UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Optimización multi-objetivo para la programación de la producción

TESIS DOCTORAL

Presentada por:
Gerardo Gabriel Minella
Dirigida por:
Rubén Ruiz García

Valencia, mayo de 2014

A mi hijo Pablo, a mi hija Andrea,
a mi querida esposa Nieves,
a mis padres Rosario y José

Agradecimientos

En primer lugar quiero agradecer enormemente a Rubén Ruiz, quien ha sido durante todo el transcurso de esta tesis un gran apoyo e impulsor y quien no ha dudado en dedicar generosamente una gran cantidad de su propio tiempo. Además, quiero agradecer especialmente a mi esposa Nieves, quien me ha apoyado todo el tiempo en este proyecto. También a mis padres Rosario y José quienes han sido un constante apoyo y ejemplo, no solo en el desarrollo de esta tesis, sino en toda mi vida. Son ellos quienes me han inculcado la importancia del estudio y la educación y quienes se han esforzado para que pueda tenerla.
También agradezco especialmente a Michele Ciavotta y Thijs Urlings y Eva Vallada con quienes he realizado parte de mi trabajo de investigación y de quienes he aprendido muchas cosas, y sobre todo quienes han sido siempre grandes amigos.

Resumen

El problema del taller de flujo surge hace unos 60 años como una aproximación de la realidad de los procesos industriales de fabricación, más exactamente de la programación de la producción. La programación de la producción se refiere a la ordenación de las tareas productivas pendientes en una industria fabril. A pesar de que han pasado muchos años desde sus comienzos, aún hoy existe una gran diferencia entre los problemas teóricos propuestos y la realidad industrial de las empresas.
Una de las diferencias más evidentes es el hecho de que al intentar resolver un problema de programación de la producción casi nunca se tiene en mente un único objetivo. Normalmente se tienen en mente varias cosas a la vez, como por ejemplo, terminar cuanto antes la producción, al mismo tiempo maximizar el uso de recursos y también cumplir con las fechas de entregas. En este contexto han surgido los problemas de taller de flujo multi-objetivo.
En los últimos 20 años los problemas de taller de flujo multi-objetivo han tenido un gran empuje, acercado el desarrollo teórico a los problemas reales.

En este trabajo de tesis presentaremos un recorrido por algunos de los problemas de taller de flujo multi-objetivo, partiendo desde los más básicos y yendo hacia los más complejos, y al mismo tiempo, los que reflejan mejor la realidad.
Este trabajo tiene además otros objetivos. Uno de los problemas que más se ha dejado de lado en la optimización multi-objetivo es la medición y comparación

correcta de los resultados. Presentaremos un recorrido por los métodos existentes para la medición de resultados multi-objetivo, señalando los problemas y ventajas de cada uno, con la finalidad de obtener una metodología válida, clara y consistente para la comparación de los resultados de problemas multi-objetivo.

Para comenzar el recorrido por el taller de flujo planteamos una tarea que nunca se ha llevado a cabo hasta la fecha: la implementación y comparación experimental de 23 algoritmos multi-objetivo. Alguno de ellos propuestos para el taller de flujo multi-objetivo y otros de carácter general. Esto nos dará un importante punto de partida para conocer las metodologías existentes en la literatura parar resolver problemas multi-objetivo. Como resultado conoceremos metodologías que van desde algoritmos genéticos, pasando por la búsqueda tabú, colonias de hormigas, recocido simulado, etc.
Todo este trabajo inicial nos permitirá ver las ventajas y desventajas de cada método propuesto y determinar los puntos fuertes de los mejores para, finalmente, proponer un método de resolución de problemas de taller de flujo general, eficaz y eficiente.

El recorrido por distintos problemas de taller de flujo nos permitirá conocer el estado actual de la literatura y acercarnos paso a paso a los problemas que mejor representan la realidad. En cada paso realizaremos un profundo estudio del estado actual de la literatura, comparando los métodos existentes contra un método propuesto por nosotros mismos. En este aspecto partiremos del problema del taller de flujo de permutación multi-objetivo, luego ampliaremos este problema añadiéndole tiempos de cambio dependientes de la secuencia y finalmente estudiaremos el problema del taller de flujo híbrido multi-objetivo.

Abstract

The flowshop problem emerges about 60 years ago as an approximation to some manufacturing industrial processes, more precisely to scheduling problems. Scheduling refers to the problem of finding an order to pending production tasks on a manufacturing industry. After so much study in the past years, today there is still a great gap between theorical problems and the reality of industrial enterprises.

One of the most relevant differences is the fact that when solving a real life scheduling problem it is usual to have in mind more than one objective, i.e. to finish the production as soon as possible, and at the same time to maximise the use of resources an also to meet customer due dates. In this context multi-objective problems started to show their relevance.
In the las 20 years multi-objective flowshop problems have seen a great development, diminishing the existing gap between theorical and real life problems.

In this thesis we present a review of some multi-objective flowshop problems, starting with more basic problems and advancing to more complex and more realistic problem settings.
This work also has other objectives. One of the issues that are frequently ignored in multi-objective optimization is the correct measurement and comparison of multi-objective results. We will present a review of the some of the most used methods, pointing out the problems and advantages of each one, with the

final objective of obtaining a valid, clear, consistent and sound methodology for the comparison of multi-objective problem results.

To begin with our work in the multi-objective flowshop problem, we will propose a task never carried out before: the implementation and experimental comparison of 23 multi-objective algorithms. Some of them where proposed for solving the multi-objective flowshop problem and others are of general purpose. This will give us deep knowledge about different methodologies proposed for solving multi-objective problems. As a result of all this implementation and experimentation work we will understand methodologies from genetic algorithms, tabu search, ant colony optimization to simulated annealing, etc. All this will allow us to see the advantages an disadvantages of each proposed method and to determine their strengths and weaknesses, thus we can finally propose a competitive, fast and effective general purpose algorithm for solving multi-objective problems.

This review of different flowshop problems will allow us to know the state-of-the-art of the literature and at the same time to get closer to real life problems. In each step we will study more complex problems, analyzing and comparing existing methods against our own proposed one. In order to achieve this we will start from the multi-objective permutation flowshop problem, then we widen this problem by adding sequence dependent setup times, and finally we will study the multi-objective hybrid flowshop problem.

Resum

El problema del taller de flux sorgeix fa uns 60 anys com una aproximació de la realitat dels processos industrials de fabricació, més exactament de la programació de la producció. La programació de la producció es refereix a l'ordenació de les tasques productives pendents en una indústria fabril. A pesar que han passat molts anys des dels seus començaments, avui existeix una gran diferència entre els problemes teòrics proposats i la realitat industrial de les empreses.

Una de les diferències més evidents és el fet que a l'intentar resoldre un problema de programació de la producció quasi mai es té en ment un únic objectiu. Normalment es tenen en ment diverses coses alhora, com per exemple, acabar com més prompte la producció, al mateix temps maximitzar l'ús de recursos i també complir amb les dates de lliuraments. En aquest context han sorgit els problemes de taller de flux multi-objectiu.

En els últims 20 anys els problemes de taller de flux multi-objectiu han tingut una gran embranzida, acostat el desenvolupament teòric als problemes reals.

En aquest treball de tesi presentarem un recorregut per alguns dels problemes de taller de flux multi-objectiu, partint des dels més bàsics i anant cap als més complexos, i al mateix temps, els quals reflecteixen millor la realitat.

Aquest treball té a més altres objectius. Un dels problemes que més s'ha deixat de costat en l'optimització multi-objectiu és el mesurament i comparança correcta dels resultats. Presentarem un recorregut pels mètodes existents per al mesurament de resultats multi-objectiu, assenyalant els problemes i avantatges de cadascun, amb la finalitat d'obtenir una metodologia vàlida, clara i consistent per a la comparança dels resultats de problemes multi-objectiu.

Per a començar el recorregut pel taller de flux plantegem una tasca que mai s'ha portat a terme fins a la data: la implementació i comparança experimental de 23 algorismes multi-objectiu. Algun d'ells proposats per al taller de flux multi-objectiu i altres de caràcter general. Açò ens donarà un important punt de partida per a conèixer les metodologies existents en la literatura parar resoldre problemes multi-objectiu. Com resultat coneixerem metodologies que van des d'algorismes genètics, passant per la recerca tabú, colònies de formigues, simmulated annealing, etc.

Tot aquest treball inicial ens permetrà veure els avantatges i desavantatges de cada mètode proposat i determinar els punts forts dels millors per a, finalment, proposar un mètode de resolució de problemes de taller de flux general, eficaç i eficient.

El recorregut per diferents problemes de taller de flux ens permetrà conèixer l'estat actual de la literatura i acostar-nos pas a pas als problemes que millor representen la realitat. En cada pas realitzarem un profund estudi de l'estat actual de la literatura, comparant els mètodes existents contra un mètode proposat per nosaltres mateixos. En aquest aspecte partirem del problema del taller de flux de permutació multi-objectiu, després ampliarem aquest problema afegint-li temps de canvi depenents de la seqüència i finalment estudiarem el problema del taller de flux híbrid multi-objectiu.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ALGORITMOS

CAPÍTULO 1

INTRODUCCIÓN Y OBJETIVOS

1.1. Introducción

Las organizaciones productivas han evolucionando con el tiempo, los mercados y con el desarrollo de la tecnología. En el siglo pasado, luego de la revolución industrial, se dedicaban a producir en masa. Para reducir los costes una de las opciones mas lógicas era la de realizar lotes de producción más grandes y el tener un catálogo de productos lo más reducido posible. De esta manera, evitaban muchos problemas logísticos, de almacenamiento, aprovisionamiento y transporte de materias primas y productos finales. En la actualidad sin embargo esta situación ha cambiado drásticamente. Con el advenimiento de un mercado global en el cual cada organización tiene la oportunidad de llegar a casi todo el planeta, también deben competir con empresas extranjeras que son capaces de presentar los mismos productos. Para poder entrar en mercados externos y mantener su parte del mercado local, las organizaciones han debido adaptarse, cambiando entre otras cosas la tecnología y la forma de llevar a cabo la producción. En este contexto, para poder competir a nivel global, la

producción debe hacerse más flexible, incrementando el catálogo de productos y al mismo tiempo reduciendo el tamaño de los lotes de fabricación. Estos cambios han impulsado la investigación en la programación de la producción o *scheduling*. El simple hecho de elegir correctamente el orden en el que se realizará cada lote dentro del sistema de producción puede incrementar enormemente el aprovechamiento de la capacidad productiva, y de esta manera reducir costos. Aún así, durante mucho tiempo ha existido una gran brecha entre los problemas de programación de producción estudiados por la ciencia y los problemas de producción reales de las organizaciones productivas.
A pesar de que en la realidad la mayoría de los problemas son multi-objetivo, hasta hace un par de décadas la ciencia se había centrado en el estudio de problemas de un único objetivo. Estos estudios estaban por lo general orientados a encontrar la solución a problemas que eran una simplificación de la realidad muy alejada de ésta.
Existen muchas maneras de representar los entornos productivos, en cada caso se trata de representar un problema diferente que por lo general es una simplificación de la realidad.

Un programa de producción industrial se puede resumir como la asignación de ciertos recursos productivos (máquinas, recursos humanos, etc.) para la producción de algún tipo de producto o servicio en un determinado momento y con un determinado orden.
En general, se pueden dividir las organizaciones productivas en dos grandes grupos, el primer grupo está conformado por los sistemas de producción en continuo. En estos existe poca variedad de productos, que además son similares entre sí, y sus volúmenes de producción son elevados. Luego están los sistemas de producción intermitente, donde existe un amplio catálogo de productos. Los productos de este tipo de sistemas pueden ser tan diferentes entre sí como para obligar a realizar cambios en la maquinaria al pasar de una tipología de producto a otra. Ejemplo de sistemas continuos de producción pueden

ser las compañías cementeras o las productoras primarias de acero, mientras que en sistemas intermitentes de producción tenemos fábricas de automóviles, electrodomésticos, ensambladoras de ordenadores, etc.

En cualquier tipología de organización productiva existen una serie de decisiones que han de tomarse respecto a la producción. A mediano plazo, entre tres meses y tres años, es importante determinar el plan de producción. Este se refiere a qué cantidad de qué productos deberá producirse antes de una fecha particular. Los planes de producción no tienen en cuenta los recursos productivos, o a lo sumo pueden tener en cuenta la capacidad productiva estimada de la organización. Por otro lado los programas de producción son a corto plazo, menos de tres meses, y en ellos se determina, para cada producto que deba procesarse, el momento exacto de su proceso y los recursos que estarán asignados a su producción. Los problemas de scheduling, se refieren justamente a los programas de producción, por lo que su finalidad es encontrar una combinación de asignación de los recursos productivos en un orden particular para producir un conjunto de productos o servicios.

Existen muchas posibles configuraciones o tipologías de los entornos productivos, en algunas de ellas cada recurso tiene un papel específico y diferente a los demás recursos, en otras configuraciones en cambio, varios recursos son capaces de realizar la misma tarea. Cada una de estas configuraciones de producción diferente puede ser representada por un problema de scheduling específico.
El taller de flujo o flowshop es uno de los problemas de scheduling más estudiado en las últimas décadas. Esto es debido a que, a pesar de ser una simplificación de la realidad, este tipo de problemas representa a una gran porción de los sistemas productivos. Existen muchas variantes del problema de flowshop, pero en general, el problema de flowshop básico se puede definir como un conjunto de $N = 1, 2, \ldots, n$ trabajos que han de ser procesados en un

conjunto $M = 1, \ldots, m$ de máquinas [*]. El tiempo de proceso de cada trabajo $j \in N$ en cada máquina $i \in M$ es conocido y se representa como p_{ij}. El orden en que los trabajos visitan las máquinas es el mismo para todos los trabajos. Esto se puede asumir, sin pérdida de generalidad como $1, 2, \ldots, m$.
Una *secuencia* es una sucesión de trabajos que indica en que orden será, cada uno de ellos, procesado en taller. El objetivo de un problema de flowshop es obtener una secuencia u orden de procesamiento para los trabajos, tal que el valor de uno o varios criterios de optimización sea el mejor posible. El número total de posibles soluciones se obtiene del producto de todas las posibles permutaciones de trabajos en las máquinas, lo que se puede expresar como $(n!)^m$ soluciones o *secuencias*.

En la literatura referente al flowshop es común utilizar la misma permutación de trabajos para todas las máquinas con la finalidad de restringir el espacio de soluciones posibles. Este tipo de problemas se llama taller de flujo de permutación o *permutation flowshop* y su sigla en inglés es PFSP (*Permutation FlowShop Problem*). La cantidad de soluciones posibles en este caso se reduce a $n!$.
Históricamente, el estudio del PFSP se centró en un solo criterio de optimización. En las aplicaciones de la vida real, sin embargo, un solo criterio de optimización no es suficiente. La investigación y desarrollo de soluciones para problemas multi-objetivo (o multi-criterio) reviste gran importancia, no solo debido a la naturaleza multi-objetivo de los problemas que se nos plantean en la vida real, sino también debido a que aún existen muchas preguntas sin responder en este área de investigación. En las últimas dos décadas la optimización multi-objetivo ha recibido una gran atención en el campo de la investigación operativa. En este aspecto, se han desarrollado nuevas técnicas para resolver funciones y problemas de la vida cotidiana, que impliquen la optimización de

[*]Sin pérdida de generalidad y por motivos de simplificación de aquí en adelante nos referiremos a los recursos como máquinas. Teniendo en cuenta siempre, que nos podemos referir a cualquier otro tipo de recursos.

varios objetivos o criterios al mismo tiempo, como así también metodologías para medir y comparar los resultados obtenidos mediante estas técnicas.

Existen distintos enfoques para tratar los problemas multi-criterio. El enfoque más instintivo consiste en combinar los objetivos mediante la asignación de un *peso* o *ponderación* a cada uno. Esto permite realizar una combinación, generalmente lineal y convexa, de los objetivos en una única función. Tomando como ejemplo un problema con dos criterios de optimización F_1 y F_2, éste puede convertirse en un problema de un solo criterio mediante una función que combine ambos criterios de la siguiente manera: $\alpha F_1 + (1 - \alpha)F_2$, donde $0 \leq \alpha \leq 1$. El mayor inconveniente de este enfoque es que α debe ser conocido de antemano. Además, aplicando diferentes valores de α al mismo problema los resultados pueden ser completamente diferentes. Si el valor de α es desconocido, una alternativa es obtener varias soluciones variando el valor de α. Pero si F_1 y F_2 son medidos en diferentes escalas, esta aproximación puede ser inviable o difícil de llevar a cabo. Este tipo de enfoque se denomina *enfoque de objetivos ponderados* debido a que los pesos de cada objetivo deben ser determinados o conocidos de antemano.
Debido a que la asignación de un peso requiere alguna manera de medir la diferencia de importancia entre criterios, en muchas ocasiones es difícil o imposible asignar un peso exacto a cada criterio de optimización. Sin embargo, sí que se puede dar una prioridad a cada uno de ellos. Esto es, se puede decidir qué objetivo es más importante que los demás, cual es el segundo, el tercero y así sucesivamente, hasta llegar al objetivo menos prioritario. En este contexto se puede resolver un problema multi-objetivo utilizando la priorización de objetivos. En primer lugar el objetivo más relevante es el que determinará si una solución es mejor que otra. Luego al encontrar dos soluciones que resulten en el mismo valor para el primer objetivo se compara el segundo objetivo, luego el tercero y así sucesivamente. Este enfoque para tratar los problemas multi-objetivo se denomina *enfoque lexicográfico*.

Otro enfoque de amplia difusión es la *programación por metas*, en este caso a cada objetivo se le asigna una meta de antemano. Esta meta puede ser el estar por encima o debajo de un valor, estar entre dos valores *aceptables* o incluso estar lo más lejos posible de cierto valor. De esta manera el enfoque de *programación por metas* optimiza los valores de los distintos criterios no por su maximización o minimización, sino al acercarse a cumplir las metas prefijadas. El principal inconveniente de esta tipología de problemas es la determinación de las metas para cada objetivo, ya que esto requiere de un amplio conocimiento previo del espacio de los mismos.
En todos estos casos, se requiere un conocimiento previo de los objetivos y el problema, para determinar los pesos, el orden de importancia de los objetivos o las metas que se pretende cumplir. Todos estos casos pueden incluirse dentro de los enfoques denominados *a priori*.

El caso opuesto a los enfoques *a priori* es el enfoque *a posteriori*, donde no es necesario un conocimiento o decisiones previas ya que el resultado es un conjunto de soluciones. De estas soluciones, la persona encargada de tomar las decisiones debe seleccionar luego la que le parece más conveniente, de ahí el nombre de *a posteriori*. El concepto de solución *óptima* no es aplicable en estos casos, ya que una solución S_1 puede tener un mejor valor para el objetivo F_1 que otra solución S_2 y al mismo tiempo un peor valor para el objetivo F_2. Ambas soluciones serán, en este caso, parte del conjunto de soluciones o frontera de Pareto. Todas las soluciones que pertenecen a una frontera de Pareto son igualmente buenas, ya que no hay manera de determinar o medir qué solución es mejor o peor que las demás. Dicho de otra forma, todas las soluciones que pertenecen a un frontera de Pareto son las *mejores* soluciones para un problema dado en un contexto multi-objetivo. La Figura 1.1 ilustra una frontera de Pareto con varias soluciones, donde se asume que los dos objetivos representados son de minimización. Las líneas claras indican el área dominada

por la frontera de Pareto. Cualquier solución que ocupe un lugar dentro de ese área será dominada por una o más soluciones de la frontera de Pareto.

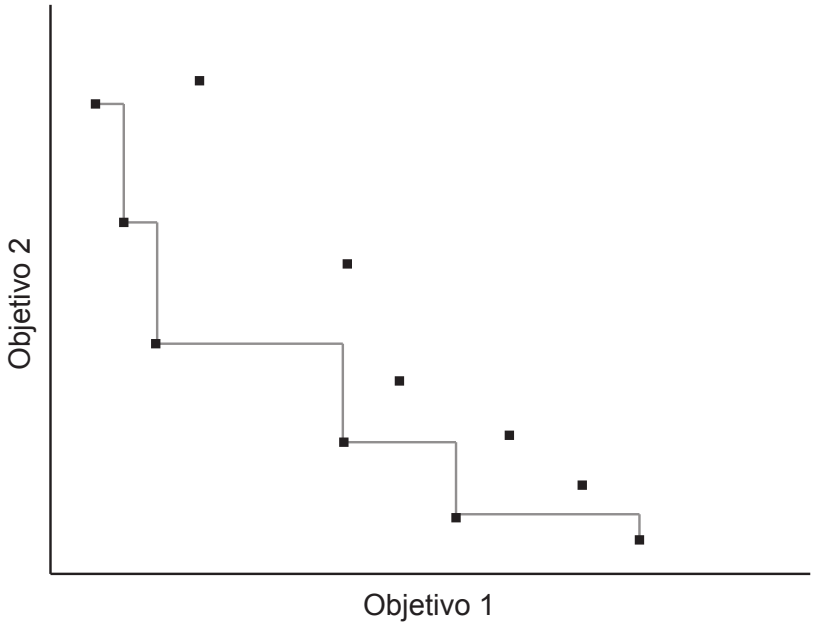


Figura 1.1: Ejemplo de frontera de Pareto para dos objetivos de minimización.

A continuación definiremos más formalmente los conceptos básicos utilizados a lo largo de esta tesis doctoral.

1.1.1. Optimización mono y multi-objetivo

La mayoría de los objetivos que se utilizan comúnmente en problemas de taller de flujo de un solo objetivo se basan en el cálculo del tiempo de finalización o *completion time* de los trabajos. El tiempo de finalización de un trabajo j en una máquina i se representa como C_{ij}, $i \in M, j \in N$. Partiendo

de una permutación π de n trabajos, en la cual $\pi_{(j)}$ se refiere al trabajo que ocupa la posición j dentro de la permutación, los tiempos de finalización se calculan mediante la siguiente expresión:

$$C_{i,\pi_{(j)}} = \text{máx}\left\{C_{i-1,\pi_{(j)}}, C_{i,\pi_{(j-1)}}\right\} + p_{i\pi_{(j)}} \tag{1.1}$$

donde $C_{0,\pi_{(j)}} = 0$ y $C_{i,\pi_{(0)}} = 0$, $\forall i \in M, \forall j \in N$. Aquí el tiempo de finalización del trabajo j es igual a $C_{m,j}$ y se representa comúnmente como C_j.

La Tabla 1.1 y la Figura 1.2 muestran un ejemplo de taller de flujo de permutación para cinco trabajos y cuatro máquinas. En este ejemplo se puede observar la manera de calcular los tiempos de terminación de los trabajos C_j y la forma en que las tareas se ordenan en las distintas máquinas.

máquinas (i)	trabajos (j)				
	1	2	3	4	5
1	31	19	23	13	33
2	41	55	42	22	5
3	25	3	27	14	57
4	30	34	6	13	19

Tabla 1.1: Tiempos de proceso para un ejemplo de problema de taller de flujo con 4 máquinas y 5 trabajos.

El tiempo máximo de proceso es, sin lugar a duda, el criterio simple más estudiado en lo referente a problemas de taller de flujo. Este criterio por lo general es denominado *makespan* y se representa con $C_{\text{máx}} = C_{m,\pi_{(n)}}$. El trabajo de Graham et al. (1979) propone $F/prmu/C_{\text{máx}}$ como notación para referirse al problema de PFSP con el objetivo de $C_{\text{máx}}$. Garey, Johnson y Ravi (1976) demostraron que es un problema $\mathcal{NP}-$Difícil en el sentido fuerte para más de dos maquinas $(m > 2)$.

El tiempo total de finalización (también referido como tiempo total de flujo)

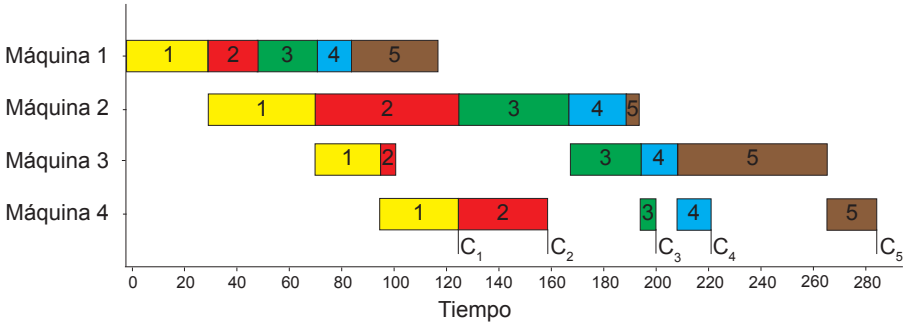


Figura 1.2: Gráfica de Gantt para el ejemplo de taller de flujo con 5 trabajos y 4 máquinas.

es el segundo objetivo más estudiado en el taller de flujo y se representa como $TCT = \sum_{j=1}^{n} C_j$. La notación para el taller de flujo de permutación con el objetivo de minimizar el tiempo total de completación es $F/prmu/\sum C_j$. Este problema es también un problema $\mathcal{NP}-$Difícil para $m \geq 2$ como ha sido demostrado en Gonzalez y Sahni (1978). Algunos resultados para estos objetivos pueden encontrarse en Framiñan, Gupta y Leisten (2004), Ruiz y Maroto (2005) y Ruiz y Stützle (2007), para el $C_{\text{máx}}$, o en Framiñan y Leisten (2006), Tseng y Lin (2009), Tasgetiren et al. (2011) o Pan y Ruiz (2013) para el tiempo total de flujo.

La tardanza total o *total tardiness* es probablemente el tercero de los objetivos más estudiados en el PFSP. Partiendo de una fecha de entrega o $due\ date$ para un trabajo j representada por d_j, la medición de la tardanza para ese trabajo j se representa como T_j y se define como $T_j = \text{máx}\{C_j - d_j, 0\}$. Este tipo de problema es también $\mathcal{NP}-$Difícil en el sentido fuerte, como demostraron Du y Leung (1990). Resultados para el PFSP con el criterio de tardanza total pueden encontrarse en Vallada, Ruiz y Minella (2008).

La literatura sobre taller de flujo para un solo objetivo es inmensa, en

comparación con ella, la literatura que se enfoca los problemas de taller de flujo multi-objetivo es pequeña. En este sentido, sin embargo, ha recibido un gran empuje en las últimas décadas. Aún así la mayoría de los estudios se han realizado utilizando una aproximación llamada de *ponderación de los objetivos*, en la que varios objetivos son combinados en un único objetivo al otorgarles una *ponderación* o *peso*, como ya hemos comentado con anterioridad. En este caso, el principal problema es la determinación de esta ponderación para cada objetivo, sobre todo en casos donde los objetivos tienen dimensiones completamente diferentes. Incluso si se normalizan los valores para todos los objetivos y se asigna un peso a cada uno de ellos, el resultado obtenido dependerá directamente de esa ponderación. Pudiéndose obtener resultados completamente diferentes para distintas combinaciones de valores en los pesos de los objetivos.
Como hemos comentado, la aproximación *a posteriori*, por otro lado, es más compleja en el aspecto que no existe una única solución *óptima* sino que se debe trabajar con un conjunto de soluciones, que por sus características, pueden considerarse como *óptimas* al mismo tiempo. En un problema de optimización multi-objetivo se debe lidiar con un conjunto de funciones a ser optimizadas. Por ejemplo, partiendo de un problema de optimización de dos objetivos F_1 y F_2 que deben ser minimizados, dadas dos soluciones x_1 y x_2 y siendo $F_1(\cdot)$ y $F_2(\cdot)$ los valores de los objetivos para una solución dada. Si, $F_1(x_1) < F_1(x_2)$ y al mismo tiempo $F_2(x_1) < F_2(x_2)$ se puede afirmar que x_1 $es\ mejor\ que$ o $domina\ a\ x_2$. Pero en el caso que exista una tercera solución x_3 tal que $F_1(x_1) < F_1(x_3)$ y $F_2(x_1) > F_2(x_3)$. ¿Cual de estas dos soluciones es la mejor?

Para poder realizar correctamente la comparación de soluciones multi-objetivo es necesario hacer ciertas definiciones. Supongamos que existen M objetivos que deben ser optimizados, sin pérdida de generalidad podemos asumir que todos los objetivos se minimizan. El operador $\lhd$ en la comparación de

dos soluciones multi-objetivo significa $mejor\ que$. Por lo tanto en el ejemplo anterior se podría poner $x_1 \lhd x_2$, significando que la solución x_1 es mejor que la solución x_2 para todos los objetivos a optimizar.
El trabajo de Zitzler et al. (2003) presenta una extensa notación acerca de los problemas multi-objetivo. Esta notación fue extendida aún más en Paquete (2005), Knowles, Thiele y Zitzler (2006) y más adelante en Zitzler, Knowles y Thiele (2008). Parte de esta notación, que es necesaria para comprender este trabajo de tesis, será introducida en los párrafos subsiguientes.

Dominación fuerte (o estricta): Una solución x_1 domina fuertemente o estrictamente a otra solución x_2 ($x_1 \prec\prec x_2$) si:

$$f_j(x_1) \quad \lhd \quad f_j(x_2) \; \forall \; j = 1, 2, \ldots, M;$$
x_1 es mejor que x_2 para los valores de todos los objetivos.

La Figura 1.3 muestra un ejemplo de dominación fuerte. En ella la solución S_1 domina fuertemente a las demás soluciones. El área gris representa el área que es dominada por la solución S_1, cualquier solución que caiga dentro de ese área será fuertemente dominada por S_1, excepto las soluciones que se ubiquen en el límite inferior o el límite lateral izquierdo, que serán débilmente dominadas.

Dominación: Una solución x_1 domina a otra solución x_2 ($x_1 \prec x_2$) si se cumplen las siguientes condiciones:

(1) $f_j(x_1) \quad \not\lhd \quad f_j(x_2) \; \forall \; j = 1, 2, \ldots, M;$
x_1 no es peor que x_2 para los valores de todos los objetivos
(2) $f_j(x_1) \quad \lhd \quad f_j(x_2)$ para al menos un $j = 1, 2, \ldots, M$
x_1 es mejor que x_2 para al menos el valor de un objetivo

Dominación débil: Se dice que la solución x_1 domina débilmente a otra

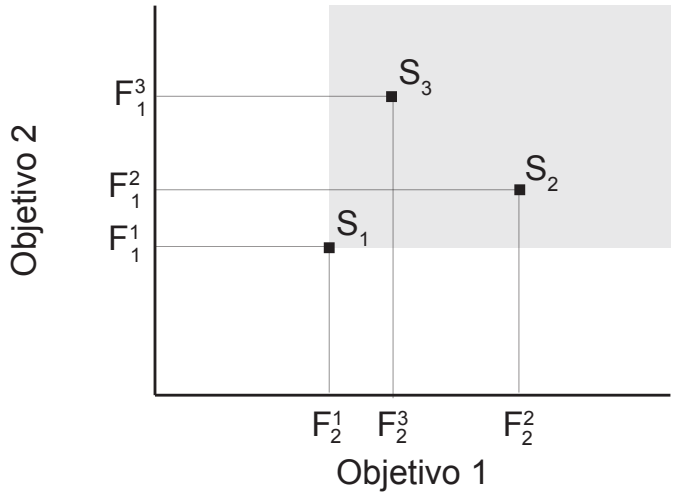


Figura 1.3: Ejemplo de dominación fuerte o estricta para un entorno de dos objetivos. La solución S_1 domina fuertemente a las soluciones S_2 y S_3.

solución x_2 ($x_1 \preceq x_2$)si:

$$f_j(x_1) \quad \not\succ \quad f_j(x_2) \; \forall \; j = 1, 2, \ldots, M;$$

x_1 no es peor a x_2 para todos los valores de los objetivos.

Soluciones incomparables: Las soluciones x_1 y x_2 son incomparables ($x_1 \| x_2$ o $x_2 \| x_1$) cuando:

$$f_j(x_1) \not\preceq f_j(x_2) \text{ ni } f_j(x_2) \not\preceq f_j(x_1) \; \forall \; j = 1, 2, \ldots, M$$

Estas definiciones pueden ser trasladadas a las relaciones entre conjuntos de soluciones. Siendo A y B dos conjuntos de soluciones para un problema de optimización multi-objetivo (en inglés MOOP), definiremos:

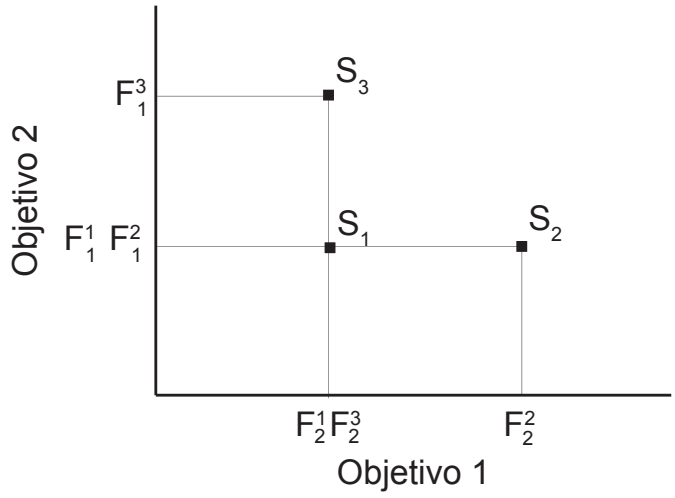


Figura 1.4: Ejemplo de dominación débil. La solución S_1 domina débilmente a las soluciones S_2 y S_3.

Dominación fuerte (o estricta): El conjunto A domina fuertemente o estrictamente al conjunto B ($A \prec\prec B$) si:

$$\text{Todo } x_i \in B \succ\succ \text{ por al menos un } x_j \in A$$

Dominación: El conjunto A domina al conjunto B ($A \prec B$) si:

$$\text{Todo } x_i \in B \succ \text{ por al menos un } x_j \in A$$

Mejor: El conjunto A es mejor que el conjunto B ($A \triangleleft B$) si:

$$\text{Todo } x_i \in B \succeq \text{ por al menos un } x_j \in A \text{ y } A \neq B$$

Dominación débil: A domina débilmente a B ($A \succeq B$) si:

$$\text{Todo } x_i \in B \succeq \text{ por al menos un } x_j \in A$$

Incomparabilidad: A es incomparable al conjunto B ($A \| B$) si:

$$\text{No se cumple que } A \succeq B \text{ ni } B \succeq A$$

Conjunto no dominado: Entre un conjunto de soluciones A, se define el subconjunto de soluciones no dominadas A' tal que se cumpla que: $x^{A'} \in A' \prec x^A$ con $x^{A'} \neq x^A$ y $A' \subset A$.

Solución óptima global de Pareto: una solución $x \in A$ (siendo A el conjunto de todas las soluciones factibles) es una solución optima global de Pareto si y solo si no existe una solución $x' \in A$ de manera que $f(x') \prec f(x)$.

Conjunto óptimo global de Pareto: un conjunto $A' \in A$ (siendo A un conjunto de soluciones de un problema) es un conjunto óptimo global de Pareto si y solo si contiene todas las soluciones óptimas globales de Pareto. Este conjunto suele ser llamado también frontera de Pareto.

A diferencia de la optimización de un solo objetivo, en la optimización multi-objetivo se persigue obtener un conjunto de valores. Este conjunto de valores debe conformar una aproximación al verdadero conjunto de Pareto global y además debe estar conformado por soluciones bien dispersas. En otras palabras, una solución es considerada buena si cubre gran parte del conjunto de soluciones de la frontera de Pareto.
Debido a la naturaleza $\mathcal{NP}-$Difícil del problema del taller de flujo de permutación (PFSP) para los tres objetivos anteriormente comentados ($C_{\text{máx}}$, Tardanza Total y Tiempo Total de Flujo), puede observarse que en un entorno multi-objetivo, el problema resultante será también $\mathcal{NP}-$Difícil para cual-

quier combinación de estos objetivos, sin importar si se lo resuelve de manera *a priori* o *a posteriori*.

