;

        bool notokeninbuf= false;

        //checks if there is at least one token in a buffer place


        while (j<um.size()) {


                pindx = pn.getCmin()[um[j]][0];


                if (pn.getPT()[pindx] > 0 && pn.getMr()[pindx] == 0) {

                //checks which places exhausted their process time. give priority to those.

                        z=j;

                        break;

                }

                else if (pn.getPT()[pindx] == 0) {

                //from all marked places which PTi=0, pick the one whose next operational place
                //maximizes a criterion given by iv
```

```
                        if (iv[pindx] >= max || notokeninbuf==false) {
                                max = iv[pindx];
                                z=j;
                        }
                notokeninbuf = true;
                }

                else if ( pn.getPT()[pindx] > 0 && notokeninbuf==false) {
                //from all marked places which PTi>0 and Mri>0, pick the one whose next PT is
                minimum only works if there are no tokens in buffer places (PTi=0);

                        if (pn.getMr()[pindx] < min) {
                                min = pn.getMr()[pindx];
                                z=j;
                        }
                }
                j++;
        }

        return z;

}



int max(int n, int m) {

        if (n >= m)
                return n;
        else
                return m;
}
```

Schedule.h: interface for the Schedule class.

```
#ifndef Schedule_H_
#define Schedule_H_



#include "PetriNet.h"
#include "Operation.h"
#include <stdlib.h>
#include <iomanip>
using namespace std;

typedef vector<Operation> Sequence;

/**
* \file    Schedule.h
* \brief   Clase Schedule
* \version 1.0
* \date    25/02/2005
* \author  gmejia
* \par     Proyecto Investigaci&oacute;n
*
* \par     Descripci&oacute;n
*                      La clase Schedule representa un programa parcial o total de
producci&oacute;n.
*                 El miembro principal "gantt" es un vector de "Secuences". Cada Sequence
corresponde
*                 un recurso y contiene al orden en el cual los trabajos van al recurso. Una
Sequence
*                 es por lo tanto un vector de "Operations".
*                 Los otros miembros son:
```

```
*                 resTimes: Guarda los tiempos acumulados de cada recurso (tiempo de
finalizaci&oacute;n del &uacute;ltimo

*                 trabajo PROGRAMADO en el recurso).

*                 resTimes: Guarda los tiempos acumulados de cada trabajo (tiempo de
finalizaci&oacute;n de la &uacute;ltima

*                 operaci&oacute;n PORGRAMADA del trabajo).

*

*

* \todo

*

*/


class Schedule

{

public:


        /**


        * \brief   Constructor de la Clase Schedule

        *

        * \pre   True

        * \post  Se tiene un nuevo objeto de la clase Schedule

        */

        Schedule();


        /**


        * \brief   Destructor de la Clase Schedule

        *

        * \pre   Se tiene un nuevo objeto de la clase Schedule

        * \post   True

        */

        virtual ~Schedule();
```

```
        /**

        * \brief   Genera un Schedule a partir de una Red de Petri pn

        *                 y una secuencia de transitions u

        *

        * \pre    Un objeto de la clase PetriNet y una deque<int> u

        *                correspondiente a una secuencia v&aacute;lida de transitions

        * \post   Se tiene un objeto de la clase Schedule con una programaci&oacute;n
v&aacute;lida

        */

        Schedule genSchedule(PetriNet pn, int ta, const deque<int>& u, deque<int>& t);

        /*Schedule genScheduleR(PetriNet pn, const deque<int>& ur, deque<int>& t); /////OOO*/

        /**

        * \brief    Recupera el componente gantt de la Clase Schedule.

        *                     gantt es un vector de vectores (vec2dim)

        */

        const vector <Sequence>& getGantt() const;



        /**

        * \brief    Recupera el componente resTimes de la Clase Schedule.

        *                     resTimes es un vector de enteros (intvec)

        */

        const intvec& getResTimes() const;



        /**

        * \brief    Recupera el componente jobTimes de la Clase Schedule.