1.1.2. Métricas de calidad en algoritmos multi-objetivo

La comparación de dos (o más) fronteras de Pareto provenientes de distintos algoritmos no es una cuestión simple. Dos conjuntos de soluciones (llamados fronteras de Pareto o mejor aún aproximaciones a la frontera de Pareto) A y B pueden ser incluso incomparables. Estudios como los llevados a cabo por Zitzler et al. (2003), Paquete (2005) o Knowles, Thiele y Zitzler (2006) son un claro ejemplo del enorme esfuerzo que se está llevando a cabo para proveer a los investigadores de mejores herramientas para la evaluación y comparación de algoritmos multi-objetivo. Sin embargo, en la literatura multi-objetivo para el taller de flujo de permutación se utilizan con frecuencia métricas de calidad que han mostrado ser erróneas, incongruentes e incorrectas. Por ejemplo, en dos artículos de entre los revisados (Rahimi-Vahed y Mirghorbani, 2007 y Geiger, 2007) se utilizan algunas métricas como la distancia generacional o la desviación máxima de la mejor frontera encontrada. En el estudio de Knowles, Thiele y Zitzler (2006) se demuestra que estas métricas, entre otras, no son Pareto compatibles. Esto quiere decir que en algunos casos pueden dar un mejor valor para una aproximación a la frontera de Pareto B y un peor valor para la frontera A, incluso en el caso donde $A \prec B$.

La Figura 1.5 muestra como un método de medición que no es Pareto compatible puede dar indicaciones erróneas e inconsistentes. En este caso el indicador es el error proporcional o *error ratio*, que mide el error entre la frontera de Pareto y un conjunto de soluciones o frontera. Debido a que la solución S_1 tiene uno de sus puntos en la frontera de Pareto, el error proporcional para esta solución será $1/3$. Por otro lado, S_2 tiene 10 puntos, solo uno de ellos está en la frontera de Pareto. Sin embargo los otros nueve valores están muy cerca de la frontera de Pareto y además dominan fuertemente al segundo punto de

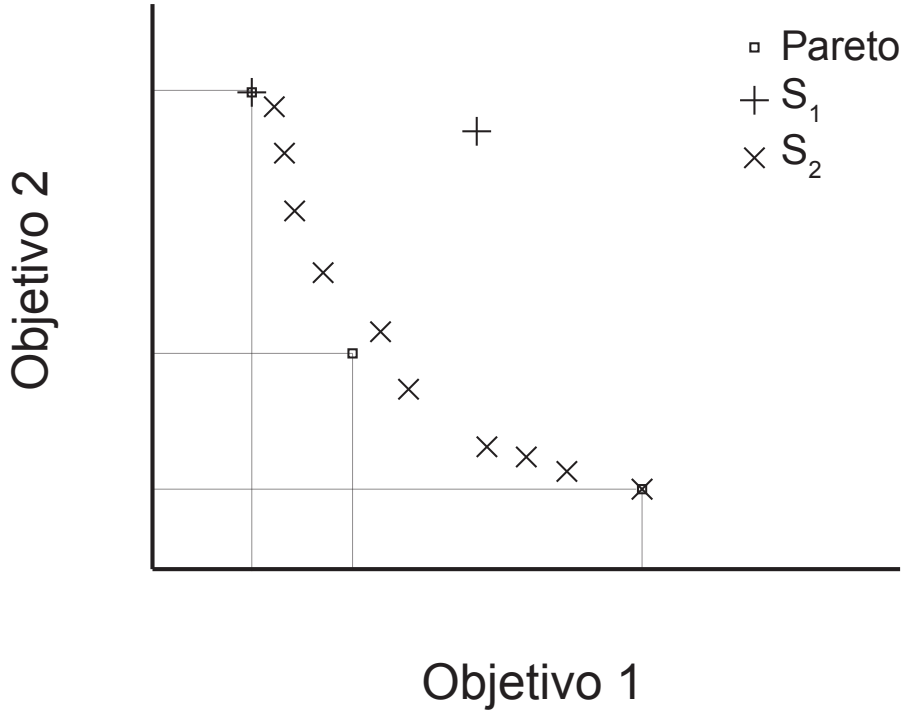


Figura 1.5: Gráfica demostrando un método de medición no Pareto compatible. La frontera P es la frontera de Pareto, S_1 y S_2 son dos conjuntos de soluciones. Ambos conjuntos de soluciones solamente se encuentran con la frontera de Pareto en un punto cada uno, sin embargo hay métodos de medición que indican que S_1 es mejor que S_2 cuando eso no es correcto.

S_1. A pesar de todo ello, el error proporcional de la segunda solución será $9/3$, con lo cual esté método de medición indica que S_1 es mejor que S_2.

En Knowles, Thiele y Zitzler (2006) y en Zitzler, Knowles y Thiele (2008) se llevan a cabo evaluaciones empíricas de las métricas de calidad más utiliza-

das. Aquí se señala también que las técnicas de medición con mayor frecuencia utilizadas suelen no ser Pareto compatibles. Todo esto indica que a la hora de elegir las métricas de calidad, utilizadas para medir los resultados de un trabajo de investigación o el desarrollo de un nuevo algoritmo, se debe prestar especial atención para asegurar que los resultados sean sólidos y generalizables. La Tabla 1.2 muestra ejemplos de las algunas de métricas más utilizadas en la literatura indicando cuales son Pareto compatibles y cuales no.

Nombre	Ventajas	Desventajas	Pareto Compatible
Cobertura	Fácil de entender Fácil de calcular	Binario No da información sobre la diversidad No da idea de la distancia entre fronteras	No
Distribución	Fácil de entender	Binario Requiere conocer los valores extremos para los objetivos Requiere una aproximación a la frontera de Pareto	No
Distancia generacional invertida	Fácil de entender Fácil de calcular Mide convergencia y diversidad	Requiere una aproximación a la frontera de Pareto	No
Distancia generacional	Fácil de entender Fácil de calcular Mide convergencia	Requiere una aproximación a la frontera de Pareto	No
Indicador épsilon	Mide convergencia	Sólo tiene en cuenta la peor solución Requiere una aproximación a la frontera de Pareto	Si
Indicador de hipervolumen	Mide convergencia y diversidad No requiere conocer Pareto		Si
Número de soluciones en frontera de Pareto	Fácil de calcular	Requiere una aproximación a la frontera de Pareto No aporta información sobre la distribución	No

Tabla 1.2: Métricas de medición mu1lti-objetivo comúnmente utilizadas en la literatura.

Knowles, Thiele y Zitzler (2006) y luego Zitzler, Knowles y Thiele (2008) proponen tres enfoques demostrando que son pareto compatibles. El primero de ellos depende de las relaciones de dominancia entre conjuntos de soluciones. Es posible comparar dos algoritmos basándose en la cantidad de veces que las soluciones pertenecientes a la aproximación a la frontera de Pareto de uno de ellos, dominan (en el sentido fuerte, normal o débil) a las soluciones del otro algoritmo o no.
El segundo enfoque propuesto depende de indicadores de calidad. Un indicador es una metodología por la cual se asigna un valor numérico a una frontera de Pareto. De esta forma, se puede comparar la bondad de varias fronteras mediante la comparación de sus respectivos indicadores.
Muchos indicadores han sido propuestos en la literatura, de ellos los más relevantes son el hipervolumen (I_H) y el indicador épsilon (I_ε) que han sido ya introducidos en Zitzler y Thiele (1999) y Zitzler et al. (2003). Estos indicadores de calidad, por lo general, transforman una aproximación a la frontera de Pareto en un número real.
Por último, el tercer enfoque está basado en las *Empirical Attainment Functions* o funciones empíricas de éxito (abreviado como EAF). Las EAF muestran, en términos del espacio de los objetivos, la frecuencia relativa en que cada región de este espacio es alcanzada por una aproximación a la frontera de Pareto obtenida por un algoritmo. La generación de las EAF requiere de un cálculo intensivo, y además, para obtener resultados aceptables se necesita una gran cantidad de soluciones (o sea aproximaciones a la frontera de Pareto) para cada algoritmo a comparar.
Estos tres enfoques abarcan desde técnicas directas y fáciles de calcular, como las de clasificar por dominancia, hasta las técnicas que requieren de una computación intensiva y son más difíciles de aplicar, como es el caso de las funciones de éxito.

La clasificación por dominancia se puede utilizar en algunos casos par-

ticulares, cuando existen claras diferencias entre fronteras generadas por un algoritmo y las generadas por el otro. Además de ello, este tipo de clasificación es aplicable sólo en el caso de comparar dos algoritmos. Para realizar la comparación se debe contar la cantidad de veces que las soluciones de un algoritmo dominan fuertemente, de manera regular o débilmente a las soluciones del otro algoritmo. En nuestro caso, como se llevará a cabo la comparación de más de 20 algoritmos, la cantidad de pares de algoritmos a comparar superará los 400 lo que hace impracticable la aplicación de estas técnicas. Para la comparación de muchas soluciones, sobre todo cuando las soluciones provienen de más de dos algoritmos, es recomendable utilizar indicadores.

Existe un tipo de indicadores llamados indicadores binarios y que solo permiten la comparación entre fronteras de dos algoritmos diferentes. En este caso, al igual que al aplicar la clasificación por dominancia, la aplicación de este tipo de indicadores tampoco es factible debido a que se comparan algoritmos de a dos. Por estos motivos, en este trabajo, hemos seleccionado el indicador unario de hipervolumen I_H^1 y el indicador unario multiplicativo épsilon I_ε^1 para realizar las comparativas entre algoritmos.
El mismo problema existe al intentar utilizar las EAF, ya que éstas también exigen que la comparación de algoritmos sea llevada a cabo de a pares. Además de esto, el costo computacional de las EAF es muy elevado y los resultados deben ser examinados gráficamente uno por uno. Como conclusión este tipo de análisis no es útil para el caso de comparar varios algoritmos entre sí. En cambio, es de gran utilidad a la hora de realizar una comparación exhaustiva entre los resultados de dos algoritmos, una vez que se ha demostrado, utilizando otros métodos de comparación, que sus resultados son demasiado cercanos entre sí como para sacar una conclusión sólida.
De acuerdo con Knowles, Thiele y Zitzler (2006), I_H^1 e I_ε^1 son Pareto compatibles y son unas de las métricas de calidad más modernas y adecuadas. Además, la combinación del análisis de estos dos indicadores es un enfoque

muy acertado, ya que si ambos indicadores muestran conclusiones contradictorias, indican que la fronteras de Pareto obtenidas de los algoritmos son incomparables. En las siguientes líneas se explicarán los detalles del cálculo de estos indicadores.

1.1.2.1. Indicador épsilon e indicador de hipervolumen

El indicador de hipervolumen I_H^1 aparece por primera vez en Zitzler y Thiele (1999). Mide el area (en caso de dos objetivos) que cubre una aproximación a la frontera de Pareto dada por un algoritmo. Para delimitar el área se utiliza un punto de referencia para cada objetivo. Un valor más grande de I_H^1 indica al mismo tiempo una mejor convergencia y un mejor cubrimiento de la frontera de Pareto óptima. Calcular el hipervolumen puede ser una tarea costosa, en este trabajo se utiliza el algoritmo propuesto en Deb (2001). Este algoritmo calcula un valor normalizado y escalado. Por lo que el hipervolumen para la mejor frontera encontrada será el más cercano al valor de referencia seleccionado y todas las demás aproximaciones tendrán un valor menor. El Algoritmo 1 muestra el pseudocódigo para cálculo del hipervolumen. Como requisito para calcular el hipervolumen de un conjunto de soluciones o $Frontera$ se necesita una frontera de referencia (llamada $Pareto$ en el algoritmo) conformada por las mejores soluciones conocidas y un punto de referencia $S_{R,O}$ para cada objetivo O.

Algoritmo 1 Cálculo del hipervolumen.

parametros: $Frontera; Pareto; S_{R,O}$
  $LastValue_1 = S_{R,1}$ //variable auxiliar que se inicializa con el punto de referencia para el primer objetivo
  para todo $S_i \in Frontera$ **hacer**
    $Hipervolumen_{Frontera} = Hipervolumen_{Frontera} + ((LastValue_1 - S_{i,1} * (S_{R,2} - S_{i,2}))$
    $LastValue_1 = S_{i,1}$ //Actualizar la variable auxiliar
  fin para

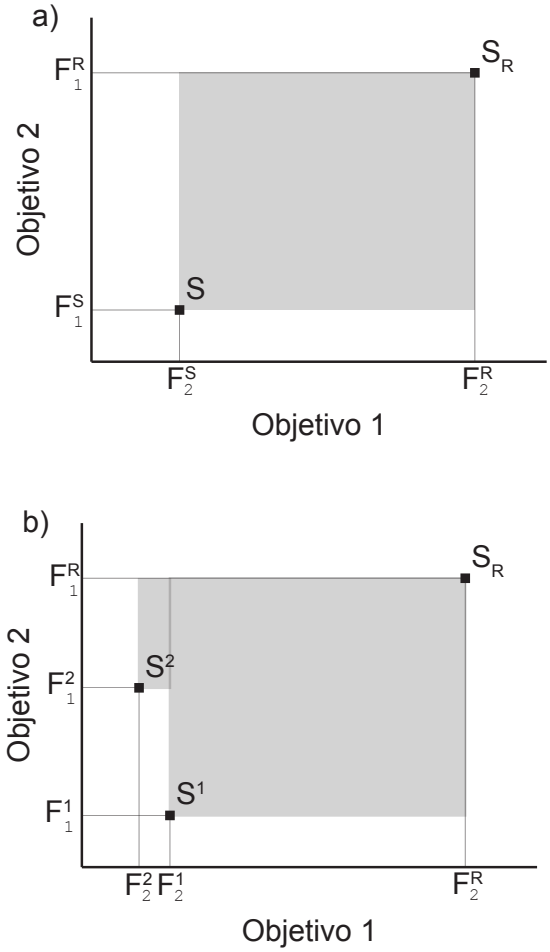


Figura 1.6: Cálculo del hipervolumen en el caso de problemas de dos objetivos. a) Ejemplo de cálculo del hipervolumen cuando la frontera de Pareto está compuesta por una única solución S. S_R es el punto de referencia. Gráfica b) indicación de cómo se agrega el área de una segunda solución al cálculo.

La Figura 1.6 muestra de forma simplificada y gráfica cómo se calcula el hipervolumen. En este caso al tener dos soluciones en la frontera. Cuando hay una única solución el hipervolumen se obtiene a través del área que hay entre la solución S y el punto de referencia S_R. Al añadir una segunda solución S_2 el hipervolumen se calcula añadiendo la parte suplementaria del área entre S_2 y el punto ficticio formado por F_1^1 y F_2^R. En los últimos años han aparecido en la literatura varios métodos de cálculo del hipervolumen que son más veloces y para más objetivos. Algunos de ellos han sido propuestos por While et al. (2005), luego por While et al. (2006), y más adelante por Zitzler, Brockhoff y Thiele (2007) y Bradstreet, While y Barone (2008).

El indicador binario épsilon I_ε propuesto inicialmente por Zitzler et al. (2003), y extendido y probado exhaustivamente luego en Zitzler, Knowles y Thiele (2008), se calcula de la siguiente manera: dadas dos aproximaciones a la frontera de Pareto A y B, el indicador binario multiplicativo épsilon $I_\varepsilon(A, B)$ será igual a $\max_{x_B} \min_{x_A} \max_{1 \leq j \leq M} \frac{f_j(x_A)}{f_j(x_B)}$, donde x_A y x_B representan a cada una de las soluciones existentes en los conjuntos A y B respectivamente. En otras palabras se puede decir que la solución A $\varepsilon-$domina a la solución B si se puede encontrar un valor ε tal que, luego de multiplicar cada uno de los valores de los objetivos de la solución B por ese ε, todavía se puede decir que $A \succeq B$ (A domina débilmente a B).
Este tipo de indicador binario debe ser calculado para todos los posibles pares de algoritmos. Sin embargo, en Knowles, Thiele y Zitzler (2006), se propone un nueva versión unaria de este indicador, el indicador I_ε^1, en el que la segunda aproximación a ser comparada (B) es reemplazada por la mejor frontera de Pareto conocida. Este es un indicador muy interesante debido a que mide la distancia o diferencia (ε) entre una aproximación a la frontera de Pareto y la mejor frontera conocida. Por lo que ε da una idea clara y directa del rendimiento. Cabe destacar, sin embargo, que algunos objetivos (como la tardanza) pueden dar un valor igual a cero. De todas maneras también es

necesario normalizar los valores de los objetivos ya que las magnitudes pueden variar de un objetivo a otro. Por lo tanto, para poder calcular el indicador I_ε^1 primero hay que normalizar los valores obtenidos reemplazando los valores del cálculo previo $f_j(x_A)$ y $f_j(x_B)$ por $\frac{f_j(x_A)-f_j^-}{f_j^+-f_j^-}+1$ y $\frac{f_j(x_B)-f_j^-}{f_j^+-f_j^-}+1$, donde f_j^+ y f_j^- son los valores máximo y mínimo para un objetivo dado j, respectivamente. Como resultado, el indicador normalizado I_ε^1 toma valores entre 1 y 2. Un valor cercano a 1 indica que una aproximación a la frontera de Pareto está muy cerca a la mejor frontera encontrada. Aplicado a la comparación de varias fronteras, la mejor será aquella cuyo valor de I_ε^1 sea el menor.
En la siguiente sección profundizaremos sobre el cálculo y utilización de las EAF, que requieren un estudio más profundo debido a su complejidad.

1.1.2.2. *Empirical Attainment Functions*

Las funciones de éxito o en inglés *Attainment Functions* fueron propuestas inicialmente por Grunert da Fonseca, Fonseca y Hall (2001) como una herramienta que permite determinar que probabilidad tiene un algoritmo de generar soluciones en el espacio de los objetivos.
La definición formal para las EAF se expresa de la siguiente manera. Sea $x \in \mathbb{R}^d$ un vector arbitrario perteneciente a un espacio de soluciones de d objetivos y $S = \{s_j \in \mathbb{R}^d, j = 1, \ldots, P\}$ un conjunto de soluciones compuesto de soluciones no dominadas, comúnmente llamado frontera. Donde P representa a cada uno de los puntos presentes en dicha frontera. Las funciones de éxito se caracterizan por $AF(x) = P(\exists s_j \in S\colon s_j \trianglelefteq x)$, que describe la probabilidad para un algoritmo de producir, en una ejecución, al menos una solución que domine débilmente a x.
En el caso de algoritmos estocásticos no es posible expresar esta función de una manera cerrada, pero sí se puede aproximar empíricamente usando las aproximaciones a la frontera de Pareto obtenidas por varias ejecuciones del algoritmo. Esta aproximación se llama *Empirical Attainment Functions* o

EAF y ha sido definida en Grunert da Fonseca, Fonseca y Hall (2001) como:

$$EAF(x) = \frac{1}{k} \sum_{i=1}^{k} I(S_i \trianglelefteq x) \tag{1.2}$$

donde $S_1, S_2, \ldots, S_i, \ldots, S_k$ representan las k aproximaciones a la frontera de Pareto obtenidas mediante k ejecuciones independientes del algoritmo.
Más adelante, López-Ibáñez, Paquete y Stützle (2006) proponen una extensión para las EAF que permite comparar, siempre mediante gráficas, las soluciones de dos EAF al mismo tiempo. Con esta herramienta se pueden ver qué sectores del espacio de los objetivos son cubiertos con más probabilidad por un algoritmo o el otro de manera excluyente. La Figura 1.7 muestra de manera simplificada la manera de generar una gráfica de EAF partiendo, en este caso, de cuatro gráficas. Cada una de estas cuatro gráficas representan una frontera de Pareto generada por la ejecución de un algoritmo. La gráfica de EAF se genera mediante la superposición de los resultados de las cuatro ejecuciones del algoritmo. En una EAF real, la única diferencia es que se necesitan los resultados de muchas más ejecuciones, en la literatura se recomiendan 50 o más.

Una gran desventaja de las EAF es la necesidad de cálculo intensivo, además cada gráfico de EAF solamente representa los resultados de un algoritmo para una sola instancia de problemas. Para poder generar una sola gráfica es necesario obtener, para la misma instancia y el mismo algoritmo, una gran cantidad de réplicas. Esto es, se debe ejecutar el mismo algoritmo una cantidad elevada de veces contra la misma instancia y recopilar todas las aproximaciones a la frontera de Pareto obtenidas en cada experimento.

En esta sección hemos hecho un repaso de las técnicas de medición y comparación de los resultados obtenidos por algoritmos en problemas multiobjetivo. También hemos identificado algunas de las métricas de calidad más

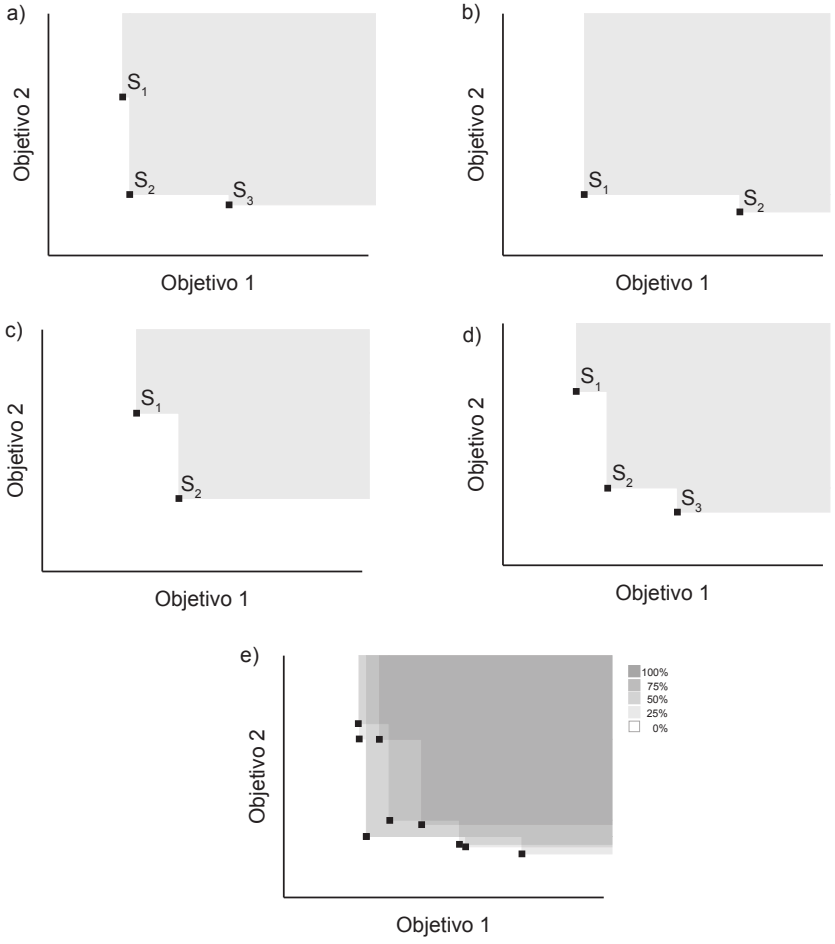


Figura 1.7: Ejemplo de conformación de una gráfica de Empirical Attainment Functions. Las gráficas a), b), c), y d) representan el resultado de cuatro ejecuciones de un algoritmo estocástico. La gráfica e) es la gráfica de EAF generada a partir de las primeras cuatro gráficas.

utilizadas que son Pareto compatibles y otras que no lo son.
Debido a las características de los experimentos que llevaremos a cabo en capítulos posteriores, hemos seleccionado tres métricas de calidad para comparar resultados. En primer lugar, utilizaremos una combinación de dos indicadores unarios, el indicador de hipervolumen I_H^1 y el indicador unario épsilon I_ε^1, que en adelante representaremos como I_H y como I_ε por motivos de simplificación. Estos indicadores utilizados en combinación permiten identificar claramente si una aproximación a la frontera de Pareto es mejor que otra o si son incluso incomparables. Más adelante compararemos el mejor algoritmo resultante de la revisión bibliográfica del Capítulo 3 contra el algoritmo que proponemos en el Capítulo 4. Para realizar una comparación más exhaustiva, y debido a que será una comparación de dos algoritmos solamente, también utilizaremos gráficas de EAF.

1.2. Objetivos de la presente tesis doctoral

En las últimas dos décadas se han propuesto muchos algoritmos interesantes para resolver distintos problemas multi-objetivo. Sin embargo estos algoritmos, por lo general, no han sido validados contra otros algoritmos existentes en la literatura. Además de ello, en muchas ocasiones, los indicadores de calidad de los resultados comúnmente utilizados no han sido los apropiados. Muchos de estos algoritmos no han sido aplicados aún al problema del taller de flujo multi-objetivo. En consecuencia, actualmente no existe una revisión profunda de acerca de algoritmos propuestos para el taller de flujo multi-objetivo, ni una comparativa de estos algoritmos contra otros algoritmos multi-objetivo genéricos o específicos para otros problemas.
En la actualidad se siguen proponiendo nuevos métodos multi-objetivo con enfoques muy interesantes y prometedores. Pero aún hoy los autores no cuentan con una relación clara y objetiva de los algoritmos existentes, ni con una

comparativa sólida que les permita identificar claramente cual método de la literatura es el estado de arte actual. Muchos autores siguen comparando sus trabajos utilizando algoritmos propuestos hace 20 años. Por otro lado, se han propuesto muchos bancos de pruebas diferentes a lo largo del tiempo. Cada banco de pruebas se ha basado generalmente en un conjunto de instancias distinto, con diferentes combinaciones de objetivos y utilizando distintas métricas de calidad que, como vimos antes, muchas veces no son Pareto compatibles.

A la hora de proponer un nuevo método para resolver un problema es importante saber el estado actual en cuanto a que otros algoritmos han sido propuestos antes, o incluso que otros métodos propuestos para resolver otros problemas podrían ser aplicados al problema que queremos resolver. Claramente, uno de los temas pendientes en cuanto al problema del taller de flujo de permutación multi-objetivo es una comparación clara y precisa de todos los algoritmos que en la actualidad claman ser estado del arte en este problema, utilizando un único banco de pruebas, con los mismos objetivos, ejecutado en los mismos ordenadores y con los mismos criterios de parada. También es necesario añadir a esta comparativa algoritmos multi-objetivo genéricos que nunca han sido aplicados aun al taller de flujo de permutación y que, por sus características, resultan prometedores. Parte importante de esta comparación es también proponer un banco de pruebas que se pueda reutilizar y que incluya una o más métricas de calidad que sean Pareto compatibles.
Los objetivos de esta tesis son: realizar una revisión profunda y análisis de los algoritmos existentes propuestos para el taller de flujo multi-objetivo. Al mismo tiempo, adaptar métodos multi-objetivo generales al problema del taller de flujo de permutación. Para ello hemos identificado los métodos multi-objetivo más prometedores, realizando la evaluación de 25 métodos que incluyen diferentes metodologías, desde búsquedas locales, búsquedas tabú, recocido simulado, pasando por algoritmos evolutivos, como los algoritmos genéticos.
Otro objetivo que surge indirectamente del primero es determinar un conjunto

de indicadores de calidad más adecuados para la comparación o medición de los resultados de métodos multi-objetivo. Para ello hemos realizado un estudio sobre las métricas existentes y su utilización así también como las posibles ventajas y desventajas de las mismas.
Finalmente proponemos una nueva metaheurística multi-objetivo basada en la metaheurística *Iterated Greedy* (o algoritmo voraz iterado). Esta metaheurística es comparada con los mejores métodos resultantes de la revisión bibliográfica anteriormente citada.
En un paso ulterior aplicaremos esta misma metaheurística a problemas de taller de flujo más realistas, como son el taller de flujo de permutación multi-objetivo con tiempos de cambio dependientes de la secuencia. Finalmente, también aplicaremos una variación de esta misma técnica, al caso de los problemas de taller de flujo híbrido. Éste se diferencia del taller de flujo de permutación, entre otras cosas, en que cada máquina tiene su propia secuencia de proceso de los trabajos.

1.3. Organización de esta Tesis

En el Capítulo 2 llevamos a cabo una completa revisión bibliográfica de algoritmos multi-objetivo específicamente propuestos para problemas de taller de flujo y también de algoritmos multi-objetivo generales.
A partir de la revisión bibliográfica, realizamos una evaluación y comparación de los algoritmos en el Capítulo 3. Describimos luego el banco de pruebas y los resultados experimentales. Finalmente, obtendremos una clasificación de los algoritmos de acuerdo a su rendimiento para el conjunto de pruebas propuesto.
En el Capítulo 4 proponemos un nuevo método multi-objetivo diseñado para resolver los problemas de taller de flujo de permutación. Se describe también su construcción y se llevan a cabo una serie de experimentos para calibrarlo en la sección 4.2.1.
Más adelante, introduciremos los problemas de taller de flujo con restricciones

más realistas. En primer lugar están los problemas de taller de flujo de permutación con tiempos de cambio. El Capítulo 5 describe este tipo de problemas detallando sus características particulares y su aplicación en la vida real. En él, también adaptamos el algoritmo propuesto en el Capítulo 4 resolver este tipo de problemas. Luego detallaremos una batería de pruebas y una comparativa contra los mejores algoritmos obtenidos del Capítulo 3 y contra los algoritmos existentes propuestos para este tipo de problemas. Al final del capítulo presentamos los resultados de estas pruebas.

El Capítulo 6 describe la aplicación del algoritmo propuesto para el problema de taller de flujo híbrido. En este caso también presentamos una comparativa entre este algoritmo y otros algoritmos propuestos para este tipo de problemas. Las conclusiones de la comparativa se muestran al final del capítulo.

Finalmente, en el Capítulo 7 presentamos las conclusiones de este trabajo junto con las futuras líneas de investigación posibles.

Luego de las conclusiones, en los apéndices, se puede encontrar información relevante a los problemas tratados, como así también gráficas y tablas de todos los experimentos, que por razones de espacio y claridad no hemos añadido en su totalidad en los capítulos correspondientes.

CAPÍTULO 2

REVISIÓN BIBLIOGRÁFICA

En las últimas décadas se ha prestado mayor atención a los problemas multi-objetivo. En este contexto se han desarrollado múltiples técnicas y métodos para la resolución de este tipo de problemas. A pesar de ello, como ya hemos comentado anteriormente, comparada con la literatura para un solo objetivo, la literatura multi-objetivo está aún por desarrollarse. Si prestamos especial atención al problema del taller de flujo de permutación multi-objetivo, la existencia de literatura es más bien escasa. Los pocos métodos propuestos para resolver el problema de taller de flujo de permutación utilizan optimización evolutiva o búsquedas locales como recocido simulado o búsqueda tabú, entre otras.

Existen varias revisiones en la literatura acerca de la programación multi-objetivo, sin embargo no se ha prestado demasiada atención a los problemas de programación de talleres de flujo con varios objetivos, a pesar de que existen innumerables publicaciones que tratan los problemas de taller de flujo con un solo objetivo. Por ejemplo la revisión de algoritmos multi-objetivo escrita por

Nagar, Heragu y Haddock (1995b) está centrada en problemas para una sola máquina, incluso hay solamente citados cuatro artículos que se refieren a los problemas de taller de flujo.
En otra revisión hecha por T'Kindt y Billaut (2001) podemos encontrar referencias a solo 15 artículos que tratan el problema del taller de flujo multi-objetivo, y además estos artículos se refieren en su mayoría a el caso específico de taller de flujo con dos máquinas. Otra revisión hecha por Jones, Mirrazavi y Tamiz (2002) consiste más bien en un recuento de artículos de optimización multi-objetivo.
Más adelante, existe otra revisión bibliográfica de optimización multi-objetivo propuesta en Hoogeveen (2005) que contiene principalmente referencias a artículos que tratan los problemas de optimización multi-objetivo para una máquina o para máquinas paralelas. Incluso los artículos citados en esta revisión que se refieren al problema del taller de flujo están restringidos a los problemas de dos máquinas en su mayoría. Por estas razones se hace necesaria una revisión más amplia que incluya los problemas de optimización en talleres de flujo multi-objetivo con más de dos máquinas.
La revisión bibliográfica más moderna referente al taller de flujo de permutación multi-objetivo se encuentra en Mutlu Yenisey y Yagmahan (2013). En este artículo los autores realizan una revisión la literatura existente sobre el problema citado y realizan una clasificación de la tipología y características de los distintos algoritmos utilizados en la resolución del problema. Cabe destacar que este artículo solamente realiza una descripción y clasificación de metodologías, objetivos y tipos de problemas estudiados en la literatura, no realiza ningún tipo de comparativa o estudio que permita determinar la originalidad o siquiera los mejores resultados obtenidos en la literatura.

En adelante adoptaremos el esquema de notación propuesto por Graham et al. (1979), y su extensión propuesta por T'Kindt y Billaut (2002), para especificar la técnica y objetivos estudiados en cada uno de los artículos revi-

sados. En esta notación se especifican los problemas multi-objetivo mediante tres campos: $\alpha|\beta|\gamma$. El primer campo α describe la topología del problema, y puede incluir dos subcampos α_1 y α_1, que describen el entorno de máquinas del problema y el número de máquinas disponibles. La tipología de problema que estudiaremos en este trabajo de tesis es el flowshop que se representa mediante F. El segundo campo β contiene restricciones explícitas del problema. Por ejemplo $prmu$ representa permutación e indica que las operaciones en las máquinas se llevarán a cabo en el mismo orden para todas las máquinas. Finalmente el tercer campo γ, indica el o los objetivos a utilizar en la resolución del problema. Este campo puede contener un único valor Z cuando se trata de problemas de un único objetivo, o varios valores $Z_1, \ldots, Z_K$ para optimización multi-objetivo. Dependiendo del enfoque de optimización multi-objetivo también puede tomar diferentes valores:

- $F_\ell(Z_1, \ldots, Z_K)$ cuando el objetivo del problema es optimizar una combinación lineal de los objetivos. En este caso se debe asignar una ponderación o peso a cada objetivo para resolver el problema.

- $\epsilon(Z_u/Z_1, \ldots, Z_u - 1, Z_u + 1, \ldots, Z_K)$ indica que solamente se optimiza el criterio Z_u, mientras que los demás criterios están sujetos a un límite superior o *upper bound*. Este caso se suele denominar enfoque ϵ-restringido.

- $GP(Z_1, Z_2, \ldots, Z_K)$ si existen metas (*goals*) que alcanzar para cada criterio. Este tipo de problema se llama comúnmente programación por metas o *goal programming*. El objetivo en este caso no es optimizar los criterios sino encontrar una solución que satisfaga las metas, incluso cuando esa solución no pertenezca a la frontera de Pareto.

- $Lex(Z_1, Z_2, \ldots, Z_K)$ en este caso los criterios vienen dados en cierto orden de importancia (el primero es el de mayor importancia), no existen soluciones de compromiso (*trade off*) sino que la bondad de una solu-

ción se mide por los valores de los objetivos en el orden dado. Este tipo de enfoque se denomina ordenación lexicográfica.

- $\#(Z_1, Z_2, \ldots, Z_K)$ indica la optimización Pareto, también llamada *a posteriori* en la cual se intenta obtener todas las soluciones que representen un compromiso o *trade off* entre los distintos valores de los objetivos. El resultado de estos problemas suele ser un conjunto de soluciones en vez de una única solución al que se le denomina frontera de Pareto.

Por ejemplo un problema de dos objetivos ponderados para el taller de flujo de permutación con los objetivos de $C_{\text{máx}}$ y tardanza total se representa de la siguiente manera: $F/prmu/F_\ell(C_{\text{máx}}, T)$. Debido a que no es un tema a tratar en esta tesis no profundizaremos más sobre la notación, excepto en los casos que sea necesario para comprender alguna parte de esta tesis. Para más detalles el lector puede consultar: Graham et al. (1979), T'Kindt y Billaut (2001) o T'Kindt y Billaut (2002).

2.1. Enfoque lexicográfico y enfoque ϵ-restringido

Uno de los enfoques estudiados para resolver problemas multi-objetivo ha sido el enfoque lexicográfico. Daniels y Chambers (1990) propusieron una heurística constructiva para un taller de flujo con m máquinas donde el $C_{\text{máx}}$ debe ser minimizado sujeto a un valor límite de tardanza. Este tipo de problema se representa como $F/prmu/\epsilon(C_{\text{máx}}/T_{\text{máx}})$. Esta heurística junto con la propuesta por Chakravarthy y Rajendran (1999) son comparadas con un nuevo método propuesto por Framiñan y Leisten (2006). En éste artículo se muestra que el nuevo método propuesto mejora los resultados obtenidos en los artículos anteriores de Daniels y Chambers (1990) y Chakravarthy y Rajendran (1999), tanto en cantidad como en calidad de soluciones factibles encontradas. En el artículo de Rajendran (1992) se consideran otros objetivos, el tiempo total de flujo sujeto al $C_{\text{máx}}$ óptimo, en un entorno de taller de flujo con dos

máquinas. Este tipo de aproximación es válida para el problema de taller de flujo de permutación, ya que el $C_{\text{máx}}$ óptimo puede ser obtenido aplicando el algoritmo de Johnson (1954).
Un algoritmo *branch & bound* (B&B), junto con varias heurísticas son propuestos por Rajendran (1992), sin embargo estos métodos son capaces de resolver problemas de a lo sumo 24 trabajos.

En el artículo de Neppalli, Chen y Gupta (1996) se presentan dos algoritmos genéticos para resolver el problema de taller de flujo con dos objetivos y dos máquinas, también utilizando el enfoque lexicográfico como en Rajendran (1992). El primero de los algoritmos está basado en el VEGA (*Vector Evaluated Genetic Algorithm*) de Schaffer (1985). En éste, se mantienen dos poblaciones (una para cada objetivo) que se combinan a través de un operador de selección para obtener nuevas soluciones. El segundo algoritmo utiliza un enfoque de criterios ponderados, en el cual se considera una combinación lineal de dos criterios. Esta suma ponderada de los dos objetivos se utiliza como valor de medida (*fitness*). El mismo problema es estudiado por Gupta, Palanimuthu y Chen (1999), quienes proponen un método basado en búsquedas tabú. Este algoritmo se evalúa exhaustivamente mediante técnicas estadísticas para encontrar sus parámetros óptimos de funcionamiento. Los resultados del algoritmo se comparan contra los resultados de la heurística propuesta por Rajendran (1992), que es la misma utilizada para generar la solución inicial del algoritmo.

Gupta, Neppalli y Werner (2001) proponen nueve heurísticas para el problema de dos máquinas minimizando el tiempo de flujo sujeto a un $C_{\text{máx}}$ óptimo, $Lex(C_{\text{máx}}, F)$. Los autores identifican algunos casos que se pueden resolver de forma polinomial y llevan a cabo un análisis exhaustivo de las heurísticas propuestas. Los métodos basados en inserción muestran los mejores resultados.

Más adelante, Gupta, Hennig y Werner (2002) presentan algunos algoritmos basados en búsquedas locales y tres metaheurísticas para el taller de flujo con dos máquinas. Los métodos desarrollados son: recocido simulado, aceptación de límites y búsqueda tabú. Los criterios a optimizar se componen de varios pares lexicográficos que incluyen $C_{\text{máx}}$, tiempo de flujo ponderado y tardanza ponderada. Los métodos propuestos se comparan con el algoritmo genético propuesto por Neppalli, Chen y Gupta (1996).

T'Kindt et al. (2002) también abordan el problema de taller de flujo con dos máquinas estudiado por Gupta, Neppalli y Werner (2001). En este caso los autores proponen un algoritmo de colonias de hormigas, también conocido como ACO ($Ant\ Colony\ Optimization$). Éste método se compara contra la heurística propuesta por T'Kindt, Gupta y Billaut (2003) y contra otros de un solo objetivo presentes en la literatura. A pesar de ser más lento en algunos casos, el método ACO propuesto demuestra obtener resultados de mejor calidad.
El artículo de T'Kindt, Gupta y Billaut (2003) trabaja con el mismo problema. Aquí los autores proponen un método B&B capaz de resolver instancias de hasta 35 trabajos en un tiempo razonable. También se proponen algunas heurísticas nuevas.

2.2. Objetivos ponderados

Como hemos mencionado antes, la mayoría de los estudios llevados a cabo utilizan la estrategia *a priori*, que consiste en combinar linealmente los objetivos. Al hacer esta combinación se pueden aplicar la mayoría de los algoritmos propuestos para un solo objetivo a problemas de más objetivos.
Nagar, Heragu y Haddock (1995a) proponen un procedimiento B&B para resolver el problema de taller de flujo con dos máquinas con una combinación ponderada de tiempo total de flujo y $C_{\text{máx}}$. El algoritmo inicializa el árbol

branch & bound (B&B) con una solución factible y una cota superior, ambos obtenidos a través de una heurística voraz. El algoritmo es capaz de encontrar soluciones óptimas para problemas con dos máquinas y hasta 500 trabajos, pero sólo bajo algunos supuestos muy restrictivos y distribuciones de tiempos de proceso concretas. Los mismos autores utilizan este algoritmo B&B en Nagar, Heragu y Haddock (1996) como una herramienta para obtener la población inicial en un algoritmo genético. Este enfoque híbrido se prueba para el mismo problema de taller de flujo con dos trabajos y dos objetivos y muestra mejores resultados que los algoritmos B&B o genético en solitario.

Otro algoritmo genético es presentado por Sridhar y Rajendran (1996) con los objetivos de $C_{\text{máx}}$ y tiempo de flujo, incluyendo también un tercer objetivo, el tiempo ocioso de máquina o *idle time*. Cavalieri y Gaiardelli (1998) estudian un problema de producción realista que modelizan como un taller de flujo con $C_{\text{máx}}$ y tardanza total como objetivos. Proponen dos algoritmos genéticos en los que muchos de sus parámetros son adaptativos.

Yeh (1999) propone otro método B&B y lo compara contra el de Nagar, Heragu y Haddock (1995a) obteniendo mejores resultados. Para problemas sin estructura, el algoritmo propuesto por Yeh es capaz de resolver instancias de hasta 14 trabajos en menos tiempo que el algoritmo de Nagar, Heragu y Haddock (1995a). El mismo autor mejora este algoritmo en Yeh (2001) y finalmente propone un algoritmo genético híbrido en Yeh (2002) mejorando los resultados de todo su trabajo previo. Cabe destacar que en todos los artículos nombrados se estudia exclusivamente el caso particular de dos máquinas.

Lee y Chou (1998) proponen algunos métodos heurísticos y un modelo híbrido de programación entera para el problema de m máquinas y los objetivos de $C_{\text{máx}}$ y tiempo total de flujo. Este estudio muestra que el enfoque de la programación entera solamente es válido para instancias muy pequeñas. Los

mismos autores obtienen resultados similares en un trabajo posterior (Chou y Lee, 1999).

Sivrikaya-Şerifoğlu y Ulusoy (1998) presentan tres algoritmos B&B y dos heurísticas para el taller de flujo de dos maquinas con los objetivos de $C_{\text{máx}}$ y tiempo total de flujo. Todos los métodos presentados se comparan entre ellos en una serie de experimentos. Estos son capaces de resolver instancias de a lo sumo 18 trabajos.
En el artículo Chakravarthy y Rajendran (1999) se estudia una combinación de $C_{\text{máx}}$ y tiempo total de flujo, pero en este caso se propone un algoritmo de recocido simulado.

Chang, Hsieh y Lin (2002) estudian el enfoque de ponderación gradualmente priorizada (*gradual-priority weighting approach*) en vez del enfoque más común de ponderación variable. Proponen un algoritmo genético y una búsqueda local genética. Los dos métodos están relacionados a los propuestos en Murata, Ishibuchi y Tanaka (1996) y en Ishibuchi y Murata (1998), respectivamente. Mediante experimentos numéricos el autor muestra que el enfoque ponderación gradualmente priorizada obtiene mejores resultados.
Framiñan, Leisten y Ruiz-Usano (2002) proponen varias heurísticas en conjunto con una extensa evaluación computacional para el problema de m máquinas con los objetivos de $C_{\text{máx}}$ y tiempo total de flujo. Allahverdi (2003) también estudia los mismos objetivos. Llevando a cabo un experimento computacional para estudiar el comportamiento de un total de 10 heurísticas diferentes. Entre los métodos estudiados, tres de las heurísticas propuestas mejoran los resultados de las demás. También se proponen algunas relaciones de dominancia para algunos casos especiales.

Allahverdi (2004) estudia los objetivos $C_{\text{máx}}$ y tardanza máxima. En este trabajo se lleva a cabo un estudio de dos variantes del mismo problema. En

la primera de ellas se estudia una combinación ponderada de los dos objetivos sujeta a una tardanza máxima, mientras que, la segunda propone una combinación ponderada de ambos objetivos. El autor propone una heurística y la compara los resultados de Daniels y Chambers (1990) y Chakravarthy y Rajendran (1999). De acuerdo a los resultados obtenidos por el autor, la heurística propuesta supera a los métodos con los que se compara.

Ponnambalam et al. (2004) proponen un algoritmo genético basándose en algunas ideas del problema del agente viajero (TSP). El algoritmo genético implementado utiliza una combinación ponderada de criterios como valor de *fitness* para cada individuo de la población. El algoritmo no se compara con ningún otro método de la literatura y solo se muestran algunos resultados obtenidos con instancias pequeñas para el taller de flujo.

Lin y Wu (2006) se centran en el caso de dos máquinas con una combinación ponderada de $C_{\max}$ y tiempo de flujo. Los autores presentan un método B&B que prueban con un conjunto de instancias. El método propuesto es capaz de encontrar las soluciones óptimas en instancias de hasta 15 trabajos.

Lemesre, Dhaenens y Talbi (2007) estudian el problema de m máquinas con los criterios de $C_{\max}$ y tardanza total. Se emplea una metodología especial llamada el método de las dos fases. La metodología se basa en una implementación de un algoritmo B&B. Debido a problemas de rendimiento el método se paraleliza. Como resultado se resuelven óptimamente instancias de hasta 20 trabajos y 20 máquinas. Sin embargo el tiempo empleado para la resolución de éstas llega a ser de siete días en un *cluster* de cuatro computadoras paralelas.

Más adelante Ruiz y Allahverdi (2009) presentan un algoritmo genético para el problema del taller de flujo de permutación con los objetivos de $C_{\max}$ y tardanza máxima, sujeto a un límite superior para la tardanza máxima. El algoritmo genético presentado es comparado contra las heurísticas de Allahverdi

(2004) y de Framiñan y Leisten (2006) obteniendo mejores resultados.

2.3. Enfoque de frontera de Pareto

La cantidad de estudios realizados con el enfoque *a posteriori* es significativamente menor que la cantidad de estudios existentes sobre otro tipo de enfoque. En el trabajo anteriormente comentado de Daniels y Chambers (1990), los autores proponen un procedimiento B&B para los objetivos $C_{\max}$ y $T_{\max}$ que es capaz de obtener la frontera global de Pareto para el caso de dos máquinas. Un algoritmo genético propuesto por Murata, Ishibuchi y Tanaka (1996) es capaz de obtener una frontera de Pareto para $C_{\max}$ y tardanza total. Este algoritmo al que llamaremos MOGA (*Multi Objective Genetic Algorithm*) aplica elitismo, mediante la copia de algunos individuos, pertenecientes al conjunto de individuos no dominados, en la población de la siguiente generación. La frontera de Pareto se mantiene en una población auxiliar. El proceso de selección del algoritmo se basa en la elección de un valor de *fitness* para cada solución, obtenida mediante la suma ponderada de los valores de los objetivos de la misma. Los autores también realizan pruebas de su algoritmo con tres objetivos incluyendo el tiempo total de flujo como tercer criterio. Luego en Ishibuchi y Murata (1998) el algoritmo se extiende añadiendo un paso más de búsqueda local que se aplica a cada solución nueva, luego del cruce y la mutación.

Sayın y Karabatı (1999) estudian un algoritmo B&B que genera la frontera de Pareto óptima para un taller de flujo con dos máquinas usando los objetivos de $C_{\max}$ y tiempo de flujo. Los experimentos llevados a cabo solo comparan los resultados obtenidos con heurísticas como la de Johnson (1954) y Rajendran (1992). Algunas instancias de hasta 24 trabajos se resuelven óptimamente. Liao, Yu y Joe (1997) proponen otro algoritmo de B&B para un problema de dos criterios de optimización, con los objetivos de minimizar el $C_{\max}$ y el

número de trabajos retrasados y también con los objetivos de $C_{\text{máx}}$ y tardanza total. Las cotas inferiores se obtienen mediante el algoritmo de Jonhson (1954) para $C_{\text{máx}}$ y la regla de Moore o EDD (*Earliest Due Date*) para el número de trabajos retrasados. Para cada uno de los nodos de la programación parcial, se calculan dos cotas inferiores utilizando las heurísticas nombradas anteriormente. Las soluciones no dominadas son guardadas en un conjunto externo. Al final del algoritmo este conjunto contiene la frontera de Pareto óptima del problema. Lee y Wu (2001) también estudian el caso de dos máquinas mediante métodos B&B, pero utilizando una combinación ponderada de tiempo total de flujo y tardanza total. Los autores no comparan los algoritmos propuestos con los existentes en la literatura y se limitan a mostrar los resultados obtenidos por sus algoritmos.

Un nuevo tipo de algoritmo genético es propuesto por Bagchi (2001). Este algoritmo está basado en el método NSGA (*Non-dominated Sorting Genetic Algorithm*) de Srinivas y Deb (1994) (ver Sección 3.1.1). Algunos experimentos cortos se llevan a cabo a con una sola instancia para taller de flujo usando los objetivos de $C_{\text{máx}}$ y tiempo total de flujo. Murata, Ishibuchi y Gen (2001) mejoran el MOGA propuesto anteriormente en Murata, Ishibuchi y Tanaka (1996). En este nuevo método, llamado CMOGA, se refina el proceso de asignación de las ponderaciones a los objetivos. Se llevan a cabo algunos experimentos con $C_{\text{máx}}$ y tardanza total como objetivos. Estos experimentos muestran que el nuevo algoritmo propuesto CMOGA mejora los resultados del algoritmo original MOGA.

Ishibuchi, Yoshida y Murata (2003) presentan un profundo estudio que muestra el efecto de agregar una búsqueda local en su algoritmo previo (Ishibuchi y Murata, 1998). La búsqueda local se aplica solamente a buenos individuos y especificando las direcciones de la búsqueda. Este tipo de búsqueda local ha mostrado muy buenos resultados en muchos algoritmos genéticos

multi-objetivo. En Loukil, Teghem y Fortemps (2000) varios problemas de programación diferentes se resuelven con diferentes combinaciones de objetivos. La técnica principalmente utilizada es un algoritmo de búsqueda tabú multi-objetivo llamado MOTS (*Multi Objective Tabu Search*). El articulo contiene un estudio que comprende problemas de una sola máquina y también de máquinas paralelas. Luego en Loukil, Teghem y Tuyttens (2005), se lleva a cabo un estudio similar, pero en este caso se emplea otro enfoque multi-objetivo, un algoritmo de recocido simulado.

En el artículo de Toktaş, Azizoğlu y Köksalan (2004) se utiliza un enfoque B&B para el caso de dos máquinas con los objetivos de $C_{\max}$ y adelanto (*earliness*). Esta combinación de objetivos no había sido investigada en la literatura con anterioridad. El procedimiento es capaz de resolver problemas de hasta 25 trabajos. Los autores también proponen un método heurístico en el mismo artículo.
Suresh y Mohanasundaram (2004) proponen un algoritmo de recocido simulado basado en Pareto para los criterios de $C_{\max}$ y tiempo total de flujo. Se llevan a cabo experimentos, y los resultados del algoritmo se comparan contra los resultados de Ishibuchi, Yoshida y Murata (2003) y contra una version anterior de un algoritmo de recocido simulado. Los resultados obtenidos mediante experimentos de hasta 20 trabajos, muestran que el algoritmo funciona mejor que los demás para ciertas métricas específicas.

Arroyo y Armentano (2004) estudian heurísticas para dos y tres objetivos con combinaciones de $C_{\max}$, tiempo total de flujo y tardanza total. Para dos máquinas, los autores comparan las heurísticas propuestas contra dos métodos *branch & bound* (B&B) propuestos en Daniels y Chambers (1990) y en Liao, Yu y Joe (1997). Para el caso de m máquinas, los autores comparan los resultados contra los obtenidos en Framiñan, Leisten y Ruiz-Usano (2002). Los resultados favorecen al método propuesto y también mejoran los resultados

del algoritmo genético propuesto en Murata, Ishibuchi y Tanaka (1996) al ser utilizado como generador de la solución inicial.
Los mismos autores del artículo anterior desarrollan un algoritmo de búsqueda tabú para los objetivos de $C_{\text{máx}}$ y tardanza máximo en Armentano y Arroyo (2004). El nuevo algoritmo incluye algunas características avanzadas, como la diversificación o búsqueda local en varios vecindarios. Para el caso de dos máquinas, el algoritmo propuesto se compara con el algoritmo de Daniels y Chambers (1990) y para más de dos máquinas contra el de Ishibuchi y Murata (1998). El algoritmo propuesto demuestra ser competitivo en los experimentos numéricos. En un artículo posterior, Arroyo y Armentano (2005), se lleva a cabo un estudio similar, pero en ese caso utilizando algoritmos genéticos como herramienta de resolución. Este método no se compara con sus propuestas anteriores para el mismo problema.

El $C_{\text{máx}}$ y tiempo total de flujo se estudian en Varadharajan y Rajendran (2005) mediante algoritmos de recocido simulado multi-objetivo (*Multi Objective Simulated Annealing* o MOSA). Los algoritmos comienzan con soluciones generadas por una heurística que luego se actualizan mediante algunos esquemas de mejora. Las dos versiones del algoritmo de recocido simulado (MOSA y MOSAII) demuestran obtener mejores resultados que el algoritmo genético presentado por Ishibuchi y Murata (1998).

Pasupathy, Rajendran y Suresh (2006) proponen un algoritmo genético (*Pareto-archived genetic algorithm*) con búsqueda local y realizan pruebas para los objetivos de $C_{\text{máx}}$ y tiempo total de flujo. Los autores comparan su algoritmo con los de Ishibuchi y Murata (1998) y Chang, Hsieh y Lin (2002). Aparentemente, el algoritmo genético propuesto obtiene mejores resultados bajo unas pruebas limitadas.
Ese mismo año Melab, Mezmaz y Talbi (2006) proponen un algoritmo genético paralelo basado en *grid computing* que apunta a obtener una frontera de

Pareto precisa para los criterios de $C_{\max}$ y tardanza total. A pesar de que los autores no realizan pruebas comparativas contra otros algoritmos existentes, los resultados parecen prometedores. Sin embargo, el tiempo utilizado por el algoritmo es de 10 días en un grupo de computadoras operando en *grid*.

Más adelante, Rahimi-Vahed y Mirghorbani (2007) proponen un complejo algoritmo híbrido de cúmulo de partículas multi-objetivo. Este método utiliza un algoritmo de búsqueda tabú con elitismo para obtener las soluciones iniciales. Luego se usa una búsqueda local paralela para mejorar la solución representada por cada partícula. Este algoritmo tan complejo se compara con el algoritmo genético de Zitzler, Laumanns y Thiele (2001) llamado SPEAII. El algoritmo de cúmulo de partículas, llamado MOPS por las siglas en inglés de *Multi Objective Particle Swarm*, obtiene mejores resultados que el algoritmo SPEAII según los experimentos computacionales llevados a cabo en el artículo. Sin embargo, utiliza para ello un tiempo de CPU mucho mayor.
Geiger (2007) publica un estudio interesante en el que el espacio de búsqueda de la topología del taller de flujo multi-objetivo es analizado. Utilizando varios algoritmos de búsqueda local, el autor analiza la distribución de varios objetivos y prueba varias combinaciones de objetivos.

Por último, hemos añadido algunos algoritmos propuestos mientras este trabajo de tesis estaba en desarrollo. Algunos de estos algoritmos serán comparados más adelante contra los mejores algoritmos resultantes de la revisión del capítulo actual y la evaluación del Capítulo 3 y contra el algoritmo propuesto en el Capítulo 4. El primero de ellos es desarrollado por Yandra y Tamura (2007) , quienes proponen un algoritmo genético multi-objetivo que presenta la particularidad de trabajar con varias poblaciones heterogéneas. Más adelante, Framiñan y Leisten (2008) proponen un nuevo método llamado MOIG (*Multi Objective Iterated Greedy*) basado en un algoritmo voraz iterativo (*Iterated Greedy* o IG), en este caso el algoritmo propuesto es comparado contra el

recocido simulado MOSAII propuesto por Varadharajan y Rajendran (2005), obteniendo muy buenos resultados. El algoritmo propuesto no aplica ningún método de selección, sino que en cada iteración procesa todas las soluciones de la frontera de Pareto obtenida en la iteración anterior. La comparación se hace utilizando instancias pequeñas. Por esta razón, más adelante realizaremos un experimento añadiendo también instancias de mayor tamaño. De esta manera comprobamos la capacidad del algoritmo MOIG para mantener un buen rendimiento con problemas más grandes.

Cheng, Chiang y Fu (2008) proponen un algoritmo genético con búsqueda local diseñado a partir del algoritmo NSGAII propuesto por Deb (2002). En este caso, los autores presentan un mecanismo de selección que mezcla características del método *Fast Non Dominating Sorting* del NSGAII con la selección por *grid* propuesto en el artículo de Zitzler, Laumanns y Thiele (2001) para el algoritmo SPEAII. Se realizan pruebas para comparar el algoritmo propuesto contra PASA de Suresh y Mohanasundaram (2004), PGA_ALS de Pasupathy, Rajendran y Suresh (2006) y MOSAII de Varadharajan y Rajendran (2005). En las comparaciones se utilizan 25 de las 120 instancias propuestas por Taillard, todas de tamaño pequeño o mediano obteniendo mejores resultados en todos los casos.

Los autores Sha y Hung Lin (2009) proponen un algoritmo de enjambre de partículas (*particle swarm o PSO*). Este tipo de algoritmos había sido propuesto antes en la literatura multi-objetivo, obteniendo resultados pobres. Básicamente uno de los principales problemas de este tipo de algoritmos es que requiere mucho tiempo de CPU. En este artículo los autores proponen un método de diversificación basado en tres reglas de asignación. Mediante estas reglas deciden si una solución se añade al archivo de soluciones no dominadas, se elige como la mejor solución de todas o se elige como la peor solución de todas. Este algoritmo es comparado contra varias heurísticas y obtiene me-

jores resultados. Lamentablemente la comparación es bastante injusta ya que además de compararse contra heurísticas solamente, los tiempos de ejecución del algoritmo son 1000 veces superiores a los de los demás algoritmos.

Dubois-Lacoste, López-Ibáñez y Stützle (2009) realizan un estudio acerca de la utilización de algoritmos híbridos y búsquedas locales. Para ello aplican dos tipos de búsquedas locales: búsqueda local de dos fases y búsqueda local Pareto, con tres combinaciones de dos objetivos. En ambas búsquedas locales, aplican dos algoritmos *Iterated Greedy* (Ruiz y Stützle, 2007) especialmente configurados para resolver cada uno de los objetivos estudiados. Finalmente proponen un método de búsqueda local estocástica para un sólo objetivo basada en el algoritmo IG propuesto.

Ese mismo año, Chiang, Cheng y Fu (2009) proponen un algoritmo memético basándose en un algoritmo genético con búsqueda local. Los autores realizan un estudio de la parametrización del algoritmo genético y proponen una búsqueda local basada en el algoritmo NEH. En esta búsqueda local se extraen algunos elementos de una solución elegida y luego se re-insertan a través de un método similar a la heurística NEH conformando nuevas soluciones. En cada generación el archivo de soluciones no dominadas es actualizado luego de la búsqueda local. El algoritmo propuesto es comparado contra los resultados originales obtenidos para los algoritmos MOSAII de Varadharajan y Rajendran (2005), PGA_ALS de Pasupathy, Rajendran y Suresh (2006) y DPSO, un algoritmo de enjambre de partículas propuesto por Wenzhong et al. (2007). La comparativa propuesta utiliza 40 de las 120 instancias de Tailard de tamaño pequeño y mediano. En estos experimentos el algoritmo propuesto obtiene mejores resultados.

Un algoritmo memético compuesto por un algoritmo genético híbrido con búsqueda local es estudiado por Chiang y Fu (2010). Los autores utilizan

el mismo método de selección propuesto por Deb (2002) para el algoritmo NSGAII. La búsqueda local se aplica cada cierto tiempo T_{LS}, este valor y otros parámetros han sido determinados mediante una serie de pruebas. Esta búsqueda local se aplica a todos las soluciones que componen el archivo de soluciones no dominadas en el momento de su ejecución. En ella se calcula un valor peso o *fitness* para cada solución del archivo, mediante la combinación lineal ponderada de los valores de los objetivos. En cada paso de la búsqueda local se cambia la ponderación de cada objetivo para dirigir la búsqueda. El algoritmo propuesto es comparado contra los resultados de Chiang, Cheng y Fu (2009) y Rajendran y Ziegler (2009). Para estas comparaciones los autores utilizan 40 de las instancias de Taillard de tamaño pequeño y mediano, obteniendo buenos resultados en todos los casos.

En 2011 aparece el primer artículo que propone la aplicación de un algoritmo de búsqueda voraz adaptativa aleatoria o GRASP (*Greedy Randomized Adaptative Search*) al problema del taller de flujo. Arroyo y Souza Pereira (2011) adaptan un algoritmo propuesto por ellos mismos para el problema de árbol de recubrimiento multi-objetivo (*multi-objective spanning tree*) al problema de taller de flujo multi-objetivo. El algoritmo propuesto se basa en generar, en primera instancia, un vector de ponderaciones para los objetivos, cada elemento de este vector se rellenará con ponderaciones generadas al azar. Luego para cada elemento del vector se ejecuta un método constructivo basado en la heurística NEH y finalmente una búsqueda local. Los autores realizan varias pruebas para parametrizar tanto la generación del vector de ponderaciones como el método constructivo y la búsqueda local. El algoritmo resultante es probado utilizando un conjunto de 360 instancias utilizado por los mismos autores en Arroyo y Armentano (2005). También contrastan los resultados de su algoritmo contra el algoritmo genético propuesto en ese artículo. El algoritmo GRASP propuesto obtiene resultados un poco mejores que el algoritmo genético en la mayoría de las pruebas, utilizando un tiempo

muy inferior.

Finalmente, Dubois-Lacoste, López-Ibáñez y Stützle (2011) proponen otro algoritmo híbrido de búsqueda local en dos fases para resolver problemas de dos objetivos. En este caso realizan un análisis profundo acerca de cada componente del algoritmo propuesto. El algoritmo híbrido consiste en aplicar dos tipos de búsquedas locales. La primera de ellas es una búsqueda local para un solo objetivo que se aplica a cada objetivo del problema. De esta fase se obtendrán dos soluciones que se añaden a un archivo. Luego se genera un conjunto de valores de ponderación para cada objetivo. Con este vector de ponderaciones se ejecuta iterativamente una búsqueda local cuyo criterio de aceptación es la dominancia. Para los dos tipos de búsqueda local propuesta, los autores utilizan algoritmos basados en el algoritmo IG. En el primer caso el algoritmo IG está orientado a resolver problemas de un objetivo y los autores modifican el criterio de aceptación de la solución de acuerdo al objetivo a resolver. En el segundo tipo de búsqueda local el criterio de aceptación del algoritmo IG depende de la combinación lineal de los objetivos. Finalmente luego de aplicar cada IG se añade al archivo de soluciones las soluciones no dominadas que se hayan encontrado. A la hora de comparar el algoritmo propuesto los autores realizan una primera batería de pruebas mostrando amplias mejoras. Luego realizan la misma batería de pruebas utilizando su propia implementación de los algoritmos de recocido simulado de Varadharajan y Rajendran (2005) y el algoritmo genético de Arroyo y Armentano (2005). En esta último caso también logran obtener mejores resultados.

2.4. Programación por metas y otros enfoques

Existen otros ejemplos de enfoques multi-objetivo, como la programación por metas. Por ejemplo, Selen y Hott (1986) proponen una formulación de programación por metas para un taller de flujo de permutación con dos objeti-

vos. Como ocurre con cualquier método de programación por metas, un valor límite para cada objetivo debe ser determinado de antemano. Más adelante, Wilson (1989) propone un modelo diferente que tiene menos variables pero un mayor número de restricciones. Sin embargo, ambos modelos tienen el mismo número de variables binarias. La comparación entre ambos modelos muestra que el presentado por Selen y Hott (1986) da mejores resultados para $n \geq 15$.

Muchos algoritmos propuestos en la literatura no consideran explícitamente varios objetivos como en las secciones anteriores. Por ejemplo, Ho y Chang (1991) proponen una heurística que está especialmente pensada para minimizar el tiempo de parada de máquinas en un taller de flujo con m máquinas. A pesar de que la heurística no permite configuración de ponderaciones, o valores límites y tampoco maneja una aproximación de Pareto, los autores la prueban con diferentes objetivos. Un enfoque similar es seguido en Gangadharan y Rajendran (1994) donde se propone un algoritmo de recocido simulado para el problema de m máquinas con los criterios de $C_{\text{máx}}$ y tiempo total de flujo. Junto con el método de recocido simulado, se proponen dos heurísticas. Rajendran (1995) propone una heurística para el mismo problema con que se trabaja en Ho y Chang (1991). Después de una amplia gama de experimentos numéricos, las heurísticas propuestas se muestran superiores a las del artículo de Ho y Chang. Un estudio muy similar es presentado por el mismo autor en Rajendran (1994).

Ravindran et al. (2005) presenta tres heurísticas dirigidas a minimizar el $C_{\text{máx}}$ y el tiempo total de flujo. Los autores contrastan los tres métodos propuestos contra la heurística de Rajendran (1995) pero utilizando para ello solo instancias muy pequeñas, de 20 trabajos y 20 máquinas a lo sumo. Las tres heurísticas mejoran los resultados del algoritmo de Rajendran pero no por mucho. Dibujar una línea divisoria en este tipo de trabajos en particular es una tarea ardua, y a veces imposible, ya que muchos autores prueban una misma

heurística con objetivos diferentes. Sin embargo, las comentadas en el párrafo anterior fueron diseñadas teniendo en mente varios objetivos, razón por la cual los hemos incluido en esta revisión bibliográfica.

La Tabla 2.4, contiene, en orden cronológico los artículos revisados, con el número de máquinas (2 o m), el tipo de enfoque multi-objetivo utilizado, los criterios estudiados como así también el tipo de método propuesto en cada caso.

Año	Autor/es	m	Enfoque y Objetivos	Comentarios
1986	Selen y Hott	m	$GP(C_{\text{máx}}, F)$	formulación mixta de programación por metas y entera.
1989	Wilson	m	$GP(C_{\text{máx}}, F)$	formulación mixta de programación por metas y entera.
1990	Daniels y Chambers	m	$\epsilon(C_{\text{máx}}/T_{\text{máx}})$	Heurísticas
	Daniels y Chambers	2	$\#(C_{\text{máx}}, T_{\text{máx}})$	B&B
1991	Ho y Chang	m	$(C_{\text{máx}}, F)$	Heurísticas
1992	Rajendran	2	$Lex(C_{\text{máx}}, F)$	B&B, heurísticas
1994	Gangadharan y Rajendran	m	$(C_{\text{máx}}, F)$	Recocido simulado y Heurísticas
	Rajendran	m	$(C_{\text{máx}}, F)$	Heurísticas
1995a	Nagar, Heragu y Haddock	2	$F_l(C_{\text{máx}}, F)$	B&B
	Rajendran	m	$(C_{\text{máx}}, F)$	Heurísticas
1996	Murata, Ishibuchi y Tanaka	m	$\#(C_{\text{máx}}, T)$, $\#(C_{\text{máx}}, T, F)$	Algoritmos genéticos
	Nagar, Heragu y Haddock	2	$F_l(C_{\text{máx}}, F)$	Algoritmos genéticos
	Neppalli, Chen y Gupta	2	$Lex(C_{\text{máx}}, F)$	Algoritmos genéticos
			$\#(C_{\text{máx}}, T, F)$	Algoritmos genéticos
	Sridhar y Rajendran	m	$F_l(C_{\text{máx}}, F)$	Algoritmos genéticos
1997	Liao, Yu y Joe	2	$\#(C_{\text{máx}}, N_T)$, $\#(C_{\text{máx}}, T)$	B&B
1998	Cavalieri y Gaiardelli	m	$F_l(C_{\text{máx}}, T)$	Algoritmos genéticos
	Ishibuchi y Murata	m	$\#(C_{\text{máx}}, T)$, $\#(C_{\text{máx}}, T, F)$	Algoritmos genéticos
	Lee y Chou	2	$F_l(C_{\text{máx}}, F)$	B&B
	Sivrikaya-Şerifoğlu y Ulusoy	2	$F_l(C_{\text{máx}}, F)$	B&B, heurísticas
1999	Chakravarthy y Rajendran	m	$F_l(C_{\text{máx}}, F)$	Heurísticas, programación entera
	Chou y Lee	2	$F_l(C_{\text{máx}}, F)$	B&B
	Gupta, Palanimuthu y Chen	2	$Lex(C_{\text{máx}}, F)$	Búsquedas Tabú
	Sayın y Karabatı	2	$\#(C_{\text{máx}}, F)$	B&B
	Yeh	2	$F_l(C_{\text{máx}}, F)$	B&B
2000	Loukil, Teghem y Fortemps	m	$\#(many)$	Búsquedas Tabú. Varios objetivos estudiados
2001	Bagchi	m	$\#(C_{\text{máx}}, F)$	Algoritmos genéticos
	Gupta, Neppalli y Werner	2	$Lex(C_{\text{máx}}, F)$	Heurísticas

Continúa en la siguiente página...

Año	Autor/es	m	Enfoque y Objetivos	Comentarios
	Lee y Wu	2	$\#(F,T)$	B&B
	Murata, Ishibuchi y Gen	m	$\#(C_{\text{máx}},T)$	Algoritmos genéticos
	Yeh	2	$F_l(C_{\text{máx}},F)$	B&B
2002	Chang, Hsieh y Lin	m	$F_l(C_{\text{máx}},T)$, $F_l(C_{\text{máx}},T,F)$	Algoritmos genéticos
	Framiñan, Leisten y Ruiz-Usano	m	$F_l(C_{\text{máx}},F)$	Heurísticas
	Gupta, Hennig y Werner	2	$Lex(F,C_{\text{máx}})$, $Lex(T,C_{\text{máx}})$,	Varios métodos. Funciones ponderadas
	T'Kindt et al.	2	$Lex(C_{\text{máx}},F)$	Optimización por colonias de hormigas
	Yeh	2	$F_l(C_{\text{máx}},F)$	Algoritmos genéticos híbridos
2003	Allahverdi	m	$F_l(C_{\text{máx}},F)$	Heurísticas
	Ishibuchi, Yoshida y Murata	m	$\#(C_{\text{máx}},T)$, $\#(C_{\text{máx}},T,F)$	Algoritmos genéticos y búsqueda local
	T'Kindt, Gupta y Billaut	2	$Lex(C_{\text{máx}},F)$	B&B, heurísticas
2004	Allahverdi	m	$\epsilon(F_l(C_{\text{máx}},T_{\text{máx}}),T_{\text{máx}})$ $F_l(C_{\text{máx}},T_{\text{máx}})$	Heurísticas
	Armentano y Arroyo	m	$\#(C_{\text{máx}},T_{\text{máx}})$	Búsqueda tabú
	Arroyo y Armentano	m	$\#(C_{\text{máx}},T_{\text{máx}},F)$	Heurísticas, combinaciones de tres objetivos
	Ponnambalam et al.	m	$F_l(C_{\text{máx}},F)$	Algoritmos genéticos
	Suresh y Mohanasundaram	m	$\#(C_{\text{máx}},F)$	Recocido simulado
	Toktaş, Azizoğlu y Köksalan	2	$\#(C_{\text{máx}},E_{\text{máx}})$	B&B, heurísticas
2005	Arroyo y Armentano	m	$\#(C_{\text{máx}},T_{\text{máx}})$, $\#(C_{\text{máx}},T)$	Algoritmos genéticos
	Loukil, Teghem y Tuyttens	m	$\#(many)$	Recocido simulado. Varios objetivos estudiados
	Ravindran et al.	m	$(C_{\text{máx}},T)$	Heurísticas
	Varadharajan y Rajendran	m	$\#(C_{\text{máx}},F)$	Recocido simulado
2006	Framiñan y Leisten	m	$\epsilon(C_{\text{máx}}/T_{\text{máx}})$	Heurísticas
	Lin y Wu	2	$F_l(C_{\text{máx}},F)$	B&B
	Melab, Mezmaz y Talbi	m	$\#(C_{\text{máx}},T)$	Algoritmos genéticos paralelos
	Pasupathy, Rajendran y Suresh	m	$\#(C_{\text{máx}},F)$	Algoritmos genéticos
2007	Geiger	m	$\#(many)$	Búsqueda local

Continúa en la siguiente página...