        *                     jobTimes es un vector de enteros (intvec)
```

```
        */
        const intvec& getJobTimes() const;



        /**

        * \brief   Imprime en el directorio Temp un programa v&aacute;lido que puede ser le&iacute;do por Lekin
        *
        * \pre   Se tiene un directorio v&aacute;lido Temp y un programa v&aacute;lido de la clase Schedule
        * \post  Se imprime en el archivo _user.seq el programa de producci&oacute;n.
        * \param wkcArray: Un vector v&aacute;lido de workcenters (clase WkCenter)
        */
        void printToLekin(const vector<WkCenter>& wkcArray);


private:
        vector<Sequence> gantt;
        intvec resTimes;
        intvec jobTimes;
};


#endif // !defined(AFX_SCHEDULE_H__B2E682A6_49F4_4930_89B5_244B26E0DF35__INCLUDED_)
```

Schedule.cpp: implementation of the Schedule class.

```
#include "Schedule.h"

//////////////////////////////////////////////////////////////////////
```

```
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

Schedule::Schedule()
{

}

const vector<Sequence>& Schedule::getGantt() const{

        return gantt;
}



const intvec& Schedule::getResTimes() const{

        return resTimes;
}

const intvec& Schedule::getJobTimes() const{

        return jobTimes;
}

Schedule Schedule::genSchedule (PetriNet pn, int ta, const deque<int>& u, deque<int>& t) {

        int delta;
        int pl_id;
        int jobNu; //pl_id gets the id of a place. It can be a job# or a resource#
        exception e;

        Operation op;
        try {
```

```
int nr = nResources(pn.getp_id());

gantt.resize(nr);

intvec resUsed;
//stores the numbers of those resources used when firing a transition u[k]

int tnow = 0;
for (int k=0; k<u.size(); k++ ) {

        resUsed.clear();

        if (u[k] < 0)
                throw e;



        for (int i=0;i<pn.getCmin()[u[k]].size();i++) {

                if (pn.getMp()[pn.getCmin()[u[k]][i]] < 1) {
                        throw e;

                }
        }

        intvec um(1);
        um[0] = u[k];

        pn.changeState(um,0,delta);
        tnow = tnow + delta;
        t.push_back(tnow);

        for (int i=0;i<pn.getCmin()[u[k]].size();i++) {
                pl_id = pn.getp_id()[pn.getCmin()[u[k]][i]];
```

```
                        if (pl_id >= 0)
                                jobNu = pl_id;
                        else
                                resUsed.push_back(-pl_id);
                }

                for (int i=0;i<resUsed.size();i++) {
                        op.setJob(jobNu);
                        op.setRes(resUsed[i]);
                        op.setStart(tnow);
                        op.setTrans(u[k]);
                        assert (resUsed[i]-1 >= 0);
                        int id = resUsed[i] - 1;
                        gantt[id].push_back(op);
                }

                for (int i=0;i<pn.getCplus()[u[k]].size();i++) {

                        pl_id=pn.getp_id()[pn.getCplus()[u[k]][i]];
                        if (pl_id < 0){
                                pl_id=-pl_id;
                                gantt[pl_id-1].back().setEnd(tnow);
                        }
                }
        }

        return *this;
}

catch (exception e) {

        //cout << "Invalid Schedule" << endl;
```

```
                system("PAUSE");
                exit(1);

        }

}

void Schedule::printToLekin(const vector<WkCenter>& wkcArray){

        ofstream out("_user.seq",ios::out);

        if (out) {

                out<< "Schedule " << setw(14) << "PetNMA (Marzo 4 - 15)" << endl;

                int rn = 0;  //indice controla número de recursos totales

                for (int i = 0; i < wkcArray.size(); i++) {


                        for (int j=0; j< wkcArray[i].getNumMch(); j++) {

                                out<<  "Machine" << setw(15) << i << "." << j << endl;

                                for (int k=0; k < gantt[rn].size(); k++) {

                                        out << "Oper:" << setw(10) << gantt[rn][k].getJob() <<
endl;
                                }

                        rn++;

                        }
                }
```

```
        }

        out.close();
}
```

Tjob.h: interface for the Tjob class.