Año	Autor/es	m	Enfoque y Objetivos	Comentarios
	Lemesre, Dhaenens y Talbi	m	$F_l(C_{\max}, T)$	B&B. Parallelism
	Rahimi-Vahed y Mirghorbani	m	$\#(F, T)$	Enjambre de partículas híbrido
	Yandra y Tamura	m	$\#(F, T)$	Enjambre de partículas híbrido
2008	Framiñan y Leisten	m	$F_l(C_{\max}, T_{\max})$	Algoritmo Voraz Iterado
	Cheng, Chiang y Fu	m	$\#(C_{\max}, F)$	Algoritmo genético con búsqueda local
2009	Sha y Hung Lin	m	$\#(C_{\max}, F, I)$	Enjambre de partículas
	Dubois-Lacoste, López-Ibáñez y Stützle	m	$\#(C_{\max}, F, T^w)$	Búsqueda local híbrida estocástica
	Chiang, Cheng y Fu	m	$\#(C_{\max}, F)$	Algoritmo memético con búsqueda local voraz
2010	Chiang y Fu	m	$\#(C_{\max}, F)$	Algoritmo memético
2011	Arroyo y Souza Pereira	m	$\#(C_{\max}, T)$ $\#(C_{\max}, T,)$	Algoritmo de búsqueda voraz adaptativa aleatoria
2011	Dubois-Lacoste, López-Ibáñez y Stützle	m	$\#(C\text{máx}, F)$ $\#(C\text{máx}, T)$ $\#(C\text{máx}, T^w)$ $\#(F, T)$ $\#(F, T^w)$	Búsqueda local híbrida estocástica

Tabla 2.1: Artículos revisados referidos al taller de flujo multi-objetivo.

2.5. Conclusiones del capítulo

Hemos revisado un total de 64 artículos. Entre todos ellos, 24 tratan con el caso específico de dos máquinas. De los 40 restantes que estudian el caso más general de m máquinas, un total de 21 utilizan el enfoque *a posteriori* o enfoque basado en frontera de Pareto.

Los resultados de estos métodos no siempre son comparables por varias razones. Primero, comúnmente los autores no tratan la misma combinación de criterios. Segundo, las comparaciones se llevan a cabo generalmente utilizando distintos bancos de pruebas, compuestos en muchos casos por conjuntos de instancias completamente diferentes. En algunos casos, incluso, los bancos de pruebas responden a tipos de problemas completamente diferentes al taller de flujo de permutación. Además de ello, existe el problema del hardware y el sistema operativo utilizados, que complica la reutilización de resultados de bancos de prueba más antiguos, ya que no es justo comparar resultados obtenidos con computadoras que realizan el procesamiento a menor velocidad. Por último y más importante, las medidas de calidad empleadas no son las apropiadas, como se ha demostrado en algunos estudios.

Como resultado de este gran esfuerzo de revisión bibliográfica, hemos publicado en 2008 el artículo Minella, Ruiz y Ciavotta (2008a), en el cual además de hacer una revisión exhaustiva de los artículos nombrados proponemos un nuevo banco de pruebas, basado en las conocidas instancias de Taillard. Para llevar a cabo este artículo hemos re-implementado 23 algoritmos y hemos ejecutado el banco de pruebas propuesto contra todos ellos. El siguiente capítulo trata acerca de la manera de medir la calidad de los resultados en algoritmos multiobjetivo y también muestra el desarrollo del banco de pruebas y los resultados obtenidos en el artículo antes nombrado.

CAPÍTULO 3

REVISIÓN Y COMPARATIVA DE ALGORITMOS MULTI-OBJETIVO

En este trabajo no solo hemos implementado algoritmos específicamente propuestos para el taller de flujo de permutación, sino también otros algoritmos generales de optimización multi-objetivo. En el caso de los algoritmos que no fueron propuestos para el PFSP es necesario realizar algunas adaptaciones. En las siguientes secciones profundizaremos sobre los algoritmos que han sido considerados para la implantación y su posterior comparación.

3.1. Enfoque de Pareto para el problema del taller de flujo

Detallamos ahora los algoritmos que han sido re-implementados y probados entre aquellos propuestos específicamente para el problema de taller de flujo. Estos métodos han sido revisados en el Capítulo 2 y en esta sección nos extenderemos sobre algunos detalles particulares acerca de su implementación.

El algoritmo MOGA de Murata, Ishibuchi y Tanaka (1996) fue diseñado específicamente pensando el problema del taller de flujo multi-objetivo. Es un algoritmo genético simple con un operador de selección modificado. Durante la selección, se generan ponderaciones para los objetivos. De esta manera el algoritmo pretende distribuir la búsqueda hacia diferentes direcciones. Los autores también incorporan un mecanismo de elitismo, que consiste en copiar varias soluciones de la aproximación actual a la frontera de Pareto en la siguiente generación. Nos referiremos a nuestra implementación del algoritmo MOGA como MOGA_Murata.

Chakravarthy y Rajendran (1999) presentan un algoritmo de recocido simulado simple que intenta minimizar la suma ponderada de dos objetivos. La solución inicial se obtiene a partir de la mejor solución generada entre las heurísticas EDD (*Earliest Due Date*), LSS (*Least Static Slack*) y NEH de Nawaz, Enscore y Ham (1983). Para generar el vecindario de la solución actual se utiliza el método AIS (*Adjacent Interchange Scheme*). Este algoritmo, al que llamaremos SA_Chakravarty, no está pensado originalmente como un enfoque de Pareto debido a que los objetivos están ponderados. Si embargo ha sido incluido en la comparación para tener una idea del rendimiento que este tipo de métodos puede tener en la práctica. Para "simular" un enfoque de Pareto se ejecuta el algoritmo SA_Chakravarty 100 veces, utilizando cada vez una combinación diferente de las ponderaciones de los objetivos. Cada una de las 100 soluciones es analizada y las soluciones no dominadas son tomadas como el resultado del algoritmo.

Bagchi (2001) propone una modificación del bien conocido algoritmo NSGA (ver Sección 3.1.1 de este capítulo) y lo adapta al problema del taller de flujo. Este algoritmo, llamado ENGA, se diferencia del NSGA en que incorpora elitismo. Las poblaciones *parent* y *offspring* se combinan en un único conjunto, luego se aplica la ordenación por dominancia a este conjunto y el

50 % de las soluciones no dominadas se copian en la población *parent* de la generación siguiente.

Murata, Ishibuchi y Gen (2001) mejoran el algoritmo MOGA original de Murata, Ishibuchi y Tanaka (1996). Para ello cambian la forma en que las ponderaciones de los objetivos son distribuidas. El nuevo método propuesto hace uso de una estructura de celdas para seleccionar las ponderaciones de los objetivos en cada iteración. Nos referiremos a este algoritmo como CMOGA.

Suresh y Mohanasundaram (2004) proponen un algoritmo de recocido simulado de Pareto llamado PASA (*Pareto Archived Simulated Annealing*). Un nuevo mecanismo de perturbación llamado *segment random insertion* o SRI (Inserción aleatoria de segmento) se utiliza para generar el vecindario de la solución procesada. Se usa un repositorio o archivo externo de soluciones no dominadas. La solución inicial se genera aleatoriamente. El método SRI se usa para obtener un vecindario del conjunto de soluciones candidatas, que se utilizan a su vez para actualizar el repositorio de soluciones no dominadas. Se utiliza una función de *fitness*, que es una suma ponderada y escalada de las funciones objetivo, para elegir una nueva solución actual. También se implementa un método de reinicio y recocido. Nos referiremos a este método como MOSA_Suresh.

Armentano y Arroyo (2004) desarrollan un método de búsqueda tabú multi-objetivo llamado MOTS. El algoritmo trabaja con varios caminos paralelos de soluciones, cada uno con su propia lista tabú. Se genera un conjunto de soluciones iniciales mediante una heurística. Se aplica una búsqueda local al conjunto de soluciones para generar varias soluciones nuevas. Se utiliza un sistema de agrupación (*clustering*) para asegurar que el tamaño del conjunto actual de soluciones se mantiene constante. El algoritmo hace uso de un repositorio externo para guardar todas las soluciones no dominadas encontradas

durante su ejecución. Luego de algunos experimentos iniciales, encontramos que este método, bajo el criterio de parada considerado (que se detallará luego), llevaba a cabo menos de 12 iteraciones. Esto, en conjunto con el hecho de que el procedimiento de diversificación no está suficientemente claro en el texto original, nos ha llevado a no incluir este procedimiento en nuestra re-implementación. El procedimiento de inicialización del MOTS toma la mayor parte del tiempo de CPU para valores grandes de n. El gran tamaño de vecindario utilizado, hace que el método se vuelva extremadamente lento para tamaños de n grandes.

Arroyo y Armentano (2005) proponen un algoritmo de búsqueda local genética con las siguientes características: preservación de diversidad de la población, elitismo (un subconjunto de la frontera de Pareto actual se copia a la siguiente generación) y el uso de una búsqueda local multi-objetivo. El concepto de dominancia se utiliza para asignar el valor de *fitness* (mediante los algoritmos de ordenación no dominada y de medición de la densidad de población o *crowding* propuestos para el método NSGAII) a las soluciones en el procedimiento de búsqueda local. Nos referiremos a este método como MOGALS_Arroyo.

En el artículo Varadharajan y Rajendran (2005) se presenta un recocido simulado multi-objetivo (MOSA). El algoritmo comienza con una inicialización, que consiste en generar dos soluciones iniciales para cada objetivo utilizando dos heurísticas simples y veloces. Estas secuencias se mejoran entonces utilizando tres métodos de mejora y finalmente se usan en forma alternativa como la secuencia actual en el algoritmo SA (simulated annealing). El MOSA intenta obtener soluciones no dominadas implementando una función de probabilidad simple, que intenta generar soluciones en la frontera de Pareto. La función de probabilidad es variada durante la ejecución, para cubrir la mayor parte posible del espacio de los objetivos. Llamaremos a este algoritmo MOSAII.

Pasupathy, Rajendran y Suresh (2006) proponen un algoritmo genético al que denominaremos PGA_ALS. El algoritmo utiliza un procedimiento de inicialización para producir buenas soluciones que se añaden a una población generada aleatoriamente. El PGA_ALS usa dos poblaciones, la población de trabajo y una población externa. La población de trabajo evoluciona mediante un procedimiento de categorización en Pareto similar al utilizado por el NSGAII. También se utiliza un procedimiento de *crowding* (medición de la densidad de la población en un sector dado) como método de selección secundario. Las soluciones no dominadas se guardan en la población externa o repositorio. A la mitad de estas soluciones se les aplican dos búsquedas locales diferentes para mejorar la calidad de la frontera de Pareto obtenida.

Finalmente, también hemos re-implementado el método llamado PILS de Geiger (2007). Este algoritmo se basa en una búsqueda local iterativa, la cual, se basa en dos principios fundamentales, primero la *intensificación* usando una búsqueda local de vecindario variable, y segundo un procedimiento de perturbación. La relación de dominancia se utiliza para guardar las soluciones no dominadas. Este esquema se repite de manera iterativa para alcanzar las regiones más favorables del espacio de búsqueda.

Entre los 16 artículos que específicamente tratan el problema del taller de flujo de permutación multi-objetivo revisados en la Sección 2, hemos re-implementado un total de 10. Hemos elegido no re-implementar los algoritmos genéticos de Ishibuchi y Murata (1998) y de Ishibuchi, Yoshida y Murata (2003) porque está demostrado que sus resultados son inferiores a los del algoritmo de búsqueda tabú de Armentano y Arroyo (2004), entre otros. Loukil, Teghem y Fortemps (2000) y Loukil, Teghem y Tuyttens (2005) presentan método generales aplicados a muchos problemas de planificación. Esta generalidad y la falta de detalles explicativos en los artículos originales nos han

disuadido de intentar su re-implementación. Arroyo y Armentano (2004) simplemente propone varias heurísticas y finalmente, el algoritmo de optimización por enjambre de partículas (particle swarm) presentado en Rahimi-Vahed y Mirghorbani (2007) es increíblemente complejo, y además hace uso de técnicas de computación paralela por lo que hemos elegido no implementarlo.

Esta parte de la tesis ha sido desarrollada durante los años 2006-2008 por lo que los algoritmos más nuevos surgidos luego de 2008 no se han incluido en la comparativa. Sin embargo la revisión de literatura del Capítulo 2 si que ha sido actualizada.

3.1.1. Otros algoritmos generales multi-objetivo Pareto

En la literatura multi-objetivo pueden encontrarse muchas propuestas interesantes, principalmente en forma de algoritmos evolutivos, que no han sido aplicados al PFSP aún. En esta sección revisaremos algunos de ellos, que han sido re-implementados y adaptados al PFSP por primera vez en esta tesis.
Srinivas y Deb (1994) proponen el bien conocido algoritmo genético de ordenación no dominada, llamado generalmente NSGA (*Non-dominated Sorting Genetic Algorithm*). Este método difiere de un algoritmo genético normal por la manera en que se lleva a cabo el proceso de selección. El procedimiento de ordenación no dominada (*NDS*) divide iterativamente la población en varias fronteras. La Figura 3.1 muestra el diagrama de flujo de este procedimiento. Luego, a cada individuo se le asigna un valor de ponderación mediante un procedimiento que mide la distancia entre elementos de una misma frontera llamado *crowding distance operator*. Además, este valor se modifica de acuerdo a un factor que se calcula de acuerdo al número de individuos cercanos en una porción del espacio de soluciones. Se utiliza un parámetro de comparación σ_{share} para calcular el valor de *crowding* de cada individuo (cuantos individuos cercanos tiene). Las demás características son comunes a cualquier algoritmo genético.

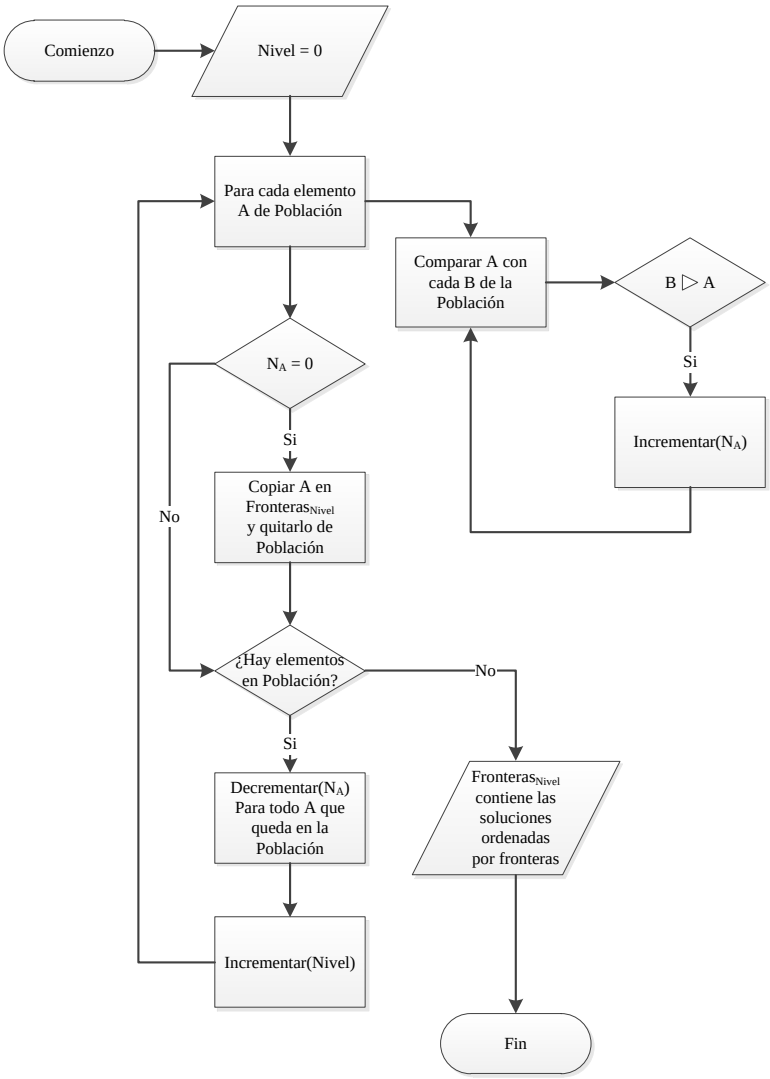


Figura 3.1: Procedimiento *Non Dominated Sorting* utilizado por el algoritmo NSGA para la ordenación de los elementos de una solución en fronteras.

Zitzler y Thiele (1999) presentan otro algoritmo genético llamado SPEA. La característica más importante de este método es que todas las soluciones no dominadas se guardan en una población externa llamada repositorio. La ponderación recibida por cada individuo depende del número de soluciones de la población externa que domine. El algoritmo también incluye un procedimiento de agrupación para reducir el tamaño del conjunto de individuos no dominados sin perder información importante. Finalmente, la diversidad de la población se mantiene utilizando la relación de dominancia. Más adelante, Zitzler, Laumanns y Thiele (2001) proponen una mejora a este algoritmo y le llaman SPEAII. Este nuevo algoritmo incorpora una estrategia diferente y más sutil para evitar algunos de los problemas del SPEA. También incluye una técnica de estimación de la densidad poblacional, que es una adaptación del método del k-*ésimo* vecino más cercano, y un nuevo procedimiento de disminución de tamaño del repositorio.

Knowles y Corne (2000) presentan otro algoritmo llamado PAES. Este método utiliza una búsqueda local y una población auxiliar o repositorio. El algoritmo tiene tres partes distinguibles, la primera es la generación de la solución candidato, que se hace mediante una mutación aleatoria sobre una misma solución. La segunda parte es la aceptación de la solución candidato. Esta parte se encarga de aceptar o descartar una solución nueva. La tercera y última parte es la generación del repositorio de soluciones no dominadas. Según los autores, este algoritmo representa la forma no trivial más simple de búsqueda local. En el mismo artículo los autores presentan una mejora del algoritmo PAES, al que denominan $(\mu + \lambda)-$PAES. En este caso se mantiene una población de soluciones candidatas μ. Una de ellas es elegida utilizando torneo binario y se crean λ soluciones mediante un procedimiento de mutación. Luego, se crea una población de tamaño $\mu + \lambda$ y se calcula para cada individuo un valor de dominancia. μ individuos se seleccionan para actualizar la población candidata mientras que las soluciones no dominadas encontradas se mantienen en un

repositorio externo.

Corne, Knowles y Oates (2000) proponen otro algoritmo genético. Este método, llamado PESA, utiliza una población externa EP y otra interna IP con la finalidad de encontrar una frontera de Pareto bien distribuida. Se implementa un procedimiento de selección y reemplazo basado en el grado de densidad poblacional. La evolución de IP se lleva a cabo mediante un esquema genético simple, mientras que EP contiene la soluciones no dominadas encontradas. El tamaño de EP tiene un límite superior. Los individuos de las zonas más pobladas se eliminan mediante un procedimiento de hiper-rejilla (*hypergrid*). Más adelante, en Corne et al. (2001) se presenta un método llamado PESAII, que es una mejora del método PESA original. Este nuevo algoritmo difiere del anterior sólo en la técnica de selección, en la que el valor de ponderación se asigna de acuerdo con una técnica de cálculo de hiper-caja (*hyperbox*) en el espacio de los objetivos. Esta técnica, en vez de asignarle una ponderación selectiva a un individuo, se la asigna a la hiper-caja del espacio del objetivo que esté ocupada por al menos un individuo. Durante el proceso de selección, se elige un individuo aleatoriamente de la hiper-caja que tenga un valor de ponderación mayor.

En Deb (2002) se presenta una evolución del algoritmo NSGA, llamado NSGAII. Este algoritmo utiliza una nueva técnica llamada *Fast Non Dominated Sorting* o FNDS. A diferencia del NSGA aquí se asigna un valor de categoría o rango a cada individuo de la población, por lo que no hay necesidad de un parámetro de compartición (σ_{share}) para calcular su ponderación. Al mismo tiempo se calcula un valor de densidad poblacional para cada elemento de la población. El operador de selección utiliza un valor de categorización y el valor de densidad poblacional para elegir los mejores individuos para los procedimientos de cruce y mutación. Se implementa también un eficiente sistema de elitismo al comparar dos generaciones sucesivas y conservar los mejores

individuos. Este método, el NSGAII, se usa ampliamente en la literatura multi-objetivo para resolver una enorme variedad de problemas. Más tarde, Deb (2003) introduce otro algoritmo genético llamado CNSGAII. Básicamente, en este algoritmo el procedimiento de *Crowding* es reemplazado por un enfoque de *Clustering*. El funcionamiento es el siguiente, una vez que se ha procesado una generación, la generación previa tiene un tamaño de P_{size} (la población *padre*) y la actual (población *hija*) es del mismo tamaño. Al combinar ambas poblaciones se obtiene una población del tamaño $2P_{size}$, pero solo la mitad de estos individuos serán necesarios en la siguiente generación. Para elegir la soluciones que serán parte de la siguiente generación se utiliza un método de ordenación dominada y luego se aplica un procedimiento de *Clustering*.

Deb (2003) estudia otro tipo de algoritmo genético. Este método, llamado $\varepsilon-$MOEA, utiliza dos poblaciones co-evolutivas, una normal llamada P y un repositorio o archivo llamada A. En cada paso, se eligen dos soluciones una de cada población P y A. La solución hija generada se compara con cada elemento de P. Si la solución hija domina al menos a un individuo en P lo reemplazará. La solución es descartada si es dominada por P. La solución hija también se comprueba contra las soluciones de A. En la población repositorio A, la $\varepsilon-$dominancia se usa de la misma manera. Por ejemplo, utilizando la notación previa, una solución x_1 $\epsilon-$domina fuertemente a otra solución x_2 ($x_1 \prec_\varepsilon x_2$) si $f_j(x_1) - \varepsilon \lhd f_j(x_2)$.

Zitzler y Künzli (2004) proponen otro método llamado $B-$IBEA. La principal idea de este procedimiento es definir el objetivo de optimización en términos de una métrica de calidad y directamente usarla en el proceso de selección. El algoritmo B-IBEA realiza torneos binarios para la selección e implementa una selección de entorno al quitar iterativamente los peores individuos de la población y actualizar el valor de ponderación de los individuos restantes. Para estas comparaciones utiliza un indicador I_ε. En el mismo traba-

jo también se presenta una variación adaptativa llamada A-IBEA. Se propone un procedimiento de escalado adaptativo con la finalidad de permitir que el comportamiento del algoritmo sea independiente del parámetro k, usado en el algoritmo básico B-IBEA.

Finalmente, Kollat y Reed (2005) proponen también una variación del NSGAII, denominada $\varepsilon-$NSGAII, que aplica la $\varepsilon-$dominancia para generar el repositorio exterior de soluciones y un procedimiento que permite cambiar el tamaño de la población de manera adaptativa. El parámetro ε establece el tamaño de la rejilla o *grid* en el espacio de los objetivos. Dentro de cada celda de la grilla no se permite más de una solución. Además, el algoritmo funciona alternando dos fases. Comienza utilizando una población muy pequeña de 10 individuos y se ejecutan varias iteraciones del algoritmo NSGAII. Durante esas iteraciones todas los soluciones no dominadas se copian a un conjunto externo de soluciones. Cuando no se pueden hallar más mejoras en la frontera de Pareto actual comienza la segunda parte. En esta segunda parte, se aplica el procedimiento de $\varepsilon-$dominancia al repositorio externo.

Hemos visto, al revisar todos estos artículos, la gran variedad de técnicas multi-objetivo propuestas para resolver problemas generales. Muchas de éstas técnicas están basadas principalmente en algoritmos genéticos o algoritmos evolutivos más generales. Paradójicamente, casi ninguna de estas técnicas se ha aplicado al PFSP. En este trabajo hemos realizado este esfuerzo. Los 23 algoritmos re-implementados, tanto si son específicos para el PFSP, como si son algoritmos genéricos están resumidos en la tabla 3.1.

Acrónimo	Año	Autor/es	Tipo algoritmo
NSGA	1994	Srinivas y Deb	Algoritmo genético. General
MOGA_Murata	1996	Murata, Ishibuchi y Tanaka	Algoritmo genético. Específico

Continúa en la siguiente página...

Acrónimo	Año	Autor/es	Tipo algoritmo
SPEA	1999	Zitzler y Thiele	Algoritmo genético. General
SA_Chakravarty	1999	Chakravarthy y Rajendran	Recocido simulado. Específico
PAES	2000	Knowles y Corne	Population local search. General
$(\mu + \lambda)-$PAES	2000	Knowles y Corne	Population local search. General
PESA	2000	Corne, Knowles y Oates	Algoritmo genético. General
SPEAII	2001	Zitzler, Laumanns y Thiele	Algoritmo genético. General
PESAII	2001	Corne et al.	Algoritmo genético. General
ENGA	2001	Bagchi	Algoritmo genético. Específico
CMOGA	2001	Murata, Ishibuchi y Gen	Algoritmo genético. Específico
NSGAII	2002	Deb	Algoritmo genético. General
CNSGAII	2003	Deb	Algoritmo genético. General
$\varepsilon-$MOEA	2003	Deb	Algoritmo genético. General
$B-$IBEA	2004	Zitzler y Künzli	Algoritmo genético. General
$A-$IBEA	2004	Zitzler y Künzli	Algoritmo genético. General
MOSA_Suresh	2004	Suresh y Mohanasundaram	Recocido simulado. Específico
MOTS	2004	Armentano y Arroyo	Búsqueda tabú. Específico
$\varepsilon-$NSGAII	2005	Kollat y Reed	Algoritmo genético. General
MOGALS_Arroyo	2005	Arroyo y Armentano	Algoritmo genético. Específico
MOSAII	2005	Varadharajan y Rajendran	Recocido simulado. Específico
PGA_ALS	2006	Pasupathy, Rajendran y Suresh	Algoritmo genético. Específico
PILS	2007	Geiger	Iterated local search. Específico

Tabla 3.1: Métodos re-implementados para el taller de flujo multi-objetivo.

3.2. Detalles del banco de pruebas y de la evaluación computacional

Cada uno de los 23 algoritmos implementados se probó con un nuevo banco de pruebas. No existen bancos de pruebas exhaustivos en la literatura para el problema de PFSP multi-objetivo. La única referencia que hemos encontrado es el trabajo de Basseur (2005) donde se propone un pequeño conjunto de 14 instancias. Para poder llevar a cabo un análisis exhaustivo y sólido, se necesita un conjunto de instancias mucho mayor. Hemos modificado y aumentado las famosas instancias de Taillard (1993). Este banco de pruebas está organizado en 12 grupos que contienen 10 instancias cada uno, sumando un total de 120 instancias. Los grupos contienen diferentes combinaciones de número de trabajos n y cantidad de máquinas m. Las $n \times m$ combi-

naciones son: $\{20, 50, 100\} \times \{5, 10, 20\}$, $200 \times \{10, 20\}$ y 500×20. Los tiempos de proceso (p_{ij}) de las instancias de Taillard se generan a partir de una distribución uniforme con rangos $[1, 99]$. Hemos tomado las primeras 110 instancias, y eliminado las últimas 10 del grupo de 500×20 debido a que por su gran tamaño se hacen impracticables para los experimentos. Al elegir estas instancias podemos utilizar los mejores valores de $C_{\text{máx}}$ como valores de referencia para los métodos multi-objetivo. Con respecto a las fechas de entrega para el criterio de tardanza total, utilizamos el mismo enfoque dado en Hasija y Rajendran (2004). En este trabajo una fecha de entrega d_j ajustada se asigna a cada trabajo $j \in N$ mediante la siguiente expresión: $d_j = P_j \times (1 + random \cdot 3)$ donde $P_j = \sum_{i=1}^{m} p_{ij}$ es la suma de los tiempos de proceso del trabajo en todas las máquinas y $random$ es un número aleatorio uniformemente distribuido entre $[0, 1]$. Este método genera las fechas de entrega en fechas muy ajustadas o relativamente ajustadas dependiendo del valor de $random$ para cada trabajo. Por ejemplo si $random$ es cercano a 0, entonces la fecha de entrega del trabajo va a ser realmente ajustada ya que va a ser más o menos la suma de sus tiempos de proceso (en cada máquina). Como resultado, el trabajo deberá ser secuenciado tempranamente para evitar las tardanzas. Las 110 instancias modificadas están disponibles para ser descargadas en `http://soa.iti.es/instancias-de-problemas`.

Cada algoritmo ha sido cuidadosamente re-implementado siguiendo todas las explicaciones dadas por los autores en los artículos originales. Hemos re-implementado todos los algoritmos con el lenguaje Delphi 2007. Cabe señalar que todos los algoritmos comparten la mayor parte de las estructuras y funciones y el mismo nivel de codificación ha sido utilizado, compartiendo todos ellos optimizaciones de código y aceleraciones en la programación. El método de ordenación rápida no dominada (*Fast Non Dominated Sorting*) se usa en muchos de los métodos. A no ser que haya sido indicado de otra manera por los autores en los artículos originales, los operadores de mutación y cruce usados

por los métodos genéticos son la mutación por inserción y el cruce por dos puntos. A no ser que haya sido explícitamente indicado, todos los algoritmos utilizan un procedimiento de eliminación de soluciones duplicadas en la población, como así también en los repositorios de soluciones no dominadas.

El criterio de parada para casi todos los algoritmos está dado por un tiempo límite dependiente del tamaño de la instancia. Detenemos los algoritmos en el momento en que el tiempo de CPU es igual a $n \cdot m/2 \cdot t$ milisegundos, donde t es un parámetro de entrada. Otorgar más tiempo a las instancias grandes es una manera natural de separar los resultados obtenidos por el algoritmo de la escurridiza variable "tiempo de CPU". De lo contrario, si se obtuvieran peores resultados para instancias grandes, no habría manera de decir si se deben a un escaso tiempo de CPU o al propio tamaño de la instancia. Cada algoritmo se ejecuta 10 veces (réplicas) en forma independiente para cada instancia con tres criterios de parada distintos, $t = 100$, 150 y 200 milisegundos. Esto significa que para las instancias más grandes de 200×20 se utiliza un total de 400 segundos de tiempo real de CPU. Para cada instancia, tiempo de parada y réplica, hemos utilizado la misma semilla aleatoria. Esta es una técnica de reducción de varianza conocida que fue propuesta por McGeoch (1992).

Para las pruebas hemos utilizado combinaciones de tres de los objetivos más usados en la literatura referente al PFSP: $C_{\max}$, tardanza total y tiempo total de flujo. Estas combinaciones se conforman de la siguiente manera: $C_{\max}$ y tardanza total, $C_{\max}$ y tiempo total de flujo y la tercera combinación es tiempo total de flujo y tardanza total. Esto hace un total de 3 combinaciones diferentes de objetivos.
Cada algoritmo se compiló y ejecutó con la bandera de optimización de código de Delphi 2007 habilitada, en un grupo de 12 computadoras idénticas, con un procesador Intel Core 2 Duo E6600 de 2.4 Ghz y 1 GByte de memoria RAM cada una. Para las pruebas, cada algoritmo, instancia y réplica se

asigna aleatoriamente a una computadora y luego los resultados se recolectan. Considerando los 23 algoritmos, 110 instancias, 10 réplicas por instancia, tres combinaciones de objetivos y tres criterios de parada diferentes, un total de 227.700 resultados se recolectan al final de las pruebas. En realidad, cada resultado es una aproximación a la frontera de Pareto, que contiene un conjunto de vectores con los valores de los objetivos. Los experimentos han requerido aproximadamente 1.780 horas de tiempo de CPU en total.

Con las $23 \cdot 10 \cdot 3 = 690$ aproximaciones a la frontera de Pareto encontradas para cada instancia, se lleva a cabo una ejecución del algoritmo de FNDS (*Fast Non Dominated Sorting*) para obtener y guardar la "mejor" frontera de Pareto obtenida. Estas "mejores" 330 fronteras de Pareto (110 por cada combinación de objetivos) se guardaron para el futuro uso de la comunidad científica y están disponibles en `http://soa.iti.es/instancias-de-problemas`. Estas fronteras también se usaron para obtener los puntos de referencia necesarios para calcular el indicador de hipervolumen (I_H) y como conjunto de referencia en el cálculo del indicador multiplicativo épsilon (I_ε).

3.3. Resultados de los experimentos

Los resultados de los experimentos se muestran a continuación, agrupados de acuerdo a cada una de las diferentes combinaciones de objetivos utilizadas en las pruebas. En cada caso se analizan los resultados y presentan las tablas y gráficas explicativas correspondientes.

3.3.1. Resultados para $C_{\text{máx}}$ y para tardanza total

De acuerdo a la revisión llevada a cabo en el Capítulo 2, el $C_{\text{máx}}$ y la tardanza total son dos criterios de optimización muy comunes. Un $C_{\text{máx}}$ bajo incrementa el aprovechamiento de las máquinas y el rendimiento. Sin embargo, para alcanzar el mejor $C_{\text{máx}}$ posible, puede ser necesario incumplir

fechas de entrega y por lo tanto ambos objetivos no están correlacionados. Hemos realizado pruebas con los 23 algoritmos re-implementados con estos dos criterios. La tabla 3.2 muestra los valores medios del hipervolumen (I_H) y del indicador épsilon (I_ε) para todos los algoritmos. Los resultados se han dividido de acuerdo a los tres criterios de parada diferentes, según el tiempo de CPU. Al mismo tiempo, los resultados están ordenados en forma descendiente por su valor de hipervolumen.

	Tiempo	100			150			200	
#	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
1	MOSAII	**1,366**	**1,049**	MOSAII	**1,360**	**1,053**	MOSAII	**1,357**	**1,056**
2	MOGALS	1,290	1,089	MOGALS	1,299	1,082	MOGALS	1,301	1,080
3	PESA	1,285	1,086	PESA	1,287	1,084	PESA	1,291	1,081
4	PESAII	1,280	1,089	PESAII	1,284	1,086	PESAII	1,288	1,084
5	PGA_ALS	1,273	1,116	PGA_ALS	1,267	1,116	PGA_ALS	1,255	1,118
6	MOTS	1,224	1,135	MOTS	1,232	1,131	MOTS	1,235	1,129
7	MOGA_Murata	1,178	1,148	MOGA_Murata	1,187	1,143	MOGA_Murata	1,195	1,138
8	CMOGA	1,150	1,162	CMOGA	1,169	1,150	CMOGA	1,181	1,143
9	NSGAII	1,142	1,168	NSGAII	1,156	1,160	NSGAII	1,165	1,155
10	SPEA	1,141	1,169	SPEA	1,154	1,161	SPEA	1,164	1,155
11	CNSGAII	1,137	1,169	CNSGAII	1,153	1,160	CNSGAII	1,162	1,154
12	$\varepsilon-$NSGAII	1,091	1,195	$\varepsilon-$NSGAII	1,106	1,187	$\varepsilon-$NSGAII	1,115	1,182
13	$(\mu+\lambda)-$PAES	1,086	1,197	$(\mu+\lambda)-$PAES	1,101	1,188	$(\mu+\lambda)-$PAES	1,110	1,183
14	PAES	1,049	1,219	$\varepsilon-$MOEA	1,051	1,222	$\varepsilon-$MOEA	1,061	1,216
15	$\varepsilon-$MOEA	1,045	1,225	PAES	1,035	1,227	PAES	1,028	1,230
16	MOSA_Suresh	0,976	1,290	MOSA_Suresh	0,960	1,301	MOSA_Suresh	0,955	1,303
17	SA_Chakravarty	0,894	1,395	SA_Chakravarty	0,882	1,402	SA_Chakravarty	0,870	1,410
18	PILS	0,803	1,409	PILS	0,843	1,379	PILS	0,866	1,363
19	ENGA	0,588	1,522	$A-$IBEA	0,592	1,518	$A-$IBEA	0,599	1,514
20	$A-$IBEA	0,578	1,528	ENGA	0,570	1,534	ENGA	0,559	1,542
21	SPEAII	0,537	1,544	SPEAII	0,524	1,550	SPEAII	0,522	1,549
22	NSGA	0,520	1,571	NSGA	0,502	1,584	NSGA	0,490	1,593
23	$B-$IBEA	0,411	1,640	$B-$IBEA	0,411	1,641	$B-$IBEA	0,417	1,637

Tabla 3.2: Resultados para los criterios de $C_{\max}$ y tardanza total. Medias de los indicadores de calidad para los 23 algoritmos probados bajo los tres diferentes criterios de terminación. Cada valor medio se calcula entre 110 instancias y 10 réplicas por instancia (1.100 valores en total). Para cada criterio de terminación, los métodos están ordenados de acuerdo a I_H en forma descendente.

Comentamos ahora los resultados. En primera instancia, se puede observar que ambos indicadores son contradictorios en muy pocos casos. Mientras

los valores del hipervolumen decrecen, el indicador épsilon aumenta. Existen solamente algunas pocas excepciones y estas ocurren entre algoritmos que se ubican en forma consecutiva y con valores de hipervolumen muy similares, como por ejemplo el SPEA y el CNSGAII, que se encuentran en las posiciones 10 y 11 en las columnas para 200 ms. Otro resultado interesante es que el orden de los algoritmos se mantiene prácticamente inalterable al incrementar el tiempo de CPU permitido y cuando cambia lo hace en cantidades muy pequeñas. Por ejemplo, los algoritmos PGA_ALS y MOGALS básicamente no pueden comparase entre sí. Esto es porque para 100 ms de tiempo de CPU, el algoritmo PGA_ALS obtiene un hipervolumen marginalmente mejor, pero al mismo tiempo el algoritmo MOGALS da un indicador épsilon menor. Lo mismo ocurre con un tiempo de 150 ms. Solo en 200 ms el MOGALS obtiene un mejores valores en ambos indicadores. Sin embargo, estamos hablando de los valores medios observados entre todas las instancias, pero luego veremos que existen diferencias más pronunciadas cuando se miran instancias específicas.

Debido a que los valores de los objetivos están normalizados y las mejores soluciones son multiplicadas por $1,2$, el máximo valor hipotético para el hipervolumen es $1{,}2^2 = 1{,}44$. Podemos ver que los resultados obtenidos por MOSAII son muy cercanos a este valor para los tres criterios de parada. De la misma forma, el mínimo posible indicador épsilon es 1. Es interesante observar como los algoritmos PESA y PESAII mejoran los resultados de PGA_ALS y de MOGALS por un amplio margen, teniendo en cuenta que ambos (PESA y PESAII) son algoritmos multi-objetivo generales y que PGA_ALS y MOGALS son algoritmos específicos para el PFSP, incluso MOGALS fue propuesto para los dos objetivos que probados en los experimentos. También cabe destacar que el algoritmo MOGA_Murata está en la posición séptima y fue uno de los primeros algoritmos propuestos para el PFSP, hace casi 20 años. Este algoritmo logra obtener mejores resultados que el algoritmo CMOGA,

propuesto por los mismos autores, y supuestamente mejor que el anterior. Cabe recordar que nuestras re-implementaciones han sido llevadas a cabo de acuerdo con los detalles dados en los artículos revisados. El método CMOGA puede resultar mejor que el MOGA_Murata bajo otras circunstancias, como tiempos de CPU diferentes o problemas específicos. Sin embargo, para el caso de las pruebas extensas y cuidadosas que hemos llevado a cabo, no ha ocurrido así.

Los demás métodos específicamente propuestos para el PFSP (MOSA_Suresh, SA_Chakravarthy, PILS y ENGA) ocupan posiciones desfavorables. El mal rendimiento del SA_Chakravarthy es de esperar, ya que este método usa un enfoque *a priori* ponderando los objetivos y en las pruebas fue corrido 100 veces variando las ponderaciones para intentar obtener una frontera de Pareto, metodología que es muy poco recomendable, por dar resultados pobres. Como indican los autores, ENGA mejora los resultados de NSGA, pero sin embargo funciona significativamente peor que otros algoritmos más antiguos como MOGA_Murata. Otro resultado no muy alentador es el pobre rendimiento del método PILS, a pesar de ser más reciente que los demás. Básicamente el PILS es un procedimiento de mejora iterativa y es extremadamente costoso en términos de tiempo de CPU. Esta es la razón por lo que la mayoría de los demás métodos obtienen mejores resultados en nuestro entorno experimental.

No todos los algoritmos se pueden detener por el tiempo de CPU que se ha dado como criterio de parada. Algunos métodos llevan a cabo búsquedas locales luego de completar las iteraciones, y otros simplemente no se pueden modificar para lograr que se detengan en un momento dado. En cualquier caso, todos los tiempos de CPU se mantienen en un intervalo. Por ejemplo, para 100 milisegundos y para las instancias más grandes probadas, las de 200×20, todos los métodos deberían detenerse a los $3,33$ minutos. Sin embargo MOGALS se detiene a los $8,37$ minutos, MOTS a los $3,63$, PILS a $3,36$, SA_Chakravarty

a $1,39$, MOSA_Suresh a $12,8$ y MOSAII a $2,13$. Es muy interesante observar que el método MOSAII, además de ser el que obtiene mejores resultados, es el más veloz y otros métodos como MOSA_Suresh necesitan una mayor cantidad de tiempo para obtener resultados peores. Todos los demás métodos pueden ser controlados para detenerse a un tiempo de CPU específico. Estas discrepancias en el tiempo de parada son importantes solo para las instancias grandes, ya que para resolver las más pequeñas todos los métodos utilizan más o menos el mismo tiempo de CPU.
La tabla 3.2 solo contiene los valores medios, y muchos de ellos son muy parecidos. A pesar de que cada promedio está compuesto de un gran número de puntos de datos, todavía es necesario llevar a cabo un análisis estadístico extenso para evaluar si las diferencias entre los valores medios son realmente estadísticamente significativas.

Hemos llevado a cabo un total de 12 experimentos diferentes, utilizando la metodología de Montgomery (2009). Llevamos a cabo el diseño de experimento (DOE) y el análisis paramétrico ANOVA, como así también pruebas no paramétricas basadas en rangos de Friedman, para ambos indicadores de calidad (I_H y I_ϵ) y para los tres criterios de parada diferentes. Utilizar pruebas paramétricas y no paramétricas al mismo tiempo ofrece varias ventajas. Primero, en la literatura de Investigación Operativa es común descartar las pruebas paramétricas debido al hecho de que este tipo de pruebas se basa en algunos supuestos que los datos deben satisfacer. Se prefieren las pruebas no paramétricas debido a que son "no dependientes de distribución". Sin embargo, las pruebas no paramétricas no son tan potentes como las paramétricas. En segundo lugar, en las pruebas no paramétricas se pierde mucha información, ya que los datos deben ser clasificados por rangos y los diferentes valores (pequeños o grandes) son transformados en un valor de rango. Los métodos de re-muestreo como por ejemplo, las pruebas de permutación, están libres de suposiciones y no se basan en las trasformaciones de rangos. Sin embargo en nuestro caso,

estas pruebas son intensivas desde el punto de vista computacional, debido a que millones de permutaciones deberían ser probadas. Tercero, las técnicas ANOVA permiten un estudio mucho más rico y profundo de los datos. Por ello también comparamos ambas técnicas con la finalidad de respaldar estas afirmaciones. Para más información el lector puede consultar los trabajos de Montgomery (2009) y Conover (1999).
Llevamos a cabo seis ANOVA de múltiples factores donde el tipo de instancia es un factor controlado con 11 niveles (instancias desde 20×5 a 200×20). El algoritmo es otro factor controlado con 23 niveles. La variable respuesta en cada experimento es el hipervolumen o el indicador épsilon. Finalmente, hay un conjunto de experimentos para cada criterio de parada. Considerando que cada experimento tiene $25{,}300$ puntos de datos, las tres hipótesis principales de ANOVA: normalidad, homoscedasticidad e independencia de los residuos se satisfacen con facilidad y se confirman mediante el estudio que realizamos con este motivo.

Para comparar resultados, llevamos a cabo un segundo grupo de seis experimentos. En este caso se trata de pruebas no paramétricas de Friedman. Como hay 23 algoritmos y 10 réplicas diferentes, los resultados para cada instancia se clasifican en rangos entre 1 y 230. Un rango de uno representa el mejor resultado para el hipervolumen o el indicador épsilon. Realizamos varias pruebas estadísticas con cada conjunto de resultados (por ejemplo, para 100 ms de tiempo de CPU realizamos pruebas ANOVA y de Friedman en ambos indicadores de calidad). Por lo tanto, una corrección en el nivel de confianza se hace necesaria, debido a que el mismo conjunto de datos se usa para hacer más de una inferencia. Tomamos entonces el enfoque más conservador, que es el ajuste de Bonferroni, y ponemos el nivel de confianza ajustado α_s a $\frac{\alpha}{4} = \frac{0{,}05}{4} \simeq 0{,}01$. Esto significa que todas las pruebas se llevan a cabo a un nivel de confianza ajustado de 99 % para un nivel de confianza real de 95 %.

Las figuras 3.15 y 3.16 muestran la gráfica de media para el factor algoritmo en el ANOVA para la variable repuesta del indicador épsilon y la gráfica de medias para los rangos del indicador épsilon, respectivamente. Ambas figuras se refieren al criterio de parada $t = 100$. Para las pruebas paramétricas utilizamos los intervalos HSD (*Honest Significant Difference*) de Tukey que compensan el sesgo producido en comparaciones múltiples. Intervalos MSD similares son utilizados para las pruebas no paramétricas.

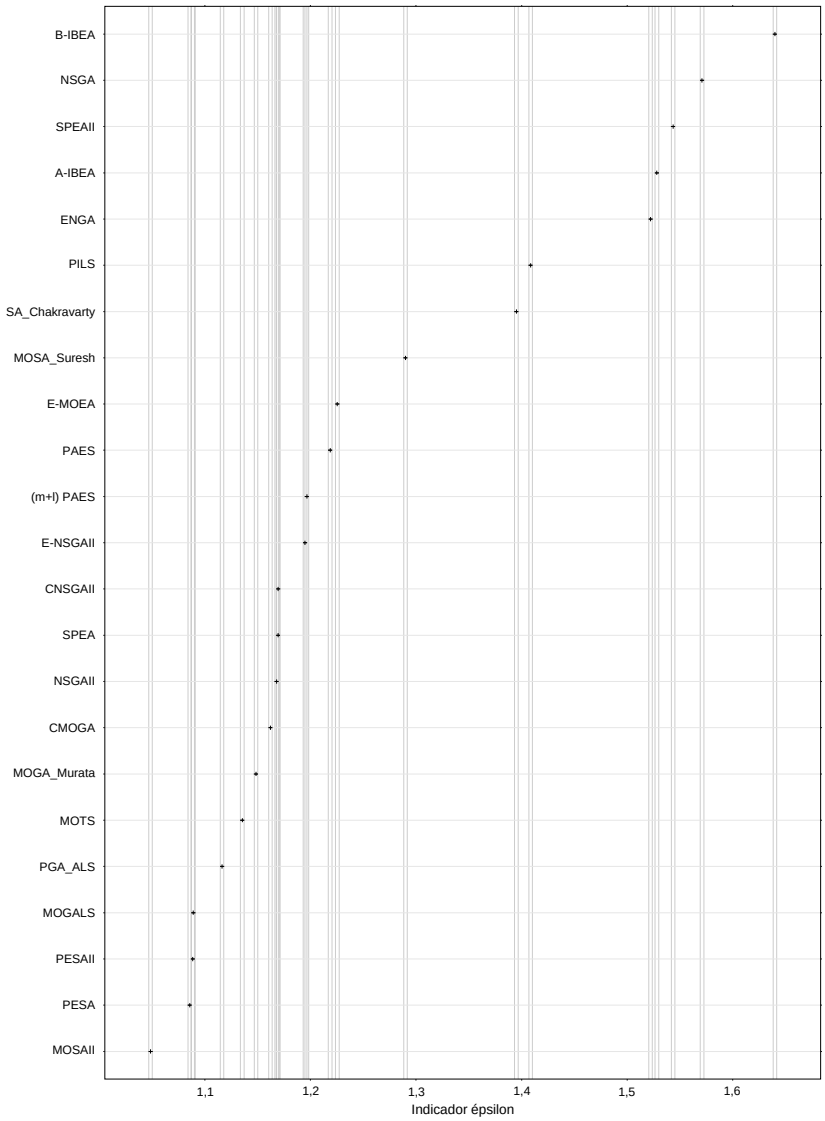


Figura 3.2: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador épsilon y criterio de parada $t = 100$.

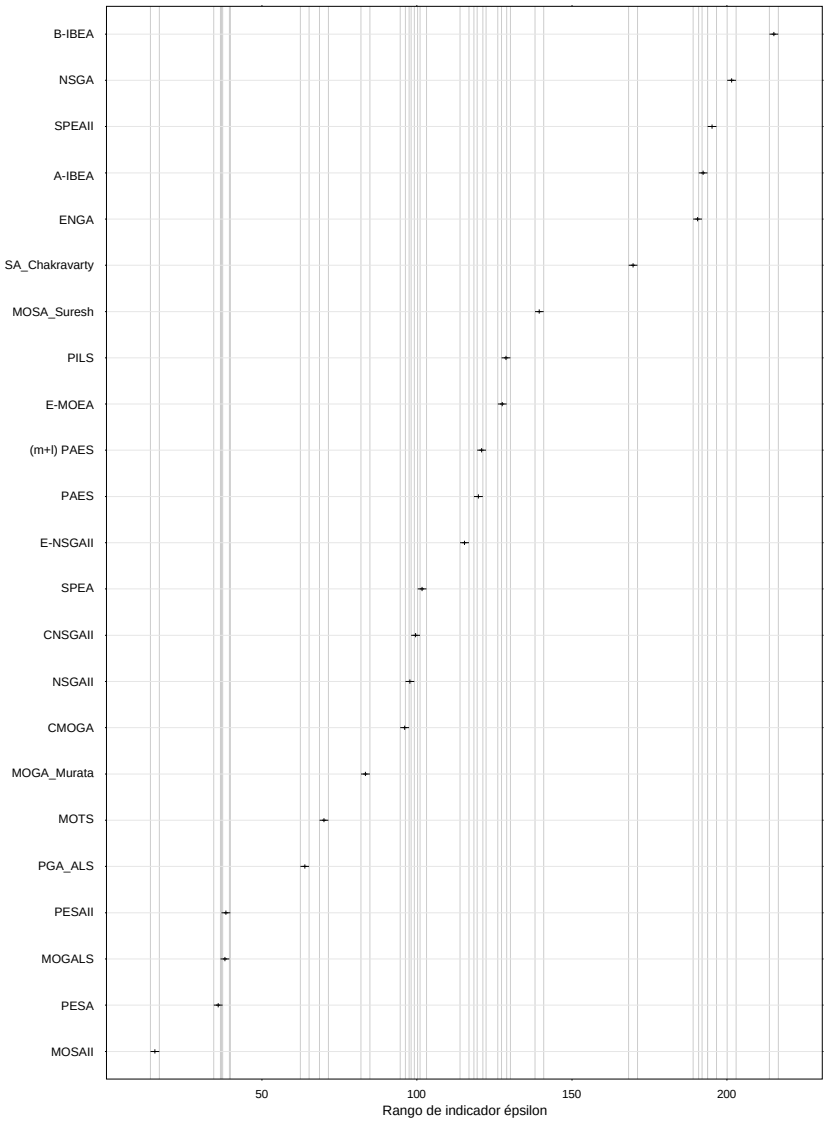


Figura 3.3: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta rango de indicador épsilon y criterio de parada $t = 100$.

Puede observarse que las pruebas no paramétricas son menos potentes. Además de que los intervalos son mucho más amplios (recordemos que intervalos solapados indican una diferencia no significativa desde el punto de vista estadístico), la clasificación por rangos no tiene en cuenta las diferencias en las variables respuesta. PILS tiene un mejor rango que MOSA_Suresh y SA_Chakravarthy a pesar que ya habíamos observado que los valores medios de hipervolumen y de indicador épsilon son menores. La razón para este comportamiento es que PILS obtiene mejores resultados en muchas instancias pequeñas con una pequeña diferencia en el indicador épsilon respecto de MOSA_Suresh y de SA_Chakravarthy. Sin embargo, para instancias grandes los resultados obtenidos son mucho peores. Cuando se transforma esto en rangos, PILS obtiene un mejor rango más veces y por lo tanto parece ser mejor, cuando en la realidad es marginalmente mejor más veces y al mismo tiempo significativamente peor en otras ocasiones. Para concluir nuestra discusión entre pruebas paramétricas contra no paramétricas, debemos señalar que si las hipótesis son satisfechas (incluso si no son estrictamente satisfechas, como se explica en Montgomery, 2009) es mucho mejor utilizar pruebas estadísticas paramétricas. Cabe destacar que estamos utilizando conjuntos de datos muy extensos, con conjuntos de datos medianos o pequeños, la potencia de las pruebas no paramétricas disminuye significativamente.

El orden relativo, así como la mayoría de las diferencias observadas en los algoritmos son estadísticamente significativas como puede verse en la Figura 3.2. De hecho, las únicas diferencias no significativas, desde el punto de vista estadístico, se dan entre $(\mu + \lambda)-$PAES y $\varepsilon-$NSGAII y entre CNSGAII, SPEA y NSGAII. PESA y PESAII son también equivalentes.

La Figura 3.4 muestra los resultados paramétricos para el indicador de calidad de hipervolumen con 100 ms de tiempo de CPU.

Cabe destacar que, en el caso de las gráficas en que la variable respuesta es el indicador de hipervolumen, el eje Y está invertido ya que un mayor hipervolumen indica mejores resultados.

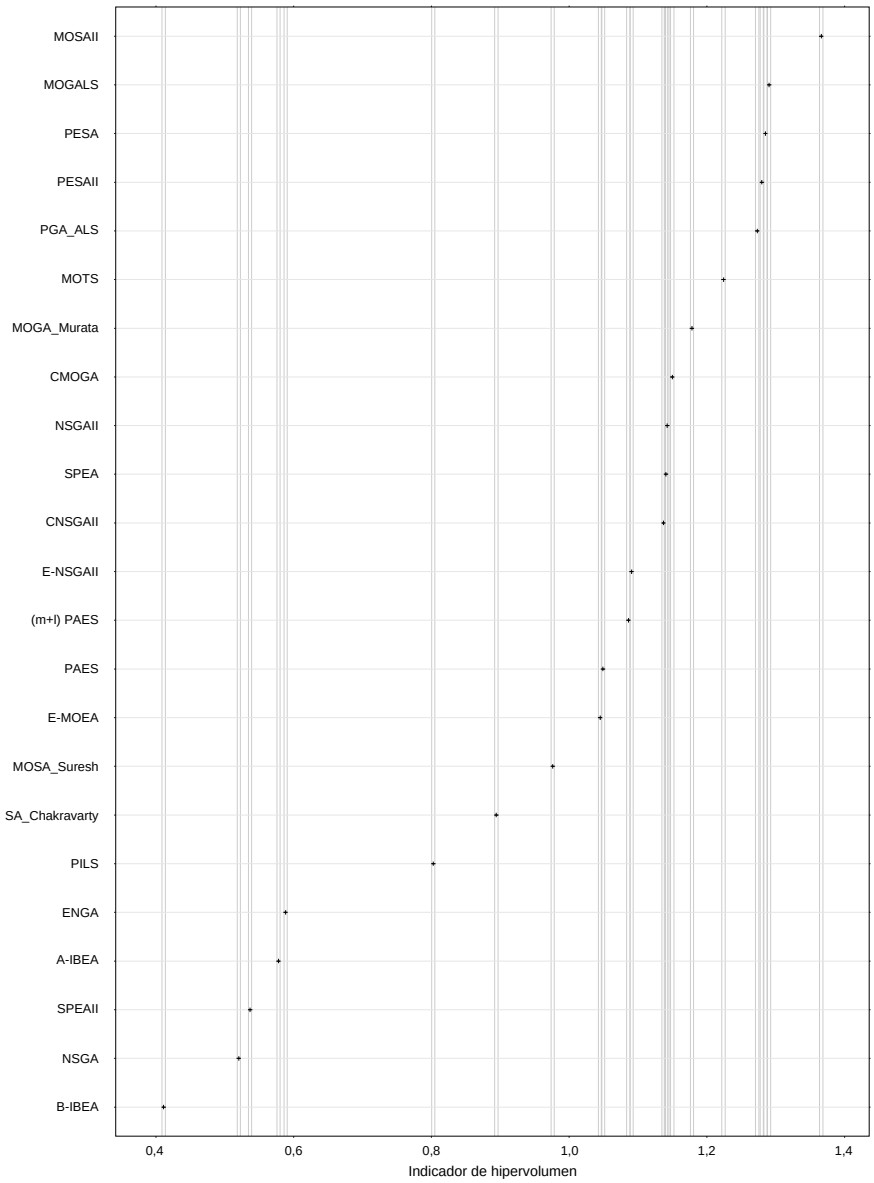


Figura 3.4: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador de hipervolumen y criterio de parada $t = 100$.

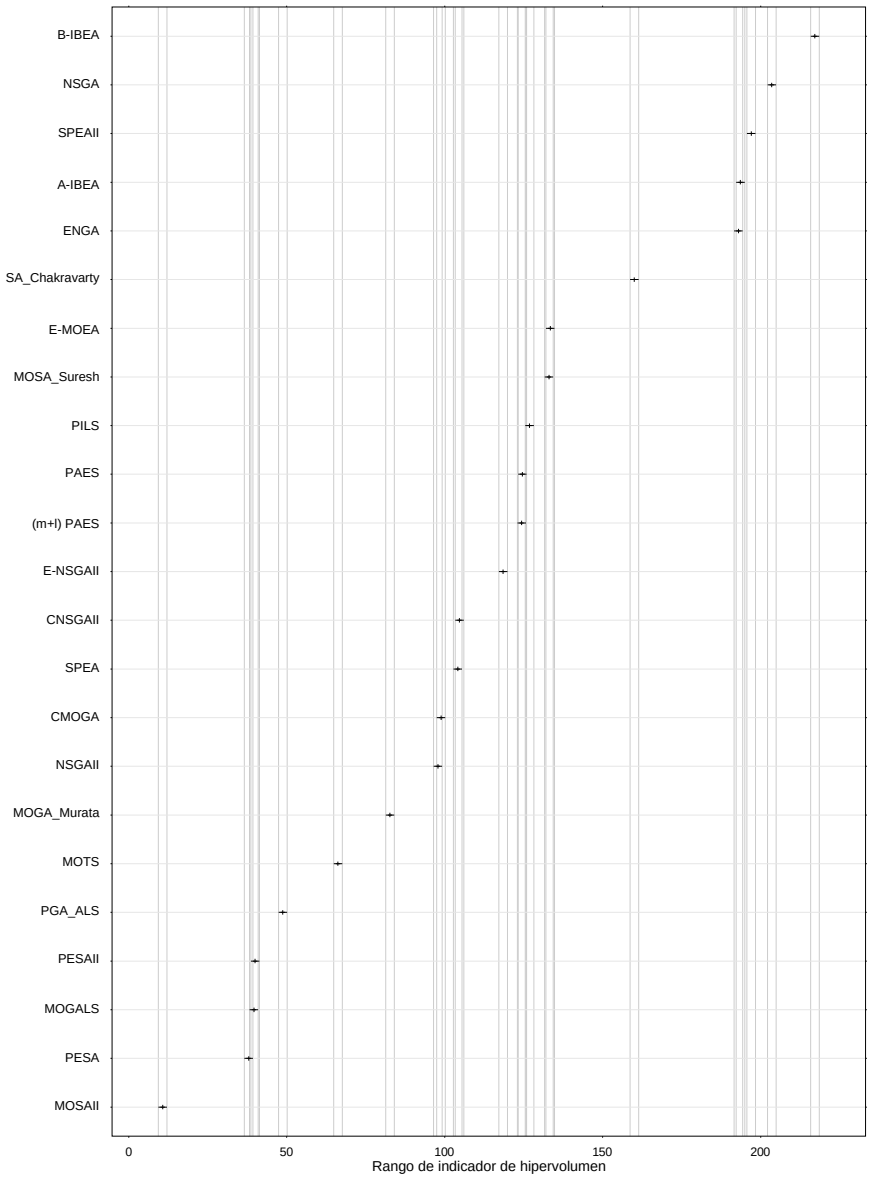


Figura 3.5: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador de hipervolumen y criterio de parada $t = 100$.

Hemos realizado los mismos experimentos con diferentes valores de tiempos de parada para observar si los resultados obtenidos son independientes del tiempo de ejecución de los algoritmos. En las siguientes gráficas se muestran los resultados de los tests paramétricos y no paramétricos para el tiempo de parada $t = 150$ y $t = 200$ y para el indicador de hipervolumen y el indicador épsilon.

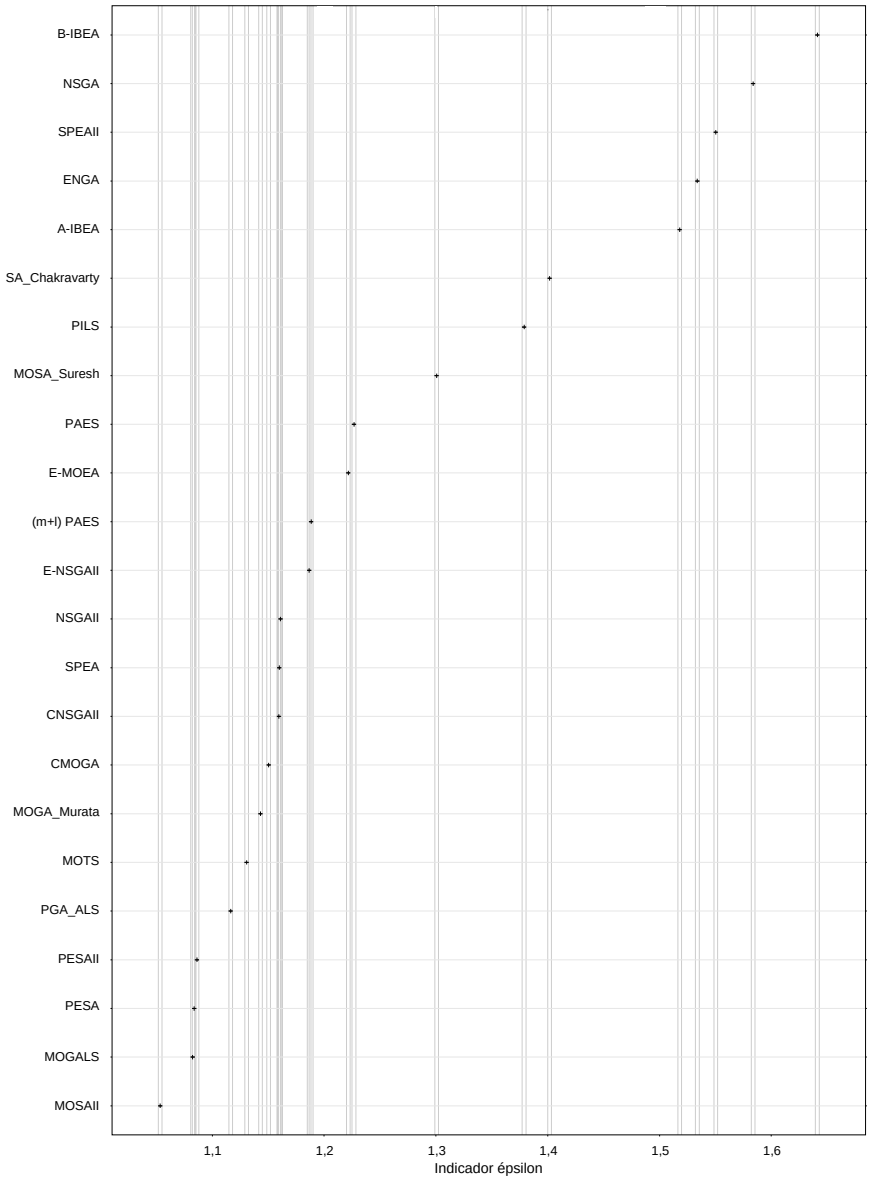


Figura 3.6: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador épsilon y criterio de parada $t = 150$.

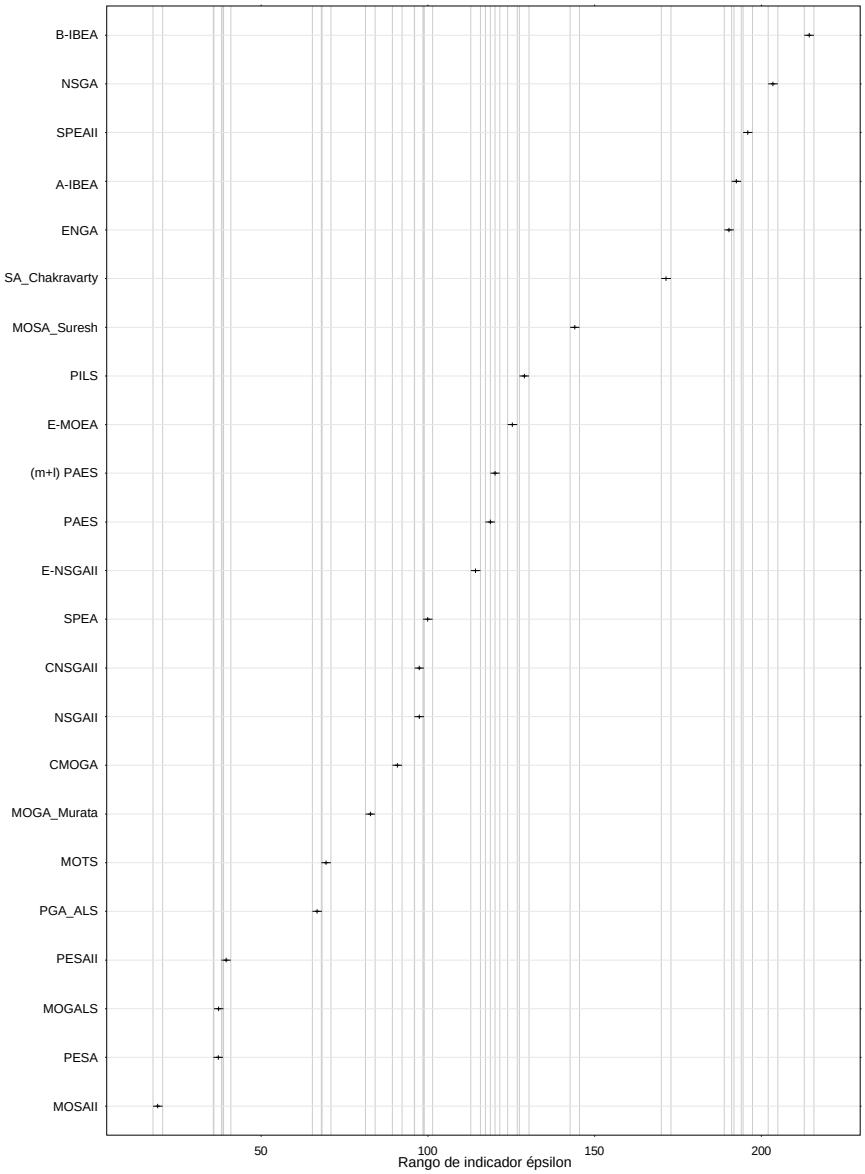


Figura 3.7: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador épsilon y criterio de parada $t = 150$.

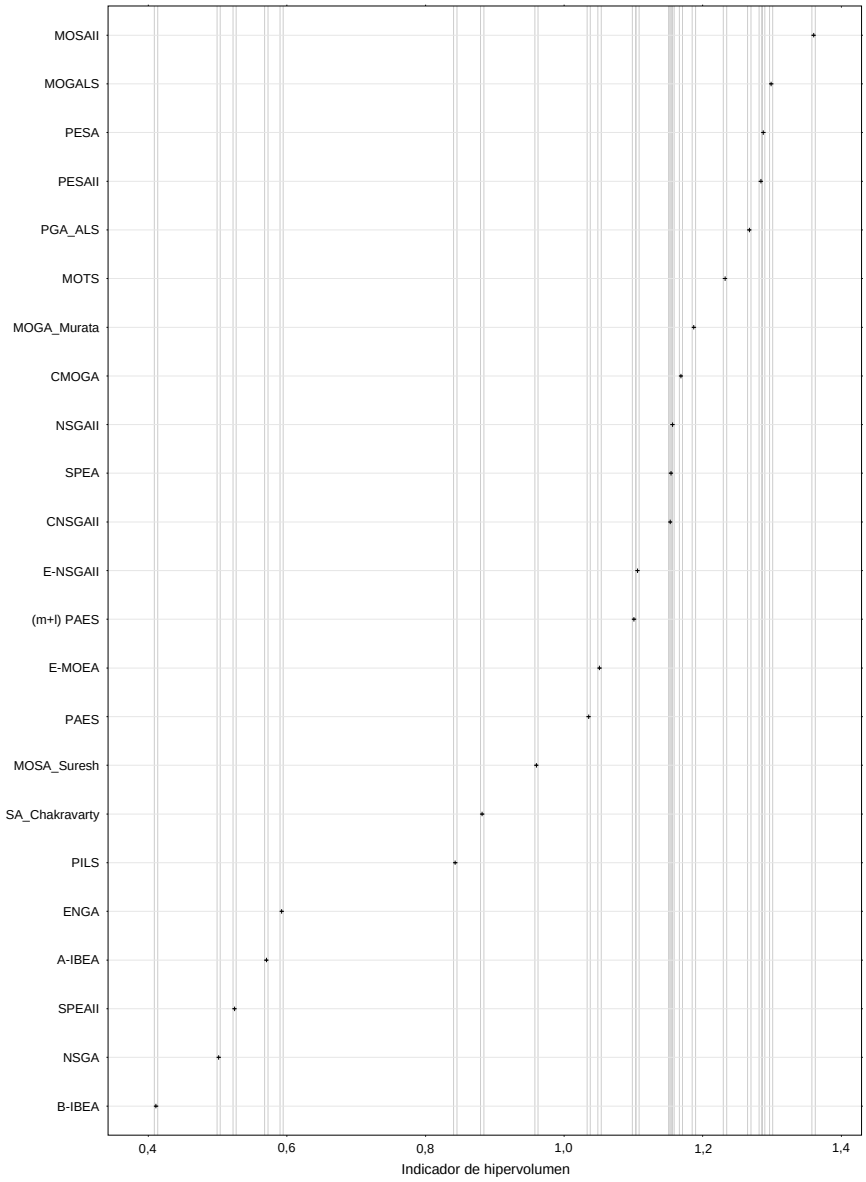


Figura 3.8: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador de hipervolumen y criterio de parada de $t = 150$.