```
#ifndef Tjob_H_
#define Tjob_H_


#include <vector>
using namespace std;

#include <iostream>
#include <fstream>

const int MAXNOP = 2000; //Maximum Number of OPerations

class Tjob
{

friend bool operator < (const Tjob& , const Tjob& );
friend void ReadJobFile(vector<Tjob>& jobArray,char* jobtype);

public:
        Tjob();
        virtual ~Tjob();
```

```
//set methods
void setId(const char* idn);
void setRelease(const int& rls);
void setDue(const int& dd);
void setWeight(const int& wgt);
void setOptime(const int& optime, const int& i);

//read methods.
void readId(istream& is);
void readRelease(istream& is);
void readDue(istream& is);
void readWeight(istream& is);

// get methods
const char* getId() const;
int getNumber() const;
int getRelease() const;
int getDue() const;
int getWeight() const;
int getNOp() const;
const int* getOpTimes() const;
int getOpTimes_i(const int& i) const;
 //returns the operation time of the i-th operation
const int* getRoute() const;
int getRoute_i(const int& i) const;
 //returns the machine that process the i-th operation
const int* getStatus() const;
int getStatus_i(const int& i) const;

//print methods
void printJob(ostream& os);
```

```
private:
  char id[80];
  // job id
  int jobNumber;
  // jobNumber
  int release;
  // job release date
  int due;
  // job due date
  int weight;
  // job weight
  int numOp;
  // Number of operations
  int opTimes[MAXNOP];
  // vector with operation times
  int route[MAXNOP];
  // "Routing sheet". Vector with the ordered sequence of machines (precedence constraints)
  int status[MAXNOP];
  // type of each operation (for sequence dependent setups)

};

#endif // !defined(AFX_TJOB_H__5C2973A3_6FE6_48F8_B9A3_976480EDA146__INCLUDED_)
```

Tjob.cpp: implementation of the Tjob class.

```
#include "Tjob.h"
#include <functional>

//////////////////////////////////////////////////////////////////////
```

```
//Friend methods///
//////////////////////////////////////////////////////////////////////

//Method that reads a vector of jobs from the file _user.job

void ReadJobFile(vector<Tjob>& jobArray, char *jobtype){

  char buffer[1024];
  //data structure used for storing data (string)
  char machBuffer [4];
  //data structure used for storing data (string)
  int jobCount =0;

  ifstream Fjob("_user.job", ios::in);
  if (!Fjob){ //cout << "Job File not found!\n";
    exit(1);
  }

  Fjob >> buffer; // buffer = "Shop:", ignore
  Fjob >> buffer; //Job Type must be "Single" or "Shop" or "Flow"


  if (strcmp(buffer, "Single") != 0 && strcmp(buffer, "Job") != 0 && strcmp(buffer, "Flow") !=
  0){
//cout << "This is not a Valid Job File!\n";
    getchar();
    exit(1);
  }
  else
         strcpy(jobtype,buffer);


       Fjob >> buffer;
       // buffer = "Job:"
```

```
if (strcmp(buffer, "Job:") != 0) {
// If not, must be the end of file
   //cout << "This is not a Valid Job File!\n";
        getchar();
        exit(1);
}

while (!Fjob.eof() && jobCount < MAXNOP){

        jobArray.resize(jobArray.size()+1);

        jobArray[jobCount].jobNumber = jobCount;
        jobArray[jobCount].readId(Fjob);
        // buffer = "Job###"

        Fjob >> buffer;
        // buffer = "Release:"
        jobArray[jobCount].readRelease(Fjob);

        Fjob >> buffer;
        // buffer = "Due:"
        jobArray[jobCount].readDue(Fjob);



        Fjob >> buffer;
        // buffer = "Weight:"
        jobArray[jobCount].readWeight(Fjob);

        Fjob >> buffer;             // buffer = "Oper:"
        int opCount = 0;

        while (strcmp(buffer,"Oper:") == 0 ){
```

```
Fjob >> buffer;
// buffer = "Wkc000;#;A", and we need the #

char* posSemicolon1 = strchr(buffer,';');
if (posSemicolon1 == NULL)
//looks for the character ';' in buffer
        break;


machBuffer[0] = buffer[3];
machBuffer[1] = buffer[4];
machBuffer[2] = buffer[5];

jobArray[jobCount].route[opCount] = atoi(machBuffer);

char* posSemicolon2 = strchr(posSemicolon1+1,';');
if (posSemicolon2 == NULL)
break;

jobArray[jobCount].status[opCount]= *(posSemicolon2+1)-64;
//reads the status and transforms it from letters to numbers

int pj;
//aux variable for processing time

if (sscanf(posSemicolon1+1, "%d", &pj) < 1) break;

jobArray[jobCount].opTimes[opCount] = pj;

Fjob >> buffer;
// buffer must be "Oper:" or "Job:" if no more operations
```

```
                        opCount++;