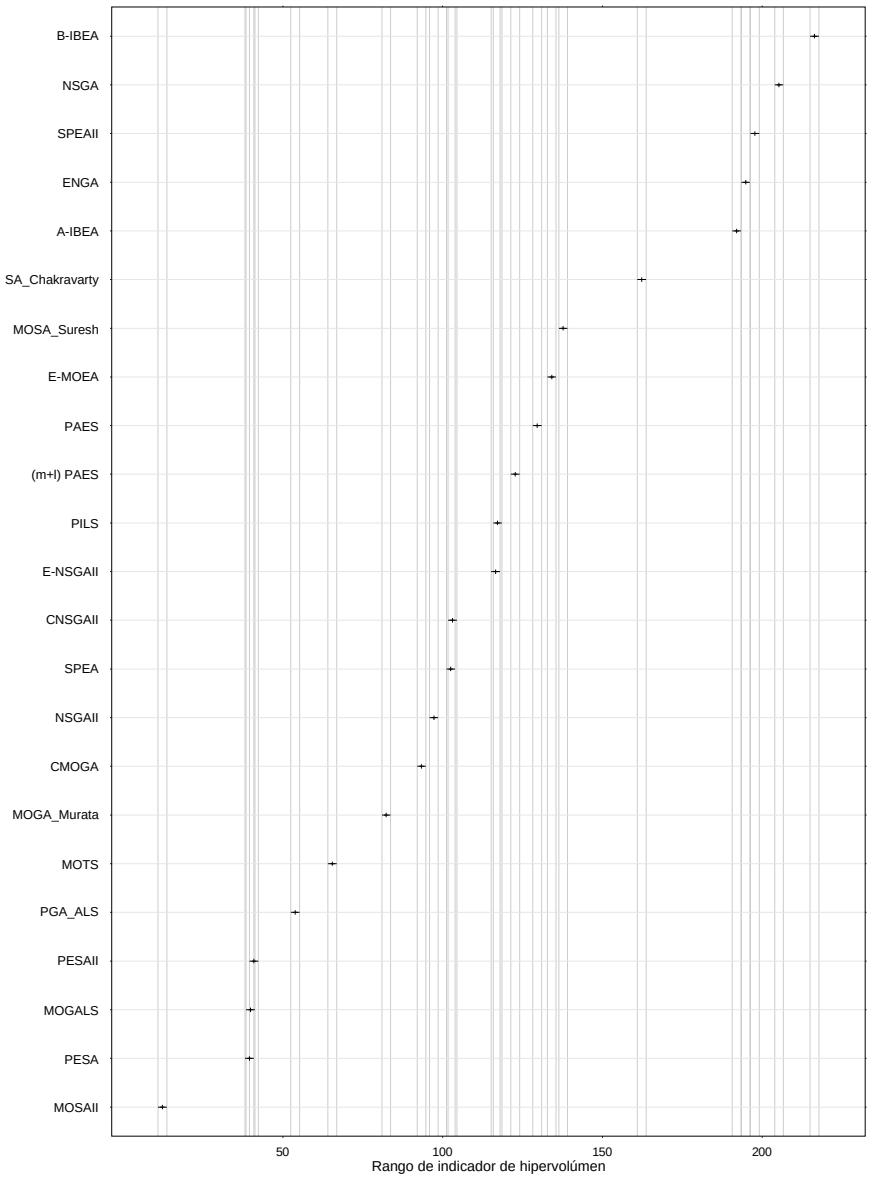


Figura 3.9: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador de hipervolumen y criterio de parada $t = 150$.

El orden relativo de los algoritmos es casi el mismo que en la Figura 3.2, la única diferencia se da entre los algoritmos MOGALS y PGA_ALS. Esto indica que estos dos algoritmos tienen un rendimiento muy similar y en promedio son incomparables. Un análisis más profundo, instancia por instancia, sería necesario para poder compararlos.
Como hemos visto, incrementar el tiempo de CPU no cambia, en promedio, el orden relativo de los algoritmos. Las siguientes figuras muestran los resultados paramétricos y no paramétricos para el tiempo de CPU 200 ms y para ambos indicadores.

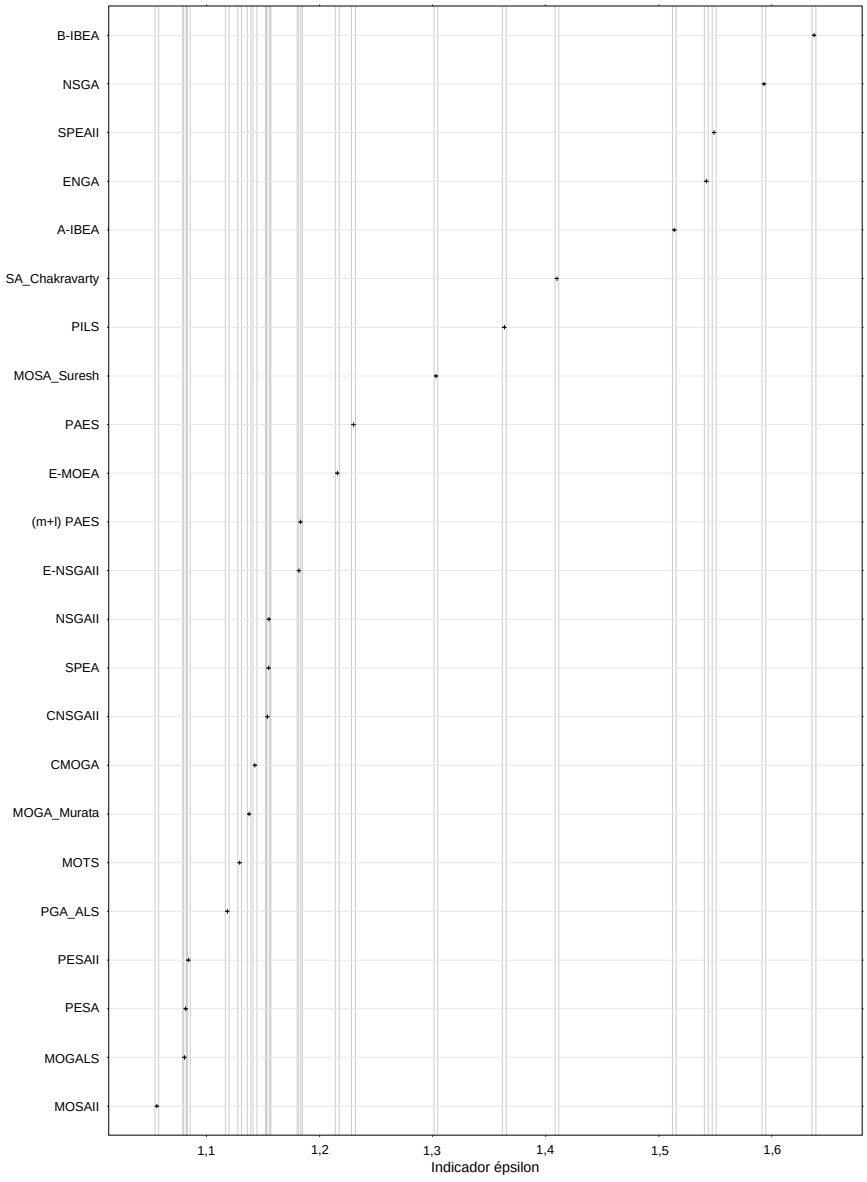


Figura 3.10: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador épsilon y criterio de parada de $t = 200$.

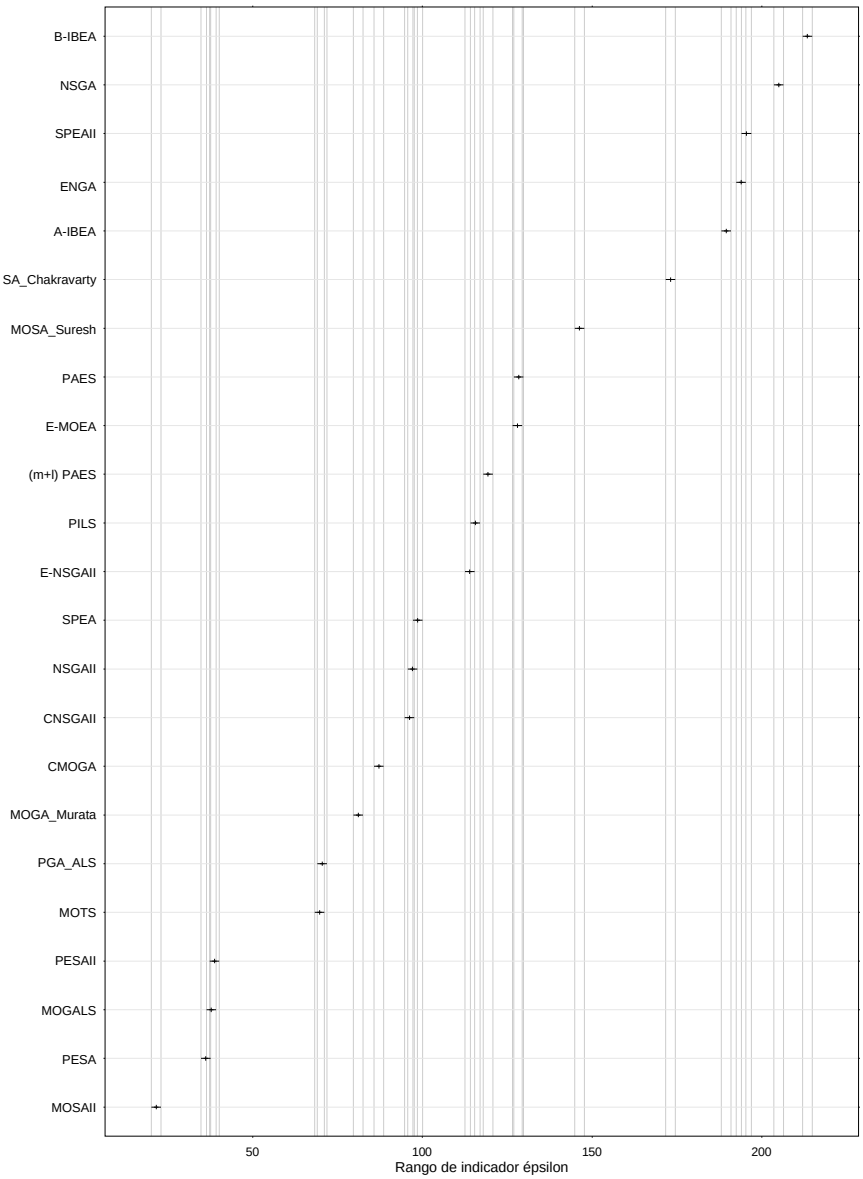


Figura 3.11: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador épsilon y criterio de parada de $t = 200$.

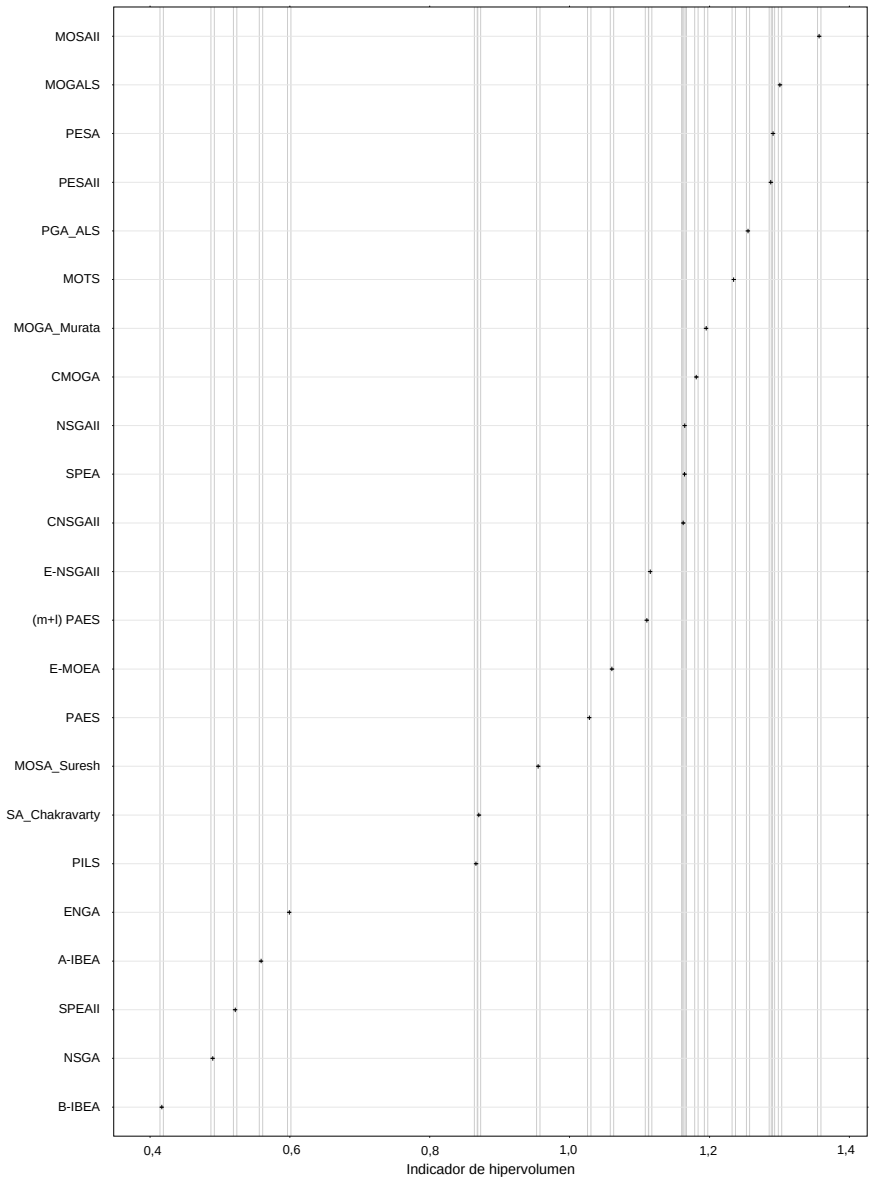


Figura 3.12: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador de hipervolumen y criterio de parada $t = 200$.

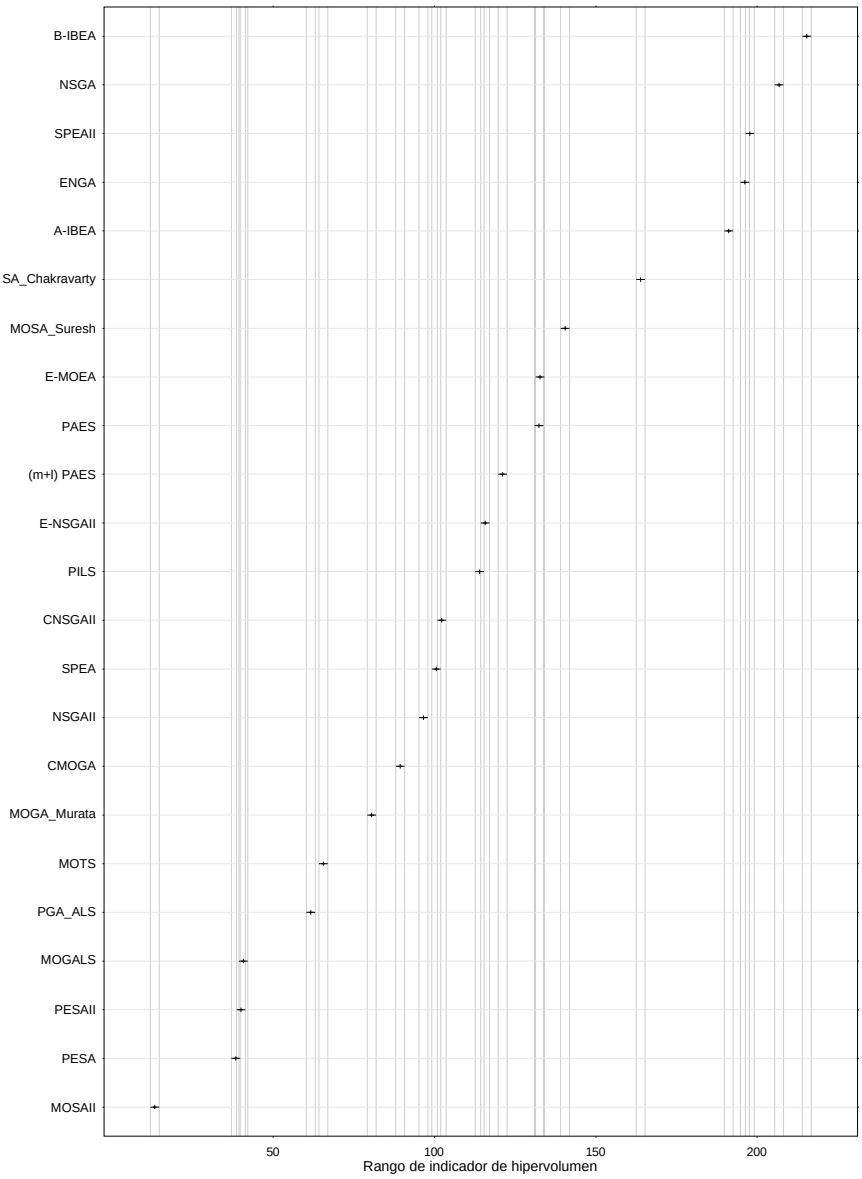


Figura 3.13: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador de hipervolumen y criterio de parada $t = 200$.

Podemos observar una mejora en PILS pero a pesar de que hay un decrecimiento relativo importante en el indicador épsilon, no es suficiente para mejorar los resultados de forma significativa. La mayor parte de los demás algoritmos mantienen sus posiciones relativas con pequeñas diferencias.
Hemos comentado con anterioridad que el rendimiento promedio no es constante para todos los tamaños de instancias. Por ejemplo, PILS puede ser un algoritmo muy bueno en problemas de tamaño reducido. La Figura 3.14 muestra los resultados paramétricos para el indicador épsilon con un tiempo de CPU de 200 ms, pero solo para las instancias de tamaño 50×5.

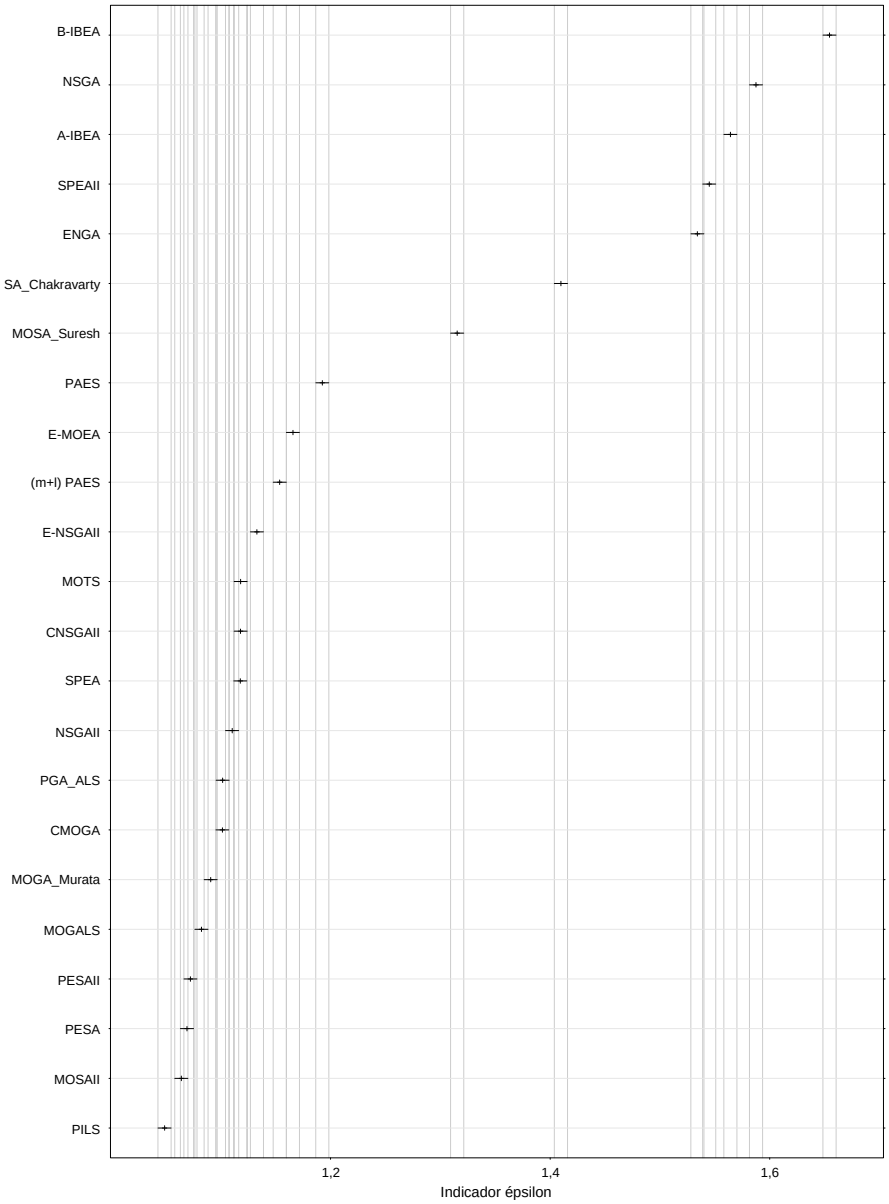


Figura 3.14: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Grupo de instancias de 50×5. Variable respuesta indicador épsilon y criterio de parada de $t = 200$.

Este grupo de instancias es "fácil" en el sentido que gran cantidad de algoritmos son capaces de obtener indicadores épsilon muy bajos. Más notable aun es que PILS mejora los resultados de MOSAII aunque por un margen pequeño. Este rendimiento, sin embargo, no es constante. Para instancias de tamaño 50×10, PILS se ubica en la tercera posición y para instancias de tamaño 100×5 PILS resulta ser uno de los peores algoritmos de la comparación. Un tema interesante para investigar es determinar los puntos fuertes del PILS y acelerarlo para mejorar su rendimiento en grupos de instancias grandes.

3.3.2. Resultados para $C_{\max}$ y tiempo total de flujo

En las siguientes páginas veremos los resultados obtenidos para los experimentos realizados para la combinación de objetivos de $C_{\max}$ y tiempo total de flujo. Igual que en la sección anterior, hemos realizado experimentos para 3 valores diferentes de t, $t = 100$, $t = 150$ y $t = 200$. Para cada experimento hemos obtenidos 3 conjuntos de resultados que se comparan en la tabla y las figuras de las páginas siguientes. En primer lugar, en la tabla se agrupan los resultados para los tres experimentos divididos en columnas, una columna por cada valor de t y debajo de cada una de ellas dos columnas para cada indicador.

	Tiempo	100			150			200	
#	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
1	MOSAII	**1,362**	**1,054**	MOSAII	**1,356**	**1,058**	MOSAII	**1,350**	**1,061**
2	MOGALS	1,337	1,066	MOGALS	1,337	1,064	MOGALS	1,336	1,064
3	MOTS	1,309	1,087	MOTS	1,312	1,085	MOTS	1,312	1,083
4	PGA_ALS	1,271	1,120	PESA	1,273	1,091	PESA	1,275	1,091
5	PESA	1,269	1,092	PGA_ALS	1,272	1,121	PGA_ALS	1,272	1,122
6	PESAII	1,262	1,095	PESAII	1,267	1,093	PESAII	1,268	1,093
7	MOGA_Murata	1,160	1,154	MOGA_Murata	1,171	1,148	MOGA_Murata	1,176	1,146
8	CMOGA	1,134	1,167	CMOGA	1,155	1,155	CMOGA	1,167	1,149
9	CNSGAII	1,111	1,180	CNSGAII	1,126	1,171	CNSGAII	1,133	1,168
10	NSGAII	1,106	1,184	NSGAII	1,121	1,177	NSGAII	1,128	1,173
11	SPEA	1,099	1,186	SPEA	1,116	1,176	SPEA	1,122	1,173
12	$(\mu+\lambda)-$PAES	1,059	1,206	$(\mu+\lambda)-$PAES	1,071	1,199	$(\mu+\lambda)-$PAES	1,077	1,196

Continúa en la siguiente página...

	Tiempo	100			150			200	
#	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
13	$\varepsilon-$NSGAII	1,051	1,213	$\varepsilon-$NSGAII	1,068	1,204	$\varepsilon-$NSGAII	1,077	1,199
14	$\varepsilon-$MOEA	1,027	1,233	$\varepsilon-$MOEA	1,036	1,228	$\varepsilon-$MOEA	1,040	1,226
15	PAES	1,005	1,238	PAES	0,992	1,244	PAES	0,980	1,252
16	MOSA_Suresh	0,950	1,296	MOSA_Suresh	0,936	1,303	MOSA_Suresh	0,922	1,314
17	SA_Chakravarty	0,813	1,410	PILS	0,836	1,383	PILS	0,861	1,367
18	PILS	0,794	1,413	SA_Chakravarty	0,798	1,422	SA_Chakravarty	0,781	1,431
19	ENGA	0,512	1,575	ENGA	0,492	1,590	ENGA	0,477	1,604
20	SPEAII	0,475	1,586	SPEAII	0,467	1,590	SPEAII	0,458	1,600
21	NSGA	0,446	1,625	$A-$IBEA	0,426	1,626	$B-$IBEA	0,425	1,631
22	$A-$IBEA	0,416	1,634	NSGA	0,426	1,642	$A-$IBEA	0,424	1,631
23	$B-$IBEA	0,416	1,634	$B-$IBEA	0,423	1,629	NSGA	0,409	1,658

Tabla 3.3: Resultados para los criterios de $C_{\text{máx}}$ y tiempo total de flujo. Medias de los indicadores de calidad para los 23 algoritmos probados bajo los tres diferentes criterios de terminación. Cada valor medio se calcula entre 110 instancias y 10 réplicas por instancia (1.100 valores en total). Para cada criterio de terminación, los métodos están ordenados de acuerdo a I_H en forma descendente.

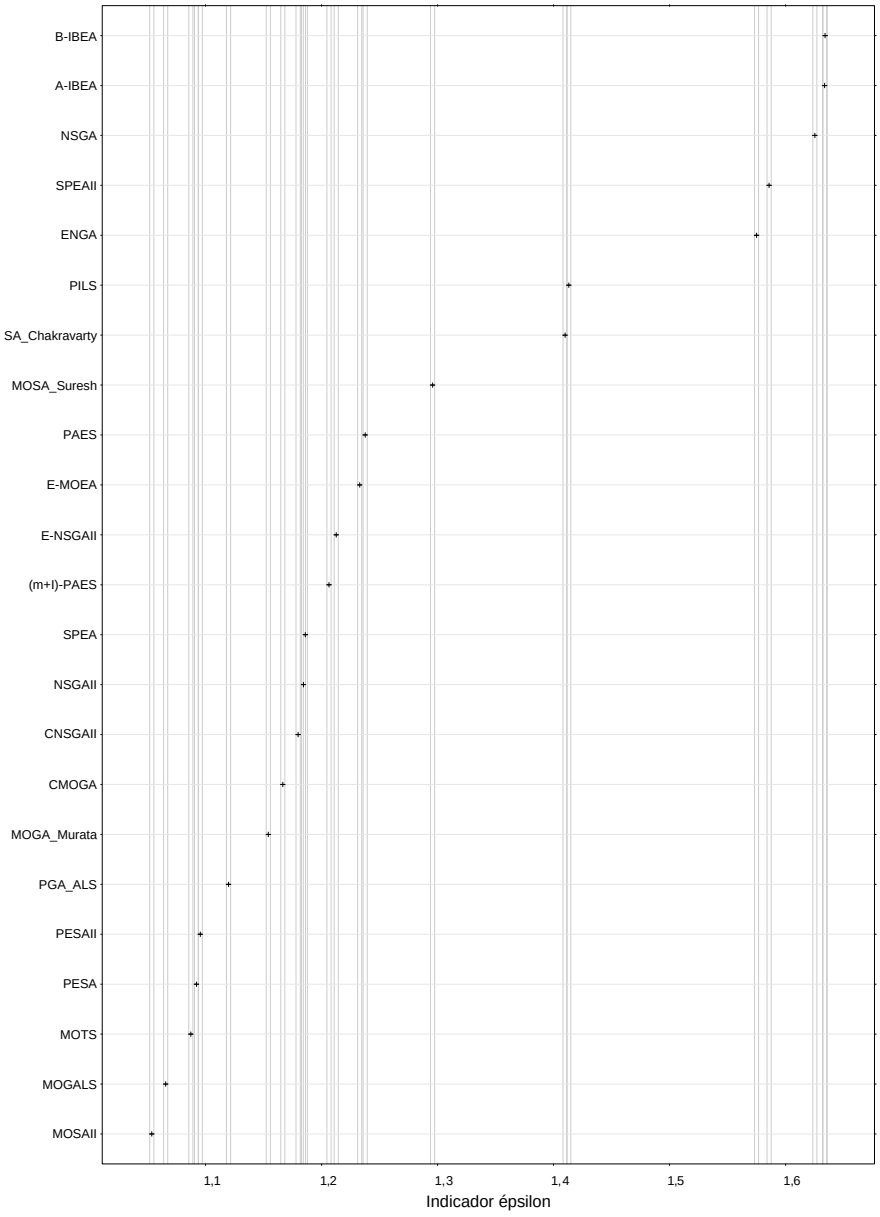


Figura 3.15: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador épsilon y criterio de parada $t = 100$.

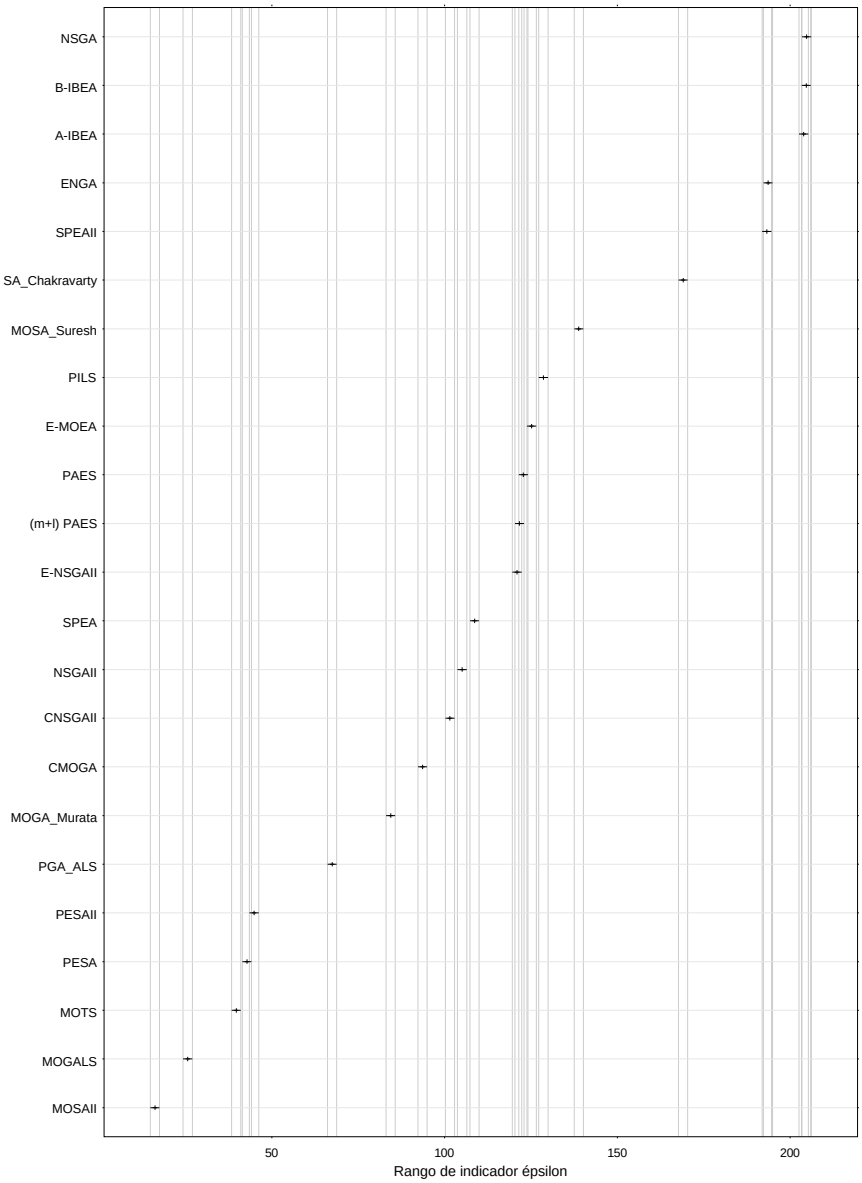


Figura 3.16: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador épsilon y criterio de parada $t = 100$.

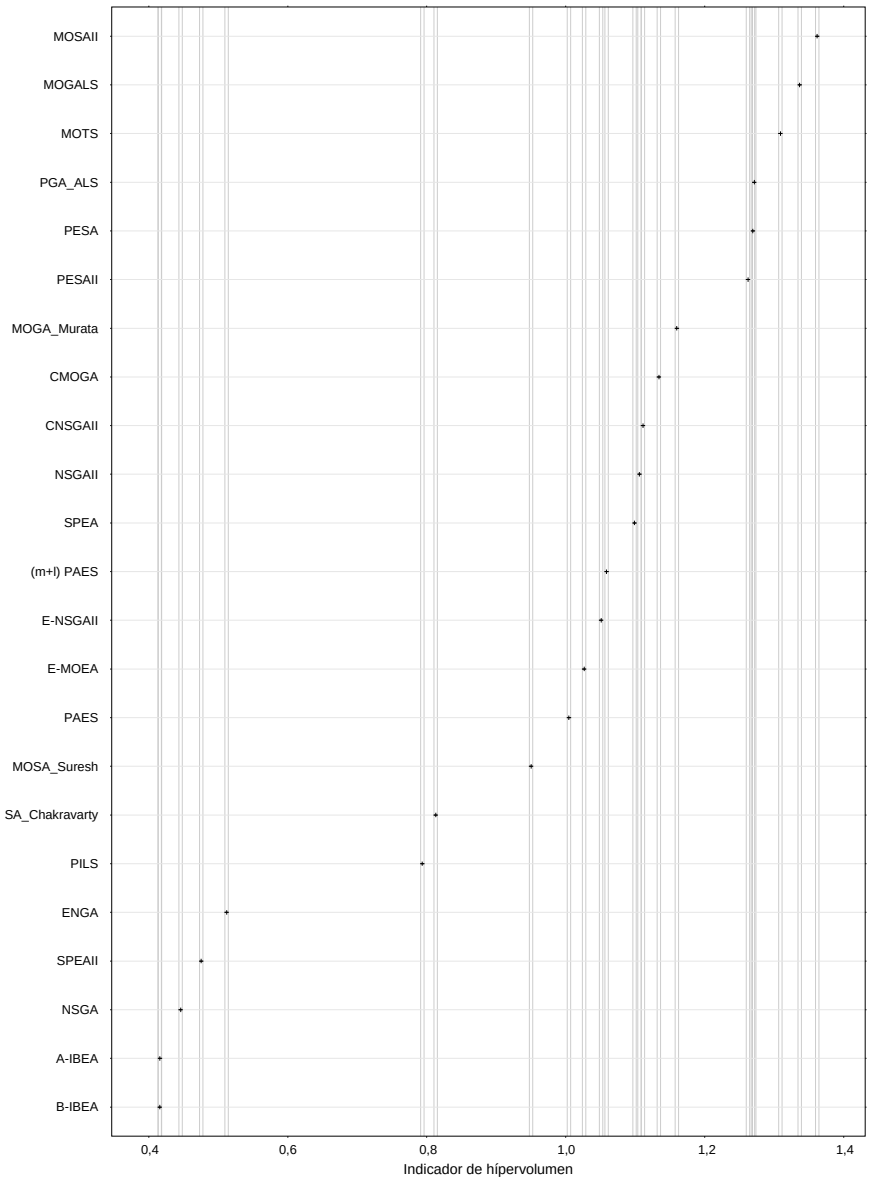


Figura 3.17: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador de hipervolumen y criterio de parada $t = 100$.

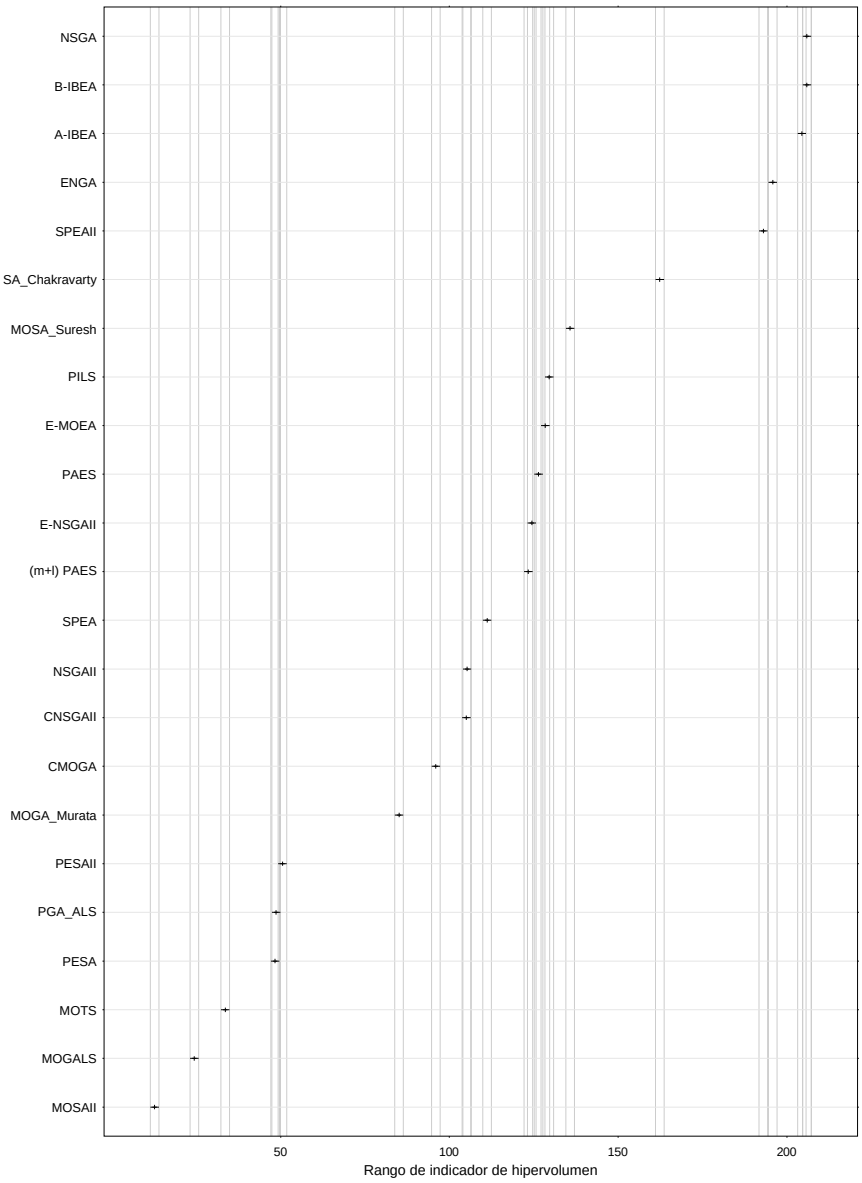


Figura 3.18: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador de hipervolumen y criterio de parada $t = 100$.

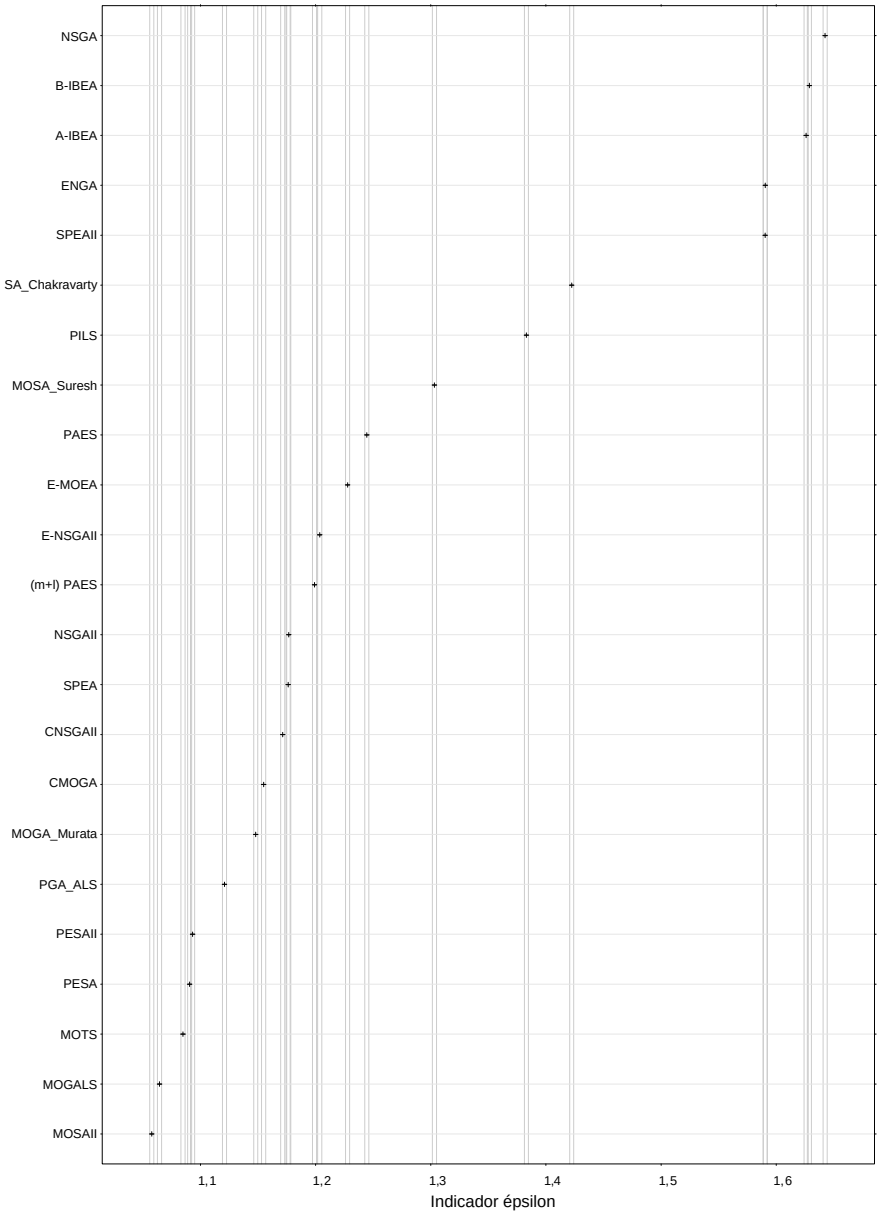


Figura 3.19: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador épsilon y criterio de parada $t = 150$.

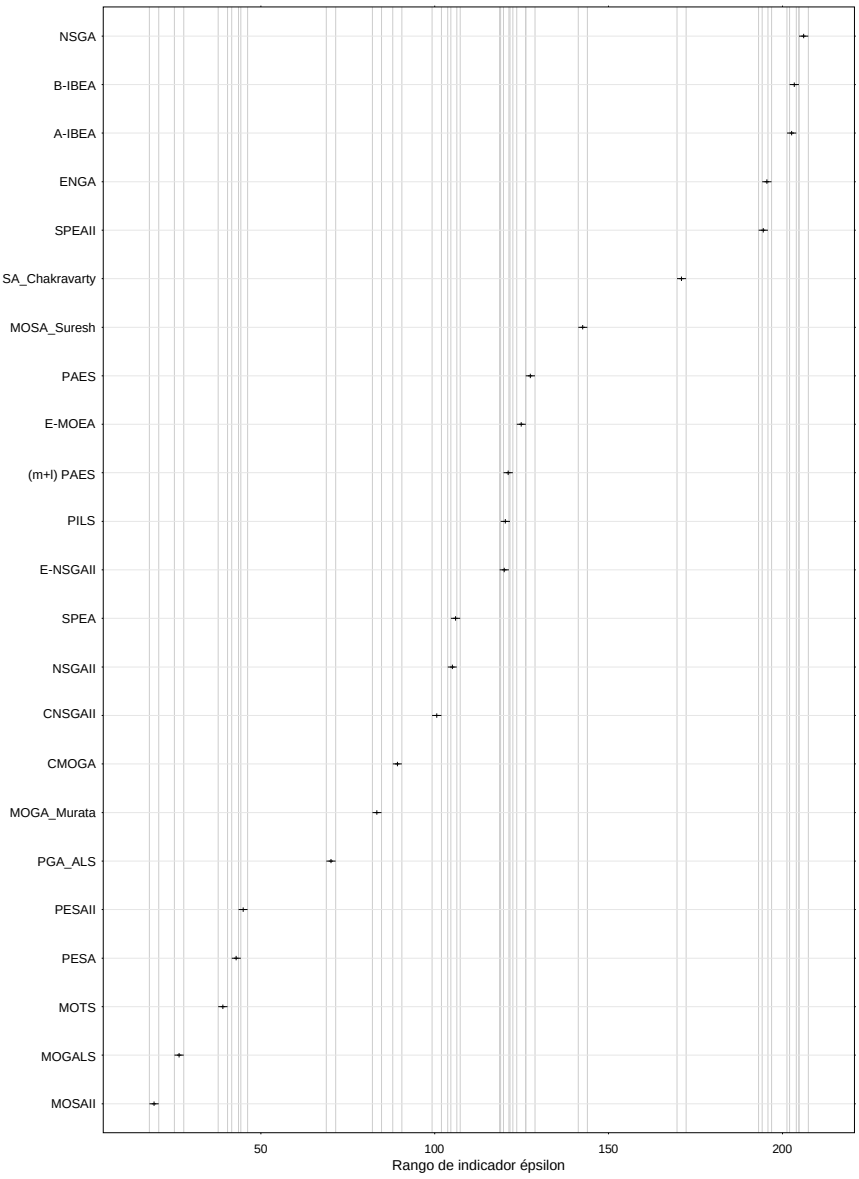


Figura 3.20: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador épsilon y criterio de parada $t = 150$.

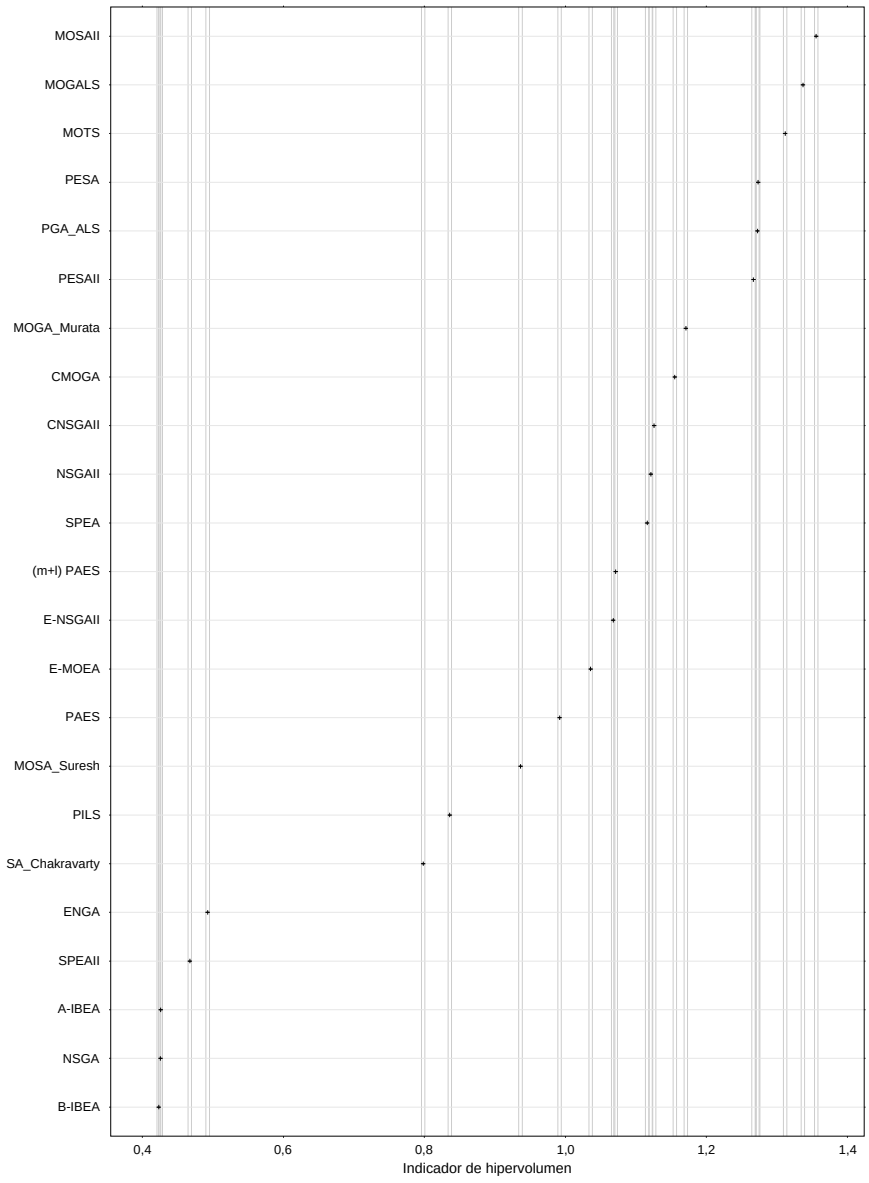


Figura 3.21: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador de hipervolumen y criterio de parada de $t = 150$.

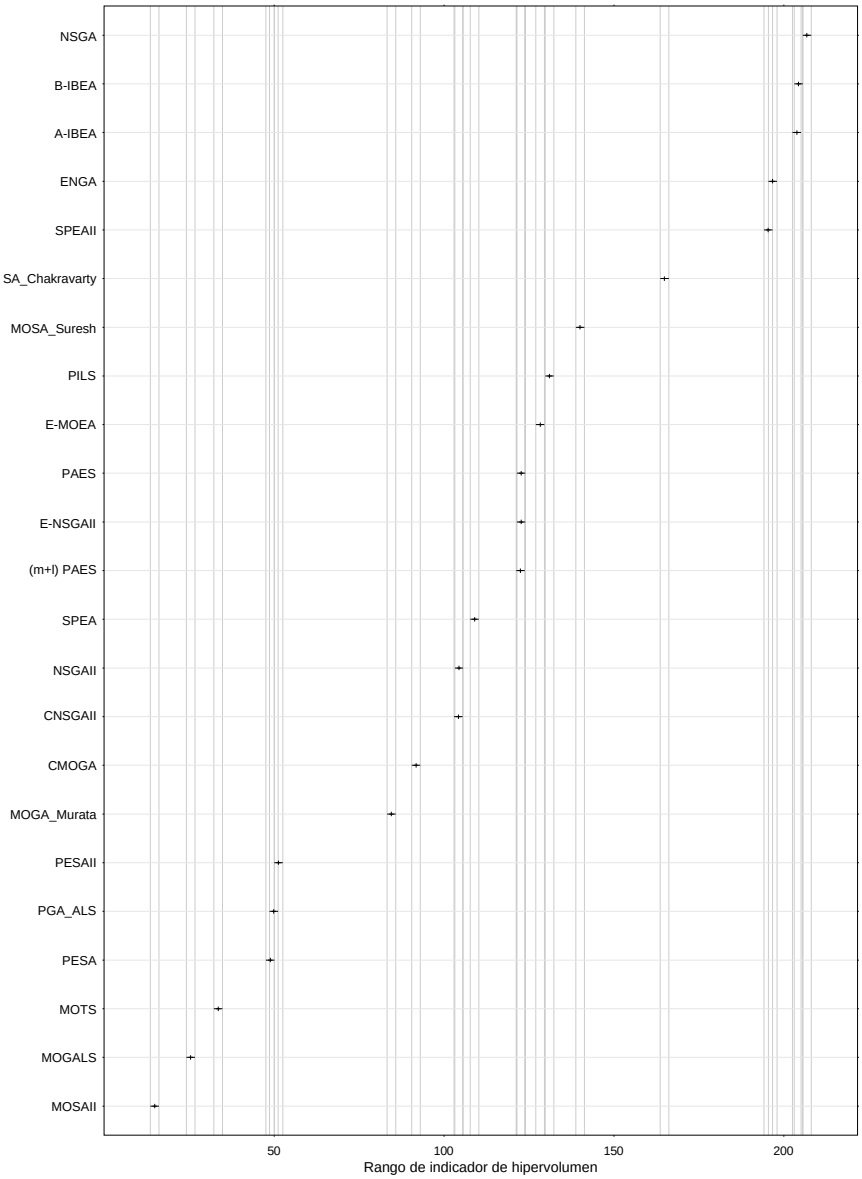


Figura 3.22: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador de hipervolumen y criterio de parada $t = 150$.

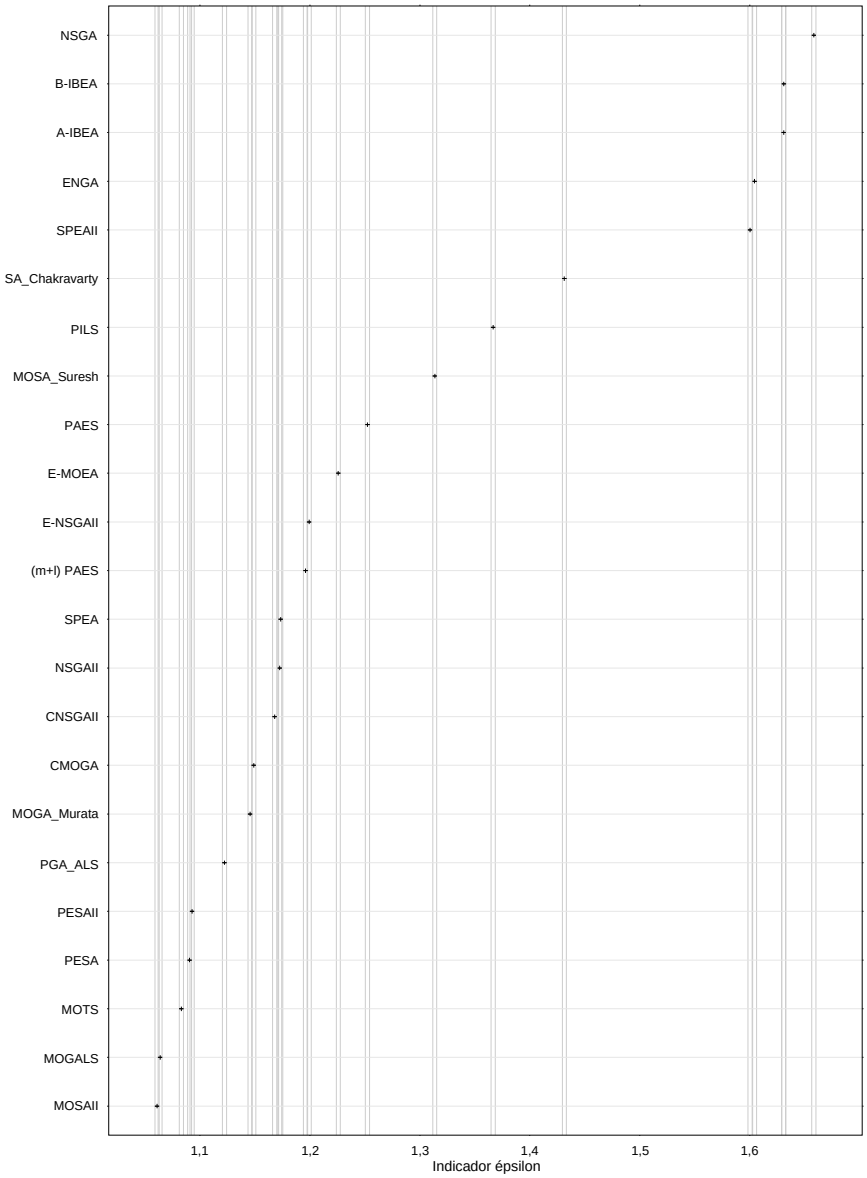


Figura 3.23: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador épsilon y criterio de parada de $t = 200$.

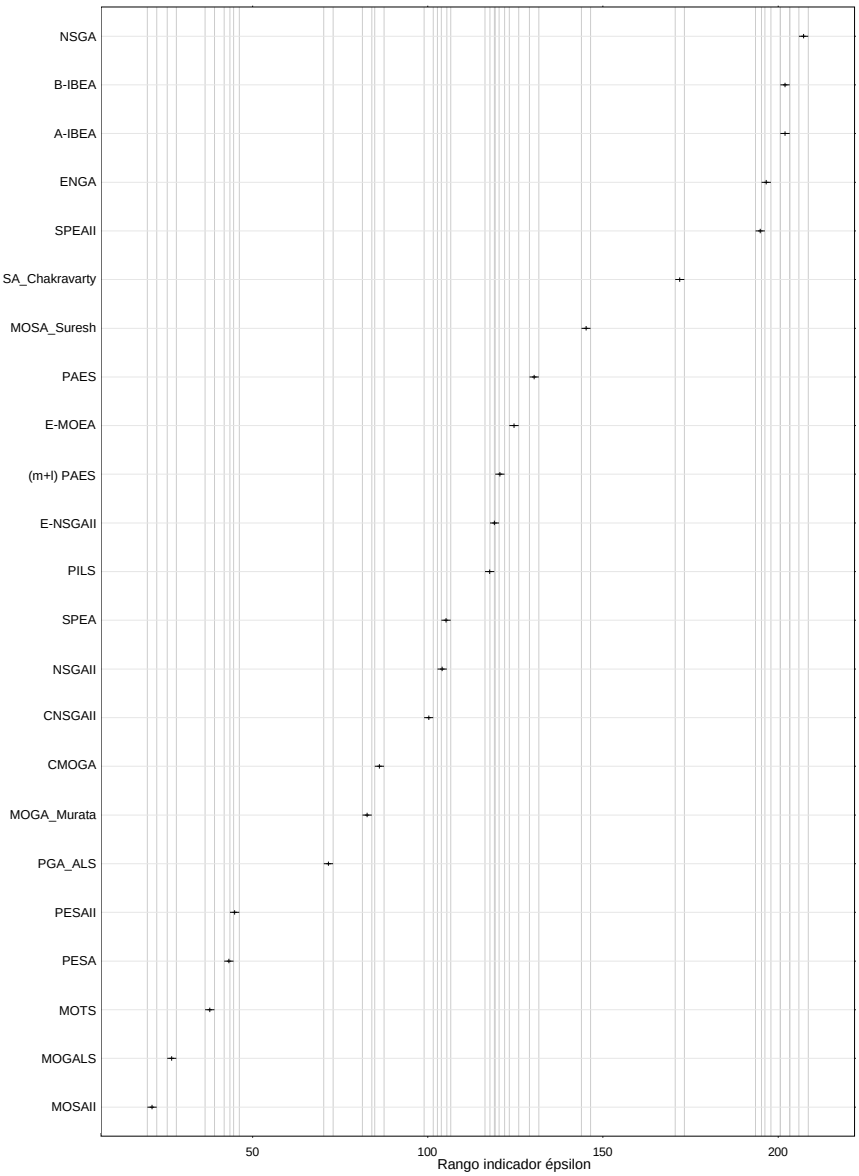


Figura 3.24: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador épsilon y criterio de parada de $t = 200$.

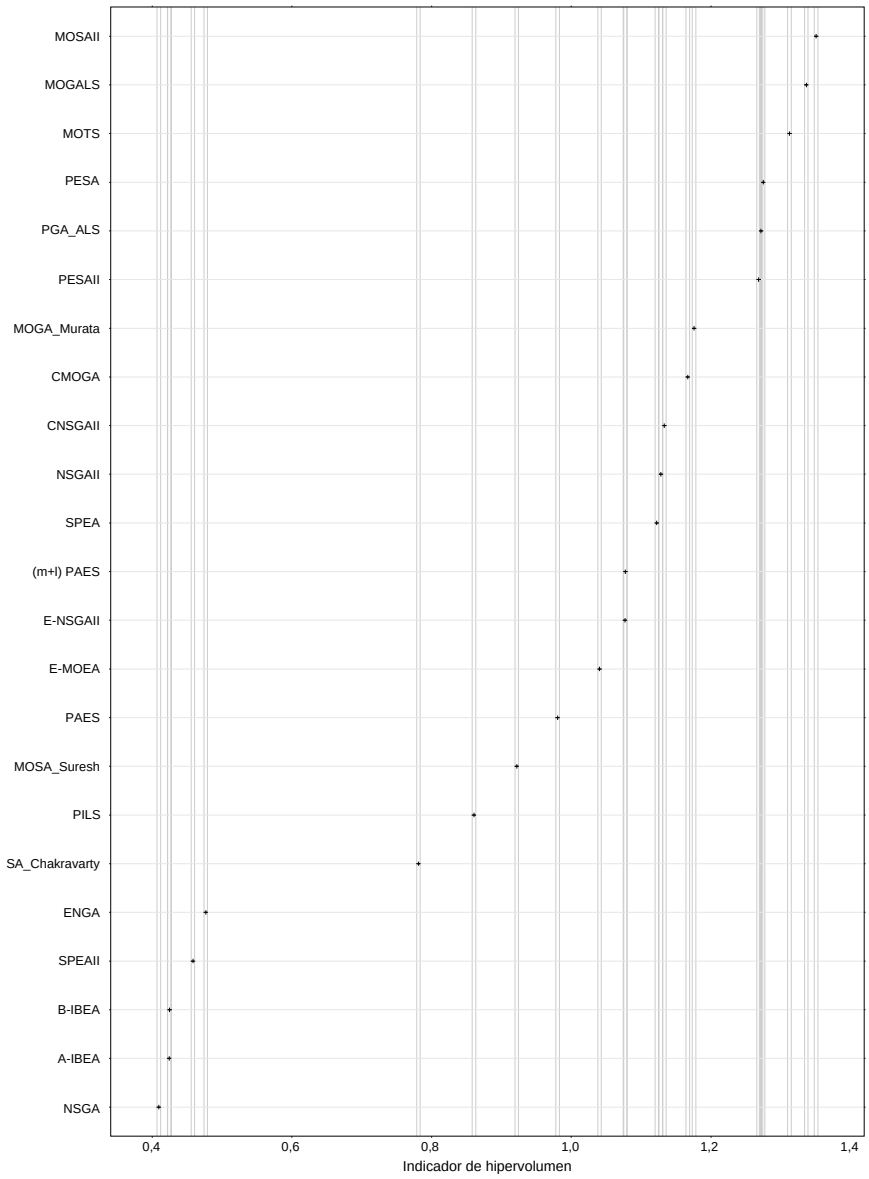


Figura 3.25: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador de hipervolumen y criterio de parada de $t = 200$.

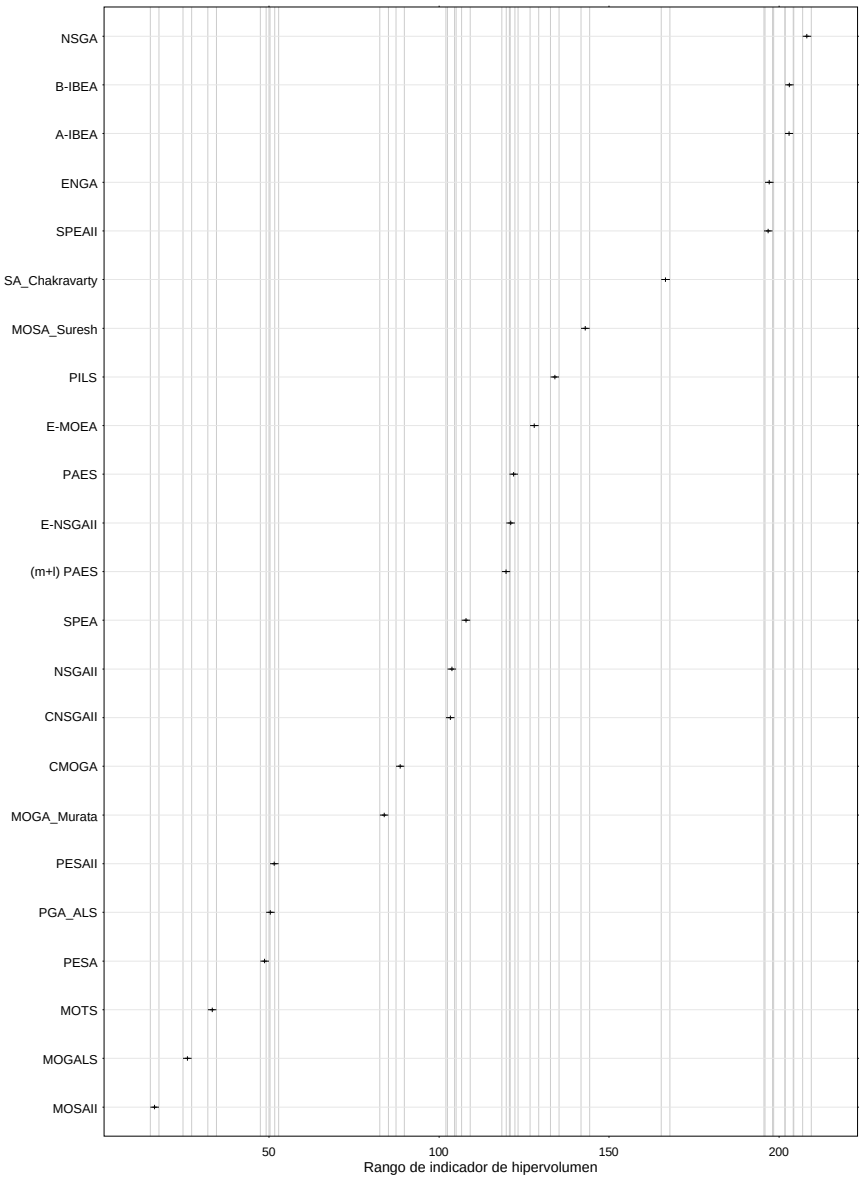


Figura 3.26: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador de hipervolumen y criterio de parada $t = 200$.

Tanto en la Tabla 3.3 como en las gráficas que le siguen puede observarse que los resultados se parecen bastante a los resultados obtenidos con los experimentos para $C_{\max}$ y tardanza total, sobre todo para los métodos que obtienen mejor rendimiento. MOSAII, MOGALS y MOTS obtienen el primer, segundo y tercer puesto, respectivamente, para todos los diferentes valores de t, tanto para el indicador de hipervolumen como para el indicador épsilon. Algo a destacar es el hecho de que MOTS ha ocupado el lugar sexto en los experimentos para $C_{\max}$ y tardanza total. PGA_ALS ocupa el cuarto puesto para el experimento de $t = 100$ pero para los experimentos de $t = 150$ y $t = 200$ baja al quinto puesto quedando PESA en el puesto número 4. Este tipo de comportamiento suele indicar alguna deficiencia de rendimiento que ocasiona una pérdida de performance cuando las instancias se hacen más grandes. El resto de los algoritmos se comportan de manera similar para todos los experimentos, variando uno o dos puestos en algunos casos.

3.3.3. Resultados para tiempo total de flujo y tardanza total

Finalmente, en esta sección comentaremos la tercera parte de los experimentos. En este caso hemos realizados los mismo experimento que en las dos secciones anteriores, pero para la combinación de objetivos de tiempo total de flujo y tardanza total. Hemos ejecutado tres tandas de experimentos, para distintos valores de t. Para cada algoritmo, tiempo de parada t y cada una de las 110 instancias se ejecutaron 10 réplicas, de lo que hemos obtenido un total de 75.900 resultados para esta combinación de objetivos.

	Tiempo	100			150			200	
#	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
1	MOSAII	1,381	1,034	MOSAII	1,377	1,036	MOSAII	1,375	1,036
2	MOGALS	1,330	1,057	MOGALS	1,326	1,060	MOGALS	1,324	1,061
3	PGA_ALS	1,311	1,076	PGA_ALS	1,313	1,075	PGA_ALS	1,315	1,074

Continúa en la siguiente página...

	Tiempo	100			150			200	
#	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
4	MOTS	1,296	1,073	MOTS	1,302	1,071	MOTS	1,309	1,067
5	PESA	1,263	1,088	PESA	1,262	1,088	PESA	1,263	1,088
6	PESAII	1,263	1,088	PESAII	1,261	1,088	PESAII	1,262	1,087
7	$(\mu+\lambda)-$PAES	1,190	1,126	$(\mu+\lambda)-$PAES	1,196	1,123	$(\mu+\lambda)-$PAES	1,200	1,121
8	MOGA_Murata	1,183	1,128	MOGA_Murata	1,191	1,124	MOGA_Murata	1,198	1,120
9	CMOGA	1,163	1,138	CMOGA	1,178	1,130	CMOGA	1,188	1,125
10	NSGAII	1,158	1,143	NSGAII	1,170	1,137	NSGAII	1,176	1,134
11	CNSGAII	1,148	1,146	CNSGAII	1,162	1,139	CNSGAII	1,171	1,135
12	SPEA	1,137	1,151	SPEA	1,149	1,145	SPEA	1,157	1,141
13	$\varepsilon-$NSGAII	1,111	1,165	$\varepsilon-$NSGAII	1,131	1,155	$\varepsilon-$NSGAII	1,144	1,148
14	$\varepsilon-$MOEA	1,100	1,170	$\varepsilon-$MOEA	1,107	1,167	$\varepsilon-$MOEA	1,111	1,164
15	PAES	1,081	1,190	PAES	1,071	1,195	PAES	1,066	1,197
16	MOSA_Suresh	1,065	1,207	MOSA_Suresh	1,053	1,214	MOSA_Suresh	1,048	1,216
17	PILS	0,824	1,388	PILS	0,865	1,357	PILS	0,900	1,334
18	SA_Chakravarty	0,612	1,480	SA_Chakravarty	0,592	1,493	SA_Chakravarty	0,584	1,498
19	SPEAII	0,453	1,572	SPEAII	0,446	1,577	SPEAII	0,445	1,576
20	ENGA	0,426	1,609	ENGA	0,406	1,628	ENGA	0,397	1,635
21	NSGA	0,368	1,658	NSGA	0,348	1,679	$A-$IBEA	0,339	1,670
22	$A-$IBEA	0,333	1,676	$A-$IBEA	0,337	1,674	NSGA	0,338	1,686
23	$B-$IBEA	0,332	1,676	$B-$IBEA	0,337	1,673	$B-$IBEA	0,338	1,671

Tabla 3.4: Resultados para los criterios de tiempo total de flujo y tardanza total. Medias de los indicadores de calidad para los 23 algoritmos probados bajo los tres diferentes criterios de terminación. Cada valor medio se calcula entre 110 instancias y 10 réplicas por instancia (1.100 valores en total). Para cada criterio de terminación, los métodos están ordenados de acuerdo a I_H en forma descendente.