                }

                jobArray[jobCount].numOp = opCount;
                //jobArray[jobCount].printJob(//cout);
                jobCount++;
  }


  if (jobCount == 0)
  { //cout << "No jobs defined!\n";
    exit(1);
  }

}

////////////////////////////////////////////////////////////////////////
// Construction/Destruction
////////////////////////////////////////////////////////////////////////

Tjob::Tjob()
{
//opTimes.resize(25);
////cout << "constructor called" << endl;
}

Tjob::~Tjob()
{

}
```

```
//////////////////////////////////////////////////////////////////////
// Set methods
//////////////////////////////////////////////////////////////////////


void Tjob::setId(const char* idn) {

        strcpy(id,idn);

}

void Tjob::setRelease(const int& rls){
        release = rls;

}



void Tjob:: setDue(const int& dd){
        due = dd;

}



void Tjob::setWeight(const int& wgt){

        weight = wgt;
}

void Tjob::setOptime(const int& optime, const int& i){

opTimes[i] = optime;

}
```

```
//////////////////////////////////////////////////////////////////////
// Read methods
//////////////////////////////////////////////////////////////////////

void Tjob::readId(istream& is){

        is >> id;
}



void Tjob::readRelease(istream& is){

        is >> release;
}

void Tjob::readDue(istream& is){

        is >> due;
}

void Tjob::readWeight(istream& is){
        is >> weight;
}

//////////////////////////////////////////////////////////////////////
// get Methods
//////////////////////////////////////////////////////////////////////

const char* Tjob::getId() const
        {return id;}

int Tjob::getRelease() const
```

```
        {return release;}

int Tjob::getDue() const
        {return due;}

int Tjob::getWeight() const
        {return weight;}

int Tjob::getNumber() const
        {return jobNumber;}

int Tjob::getNOp() const
        {return numOp;}

const int* Tjob::getOpTimes() const
        {return opTimes;}

int Tjob::getOpTimes_i(const int& i) const
        {return opTimes [i];}

const int* Tjob::getRoute() const //returns the array with the routing
    {return route;}

int Tjob::getRoute_i(const int& i) const//returns the machine that process the i-th operation
    {return route[i];}

const int* Tjob::getStatus() const //returns the array with the statuses of the job operations
    {return status;}

int Tjob::getStatus_i(const int& i) const
    {return status[i];}

//////////////////////////////////////////////////////////////////////
```

```
// print Methods
//////////////////////////////////////////////////////////////////////

void Tjob::printJob (ostream& os) {

        os << "Job: " << id << endl;
        os << "Release: " << release << endl;
        os << "Due: " << due << endl;
        os << "Weight: " << weight << endl;
        char ch = 'A';

        for (int i= 0; i < numOp; i++) {

                if (route[i] < 9)
                  os << "Operation Wkc00" << route[i] << ": " << opTimes[i] << " Status: ";
                else if (route[i] < 99)
                  os << "Operation Wkc0" << route[i] << ": " << opTimes[i] << " Status: ";
         else
           os << "Operation Wkc" << route[i] << ": " << opTimes[i] << " Status: ";


         ch = ch + status[i] -1;
                os << ch  << endl;
                ch = 'A';
        }

        os << endl;

}
```

---------- WkCenter.h: interface for the WkCenter class. ----------

```
#ifndef WkCenter_H_
#define WkCenter_H_

#include <iostream>
#include <fstream>
#include <iterator>
#include <vector>
#include "Machine.h"
#include "matrix.h"

using namespace std;

//This class represents a workstation which is a set of identical parellel machines

class WkCenter
{

friend class Schedule;



friend void ReadWkcFile(vector<WkCenter>& wkcArray,char* wktype);

public:
        WkCenter();                                     //Default constructor
        virtual ~WkCenter()    ;              //Destructor
        void readId(istream& is);   //reads the id of the workcenter

        //get methods
        const char* getId() const;
        int getNumMch() const;
```

```
        const matrix& getSetup() const;
        const vector<Machine>& getMchArray() const;

        //print method
        void printWk(ostream& os);

private:

  char id[80];                        // WkCenter id
  vector<Machine> mchArray; //array of parallel machines belonging to the workcenter
  matrix setup;                       //matrix with setup times
  int numMch;                         //number of machines in the workcenter

};

#endif // !defined(AFX_WkCenter_H__057437C7_5E9A_4608_8E6F_FBCB67CC7464__INCLUDED_)
```

WkCenter.cpp: implementation of the WkCenter class.

```
#include "WkCenter.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

WkCenter::WkCenter(){}

WkCenter::~WkCenter(){}

//////////////////////////////////////////////////////////////////////
```

```
// read methods
//////////////////////////////////////////////////////////////////////

void WkCenter::readId(istream& is){
        is >> id;}

//////////////////////////////////////////////////////////////////////
// get methods
//////////////////////////////////////////////////////////////////////

const char* WkCenter::getId() const
        {return id;}

int WkCenter::getNumMch() const
        {return numMch;}

const matrix& WkCenter::getSetup() const
        {return setup;}

const vector<Machine>& WkCenter::getMchArray() const
        {return mchArray;}

//////////////////////////////////////////////////////////////////////
// print methods
//////////////////////////////////////////////////////////////////////

void WkCenter::printWk(ostream &os) {

        os << "Workcenter " << id << endl;
        os << "Setup Matrix " << endl;
        os <<  setup;

        for (int i= 0; i < numMch; i++) {
```

```
                os << "Machine: " << mchArray[i].mchId << endl;
                os << "Release: " << mchArray[i].release << endl;
                os << "Status: " <<  mchArray[i].status << endl;

        }
        os << endl;

}

//Reads workcenter data from the file _user.mch in the TEMP directory

void ReadWkcFile(vector<WkCenter>& wkcArray, char* wktype){

        char buffer[1024];
        //data structure used for storing data (string)

        int wkCount =0;

        ifstream Fmch("_user.mch", ios::in);
        if (!Fmch){
        //cout << "Job File not found!\n";
                exit(1);
        }

        Fmch >> buffer;
        //WkCenter Type must be "Ordinary" or "Flexible"

        if (strcmp(buffer, "Ordinary:") != 0 && strcmp(buffer, "Flexible:") != 0){
         //cout << "This is not a Valid Machine File!\n";
                getchar();
                exit(1);
        }
```

```
strcpy(wktype,buffer);
wktype=buffer;

Fmch >> buffer;             // buffer = "Workcenter:"

if (strcmp(buffer, "Workcenter:") != 0) {// If not, must be the end of file
        //cout << "This is not a Valid Machine File!\n";
        getchar();
        exit(1);
}

while (!Fmch.eof()){

        wkcArray.resize(wkcArray.size()+1);

        wkcArray[wkCount].readId(Fmch);// buffer = "Wkc###"

        Fmch >> buffer; //buffer = "Setup:"

        Fmch >> buffer;
        //setup matrix. If no setup matrix, next line must be "Machine:"
        //typical setting: A;B;4 B;A;3 B;B;2

        wkcArray[wkCount].setup.resize_matrix(1,1);

        while (strcmp(buffer,"Machine:") !=0) {

                int i = buffer[0] - 'A';
                //row index of setup matrix

                char* posSemicolon1 = strchr(buffer,';');
                if (posSemicolon1 == NULL)
                //looks for the first character ';' in buffer
```

```
                break;

        int j = *(posSemicolon1+1) - 'A';
        //column index of setup matrix

        wkcArray[wkCount].setup.resize_sqmatrix(i+1,j+1);

        char* posSemicolon2 = strchr(posSemicolon1+1,';');
        if (posSemicolon2 == NULL)
                break;

        int sij;
        //aux variable for setup times

        if (sscanf(posSemicolon2+1, "%d", &sij) < 1)
                break;

        wkcArray[wkCount].setup(i,j) = sij;
        Fmch >> buffer;

}

int mchCount = 0;
//Counts the number of machines

while (strcmp(buffer,"Machine:") ==0) {

        Fmch >> buffer; //reads the machine id
        wkcArray[wkCount].mchArray.resize(wkcArray[wkCount].mchArray.size()+1);
        wkcArray[wkCount].mchArray[mchCount].setMchId(buffer);

        Fmch >> buffer; //"Release" ignore
        Fmch >> buffer;
```

```
                wkcArray[wkCount].mchArray[mchCount].setRelease(atoi(buffer));

                Fmch >> buffer; //"Status" ignore

                Fmch >> buffer; //Status letter from A to Z
                wkcArray[wkCount].mchArray[mchCount].setStatus(buffer);

                Fmch >> buffer;
                mchCount ++;
            }

            wkcArray[wkCount].numMch = mchCount;
            //wkcArray[wkCount].printWk(//cout);
            wkCount++;

        }

}
```