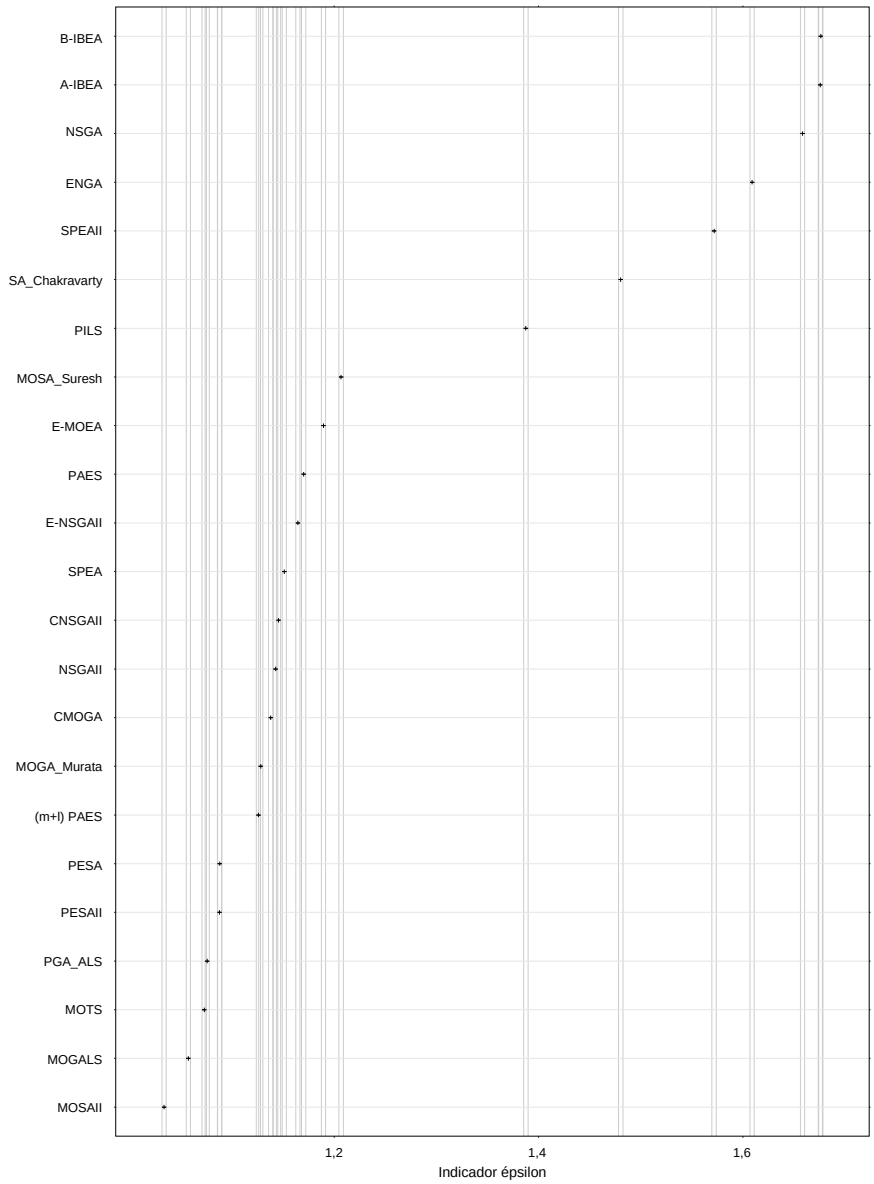


Figura 3.27: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador épsilon y criterio de parada $t = 100$.

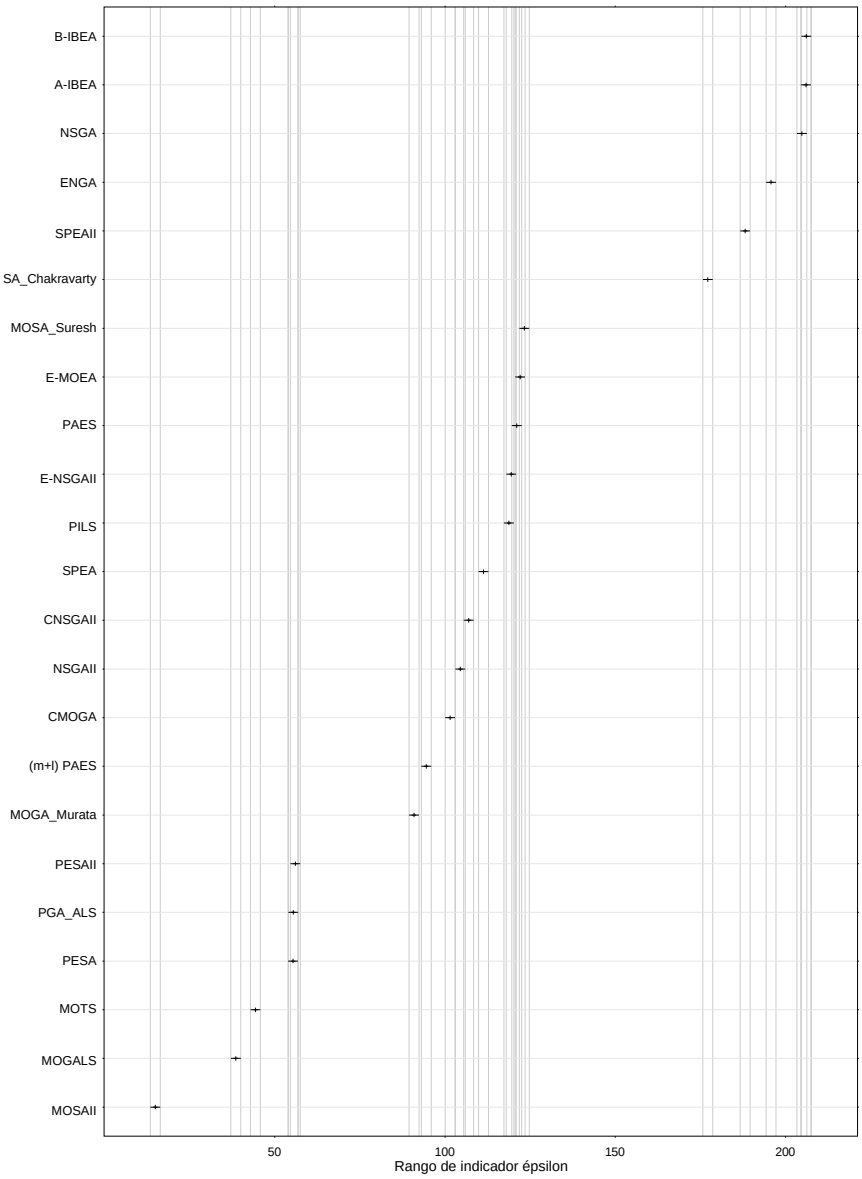


Figura 3.28: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador épsilon y criterio de parada $t = 100$.

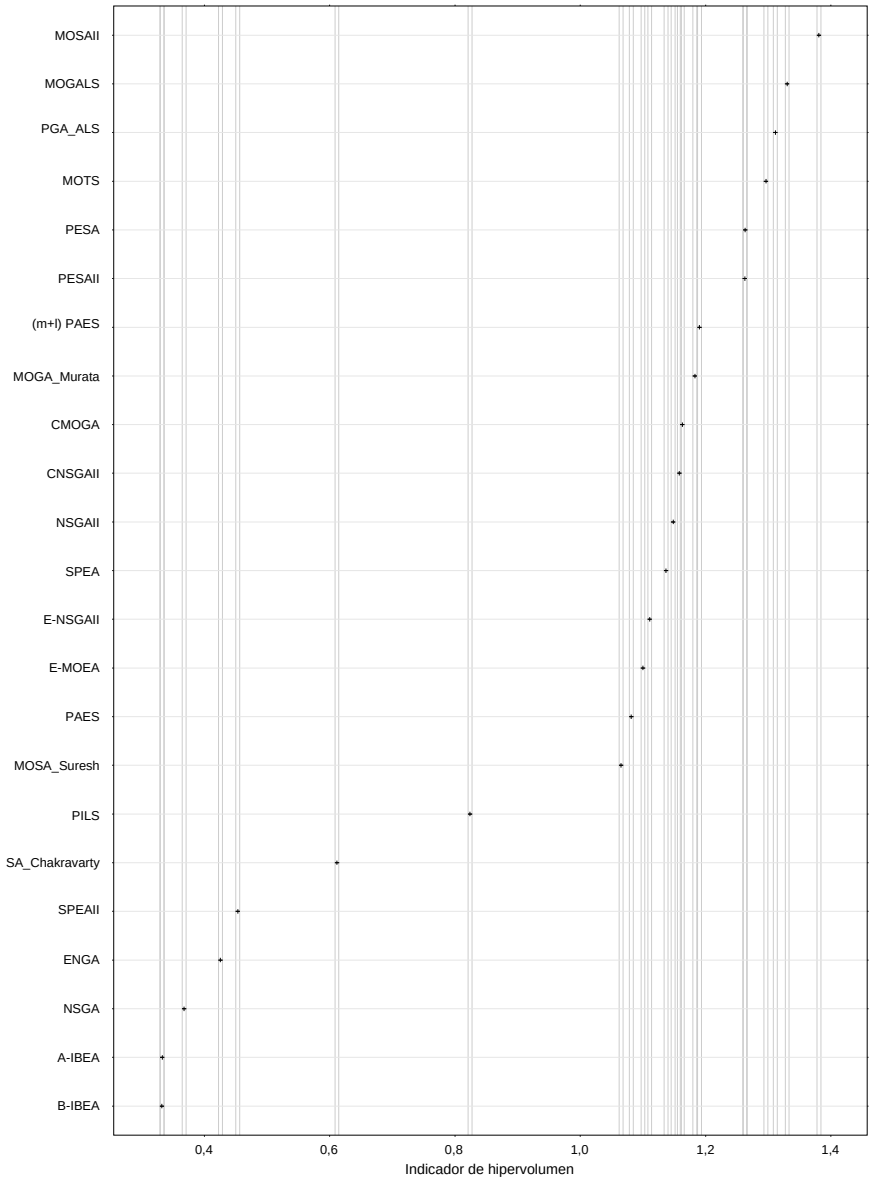


Figura 3.29: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador de hipervolumen y criterio de parada $t = 100$.

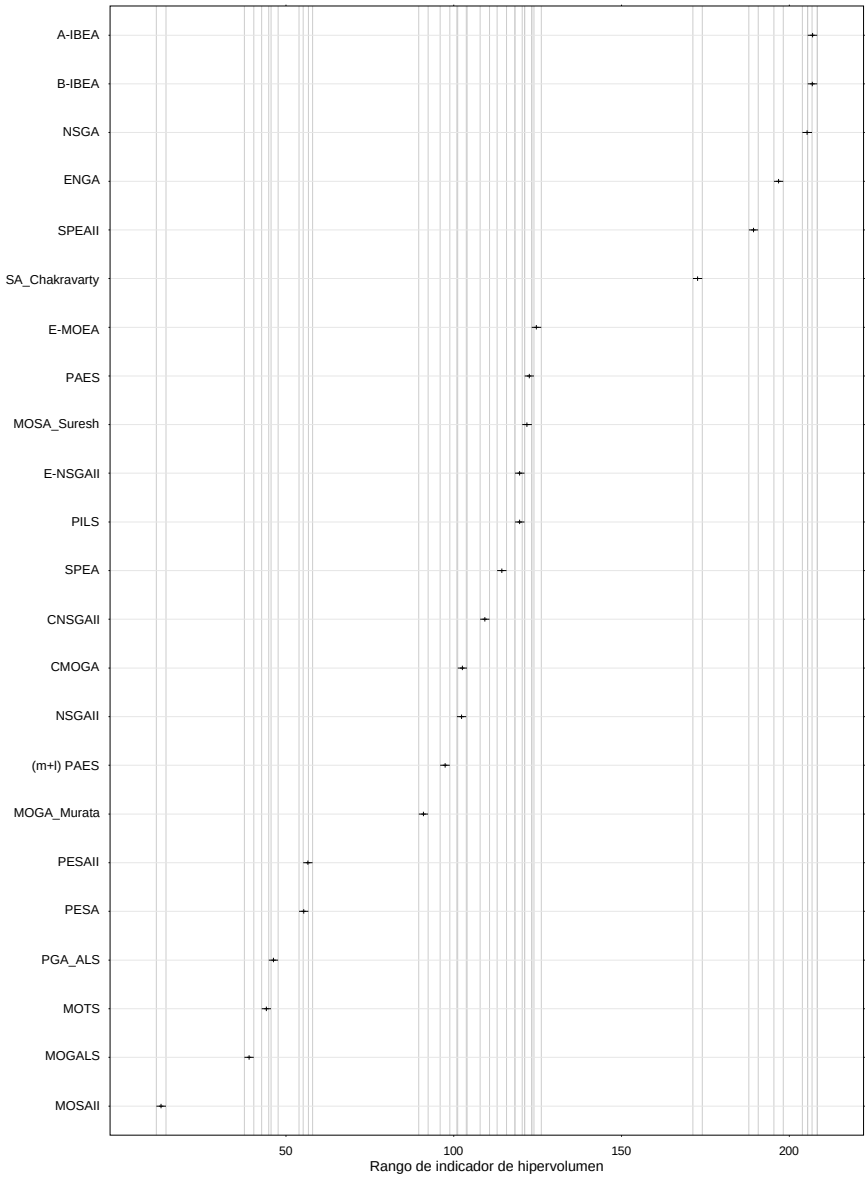


Figura 3.30: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador de hipervolumen y criterio de parada $t = 100$.

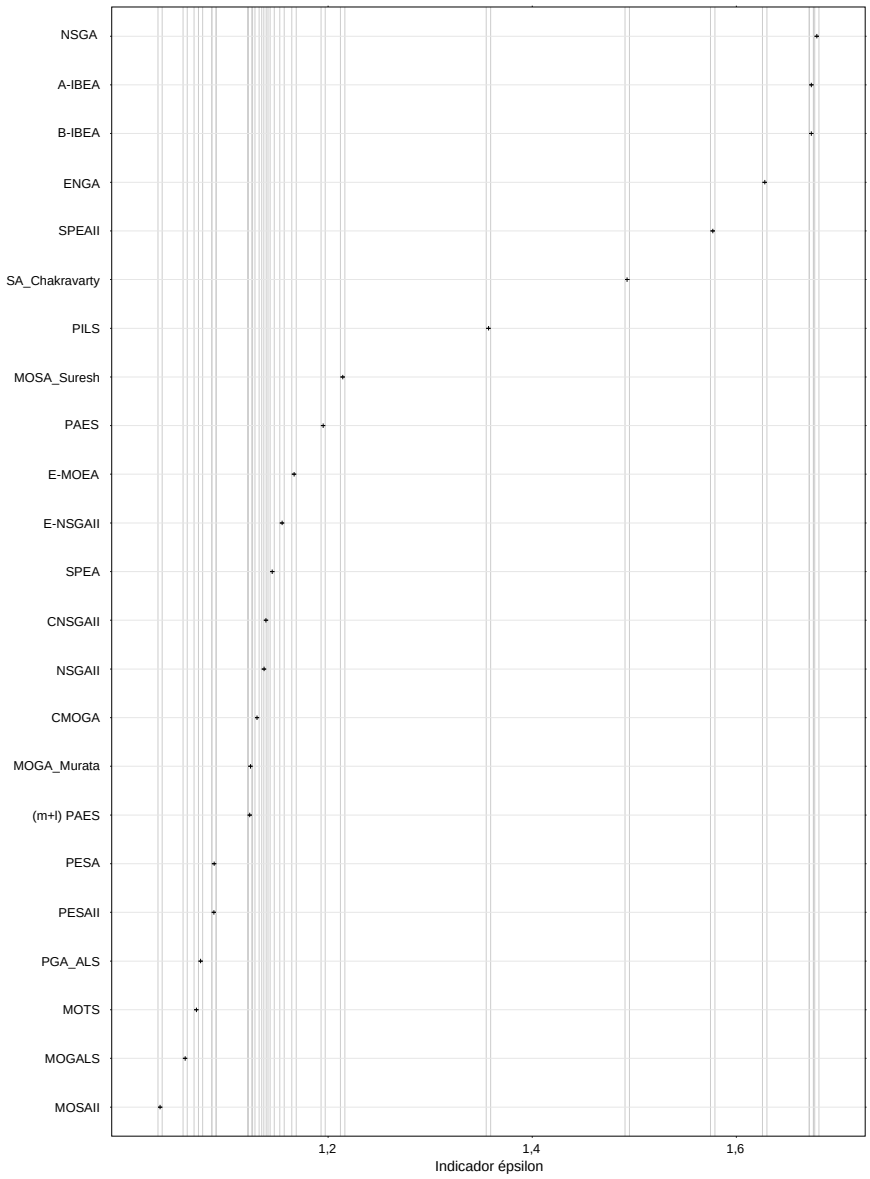


Figura 3.31: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador épsilon y criterio de parada $t = 150$.

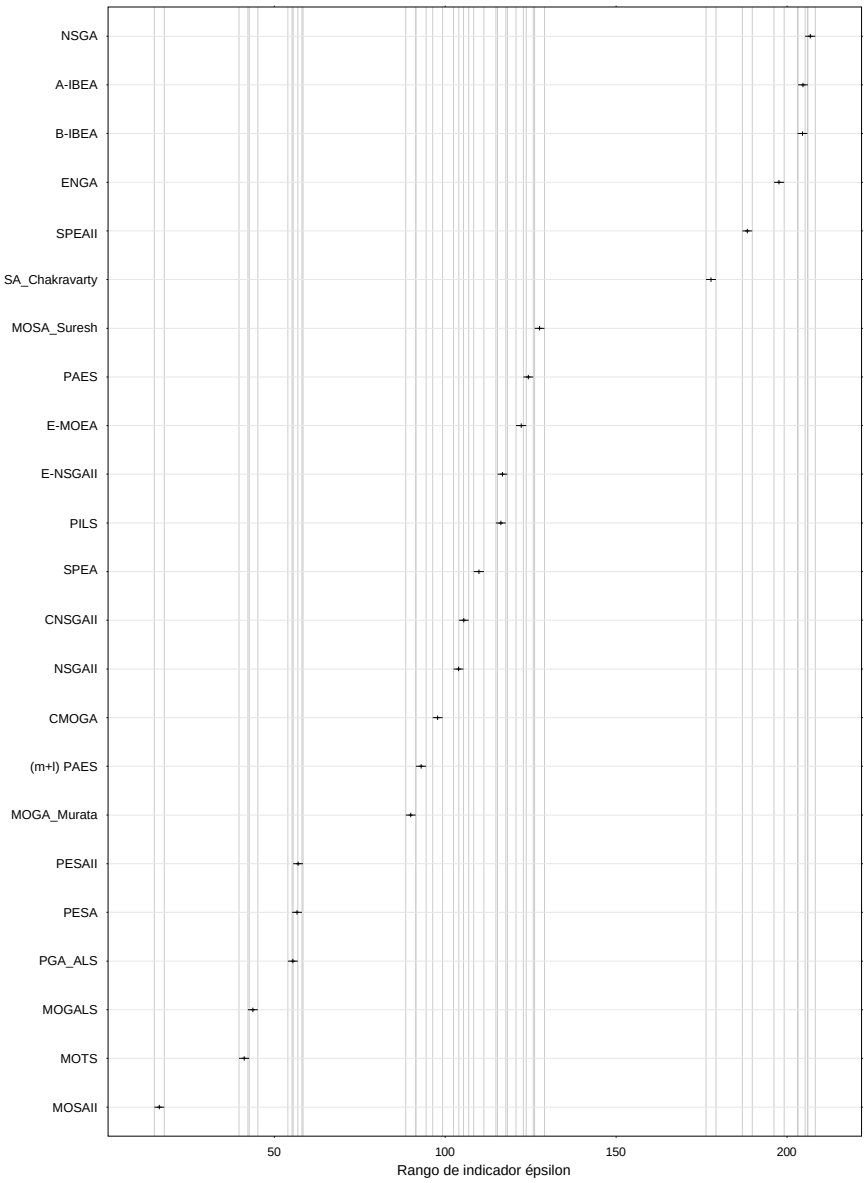


Figura 3.32: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador épsilon y criterio de parada $t = 150$.

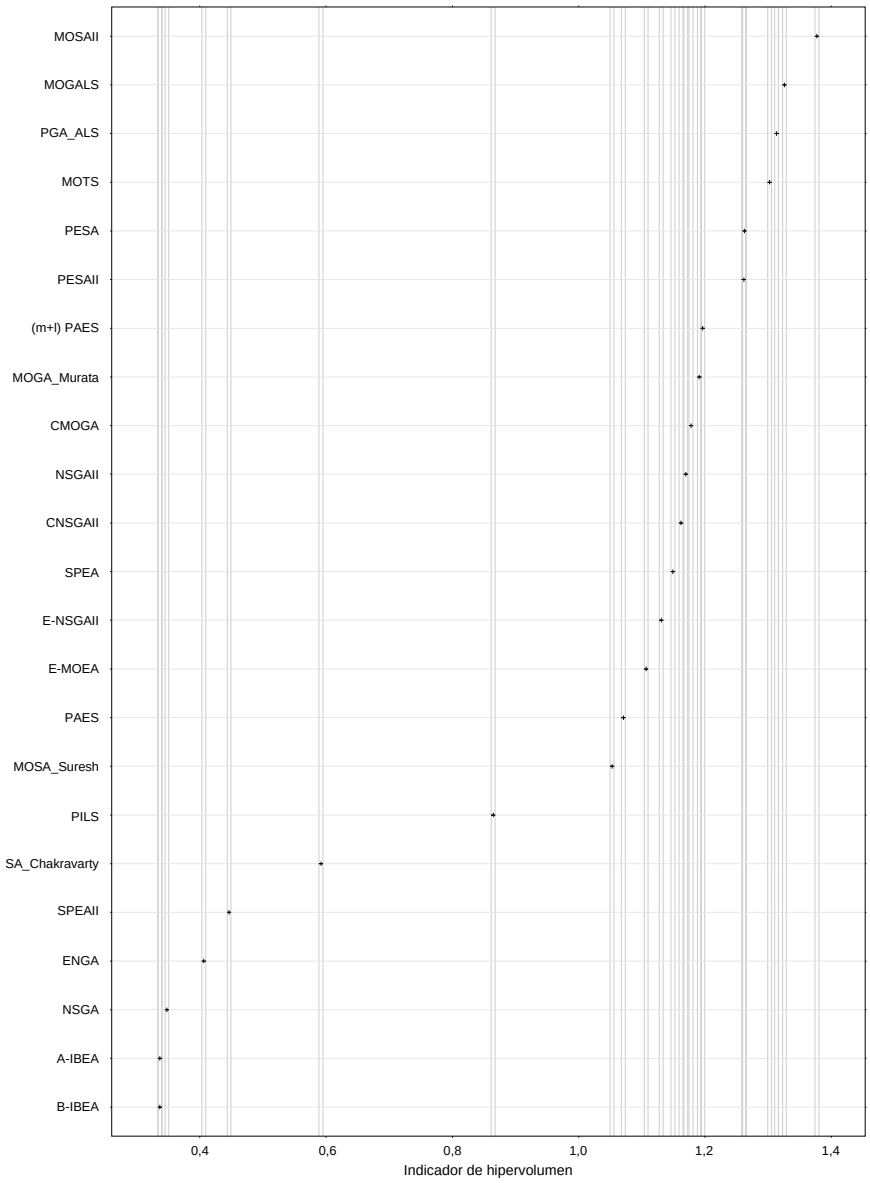


Figura 3.33: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador de hipervolumen y criterio de parada de $t = 150$.

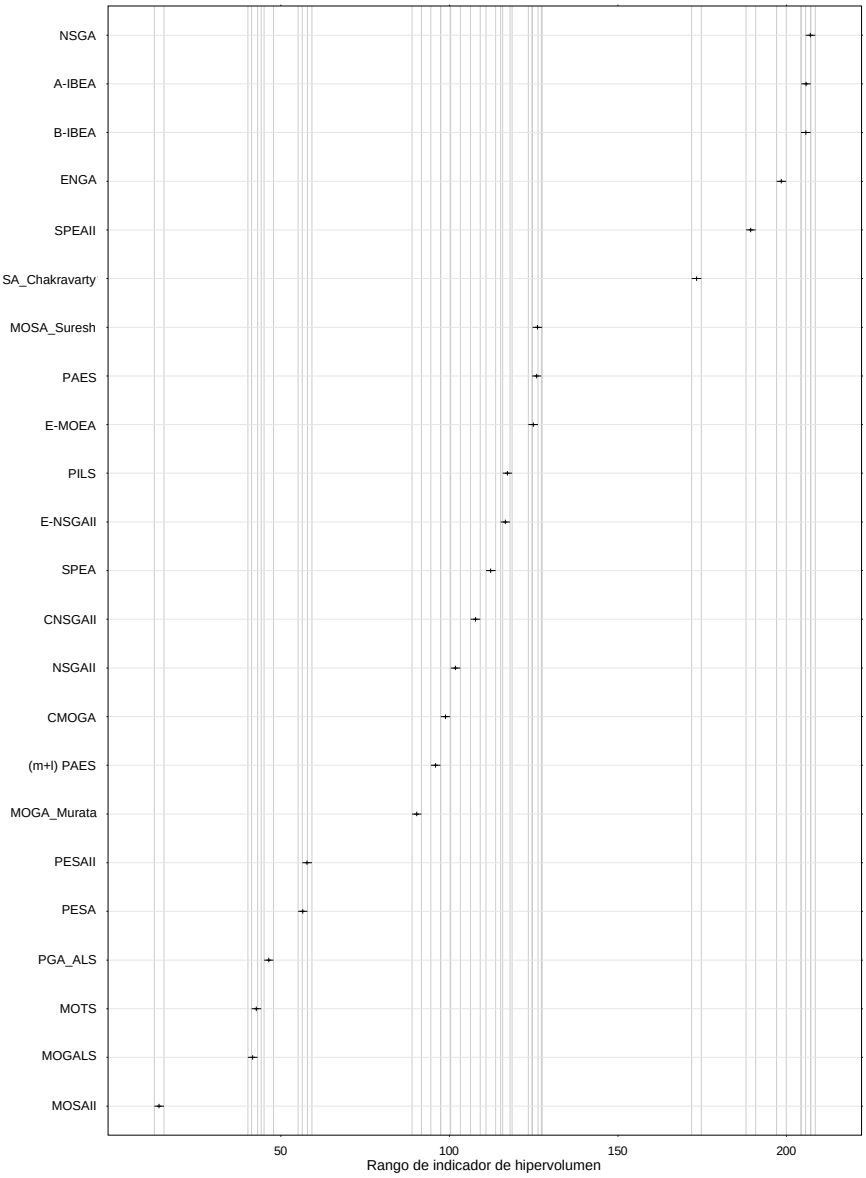


Figura 3.34: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador de hipervolumen y criterio de parada $t = 150$.

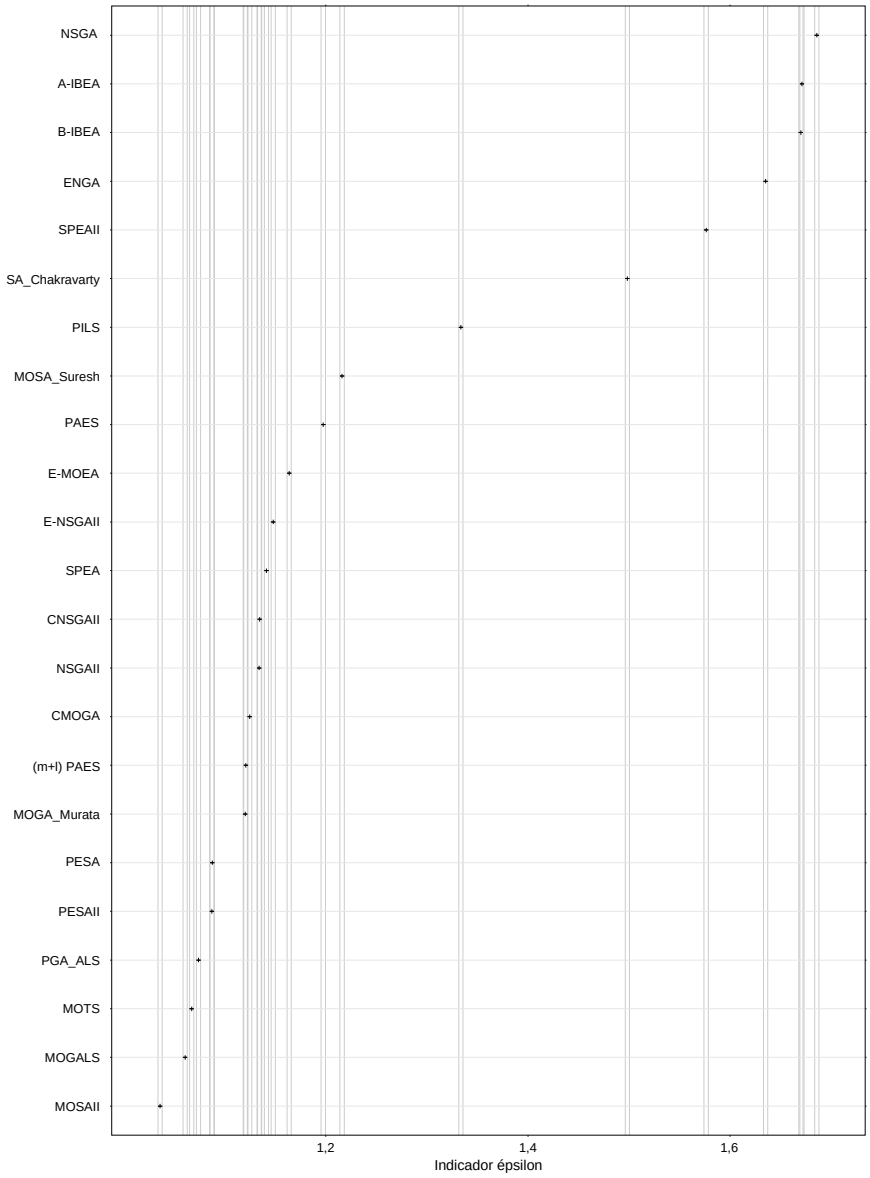


Figura 3.35: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador épsilon y criterio de parada $t = 200$.

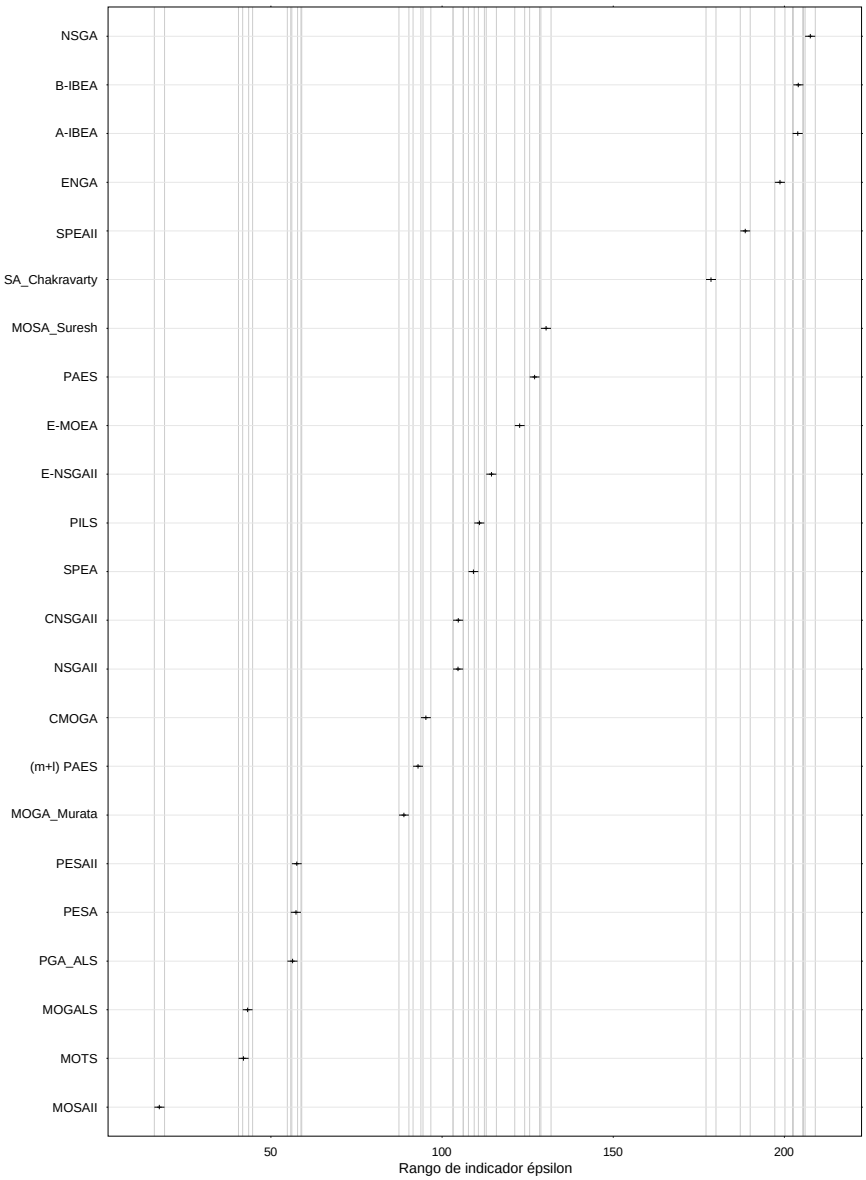


Figura 3.36: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador épsilon y criterio de parada $t = 200$.

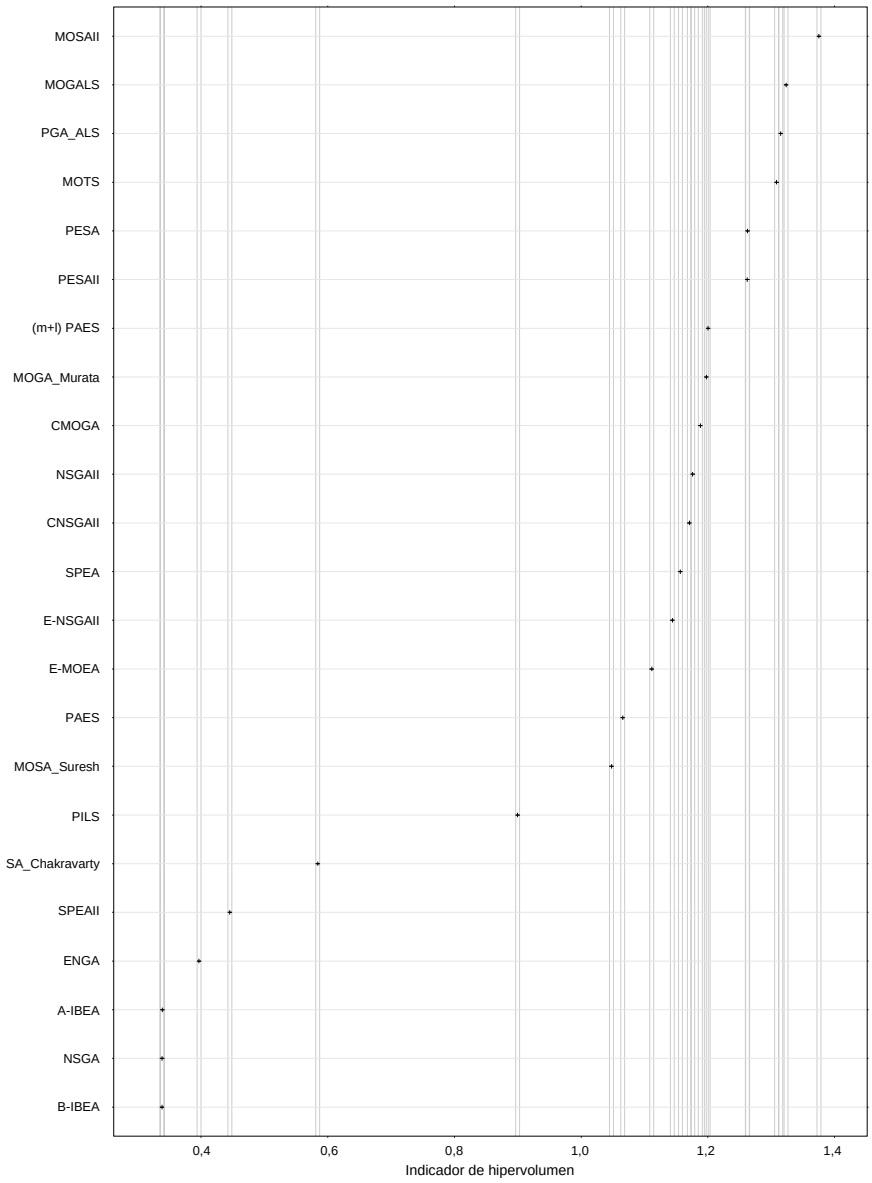


Figura 3.37: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) HSD de Tukey para el factor algoritmo en el experimento ANOVA. Variable respuesta indicador de hipervolumen y criterio de parada $t = 200$.

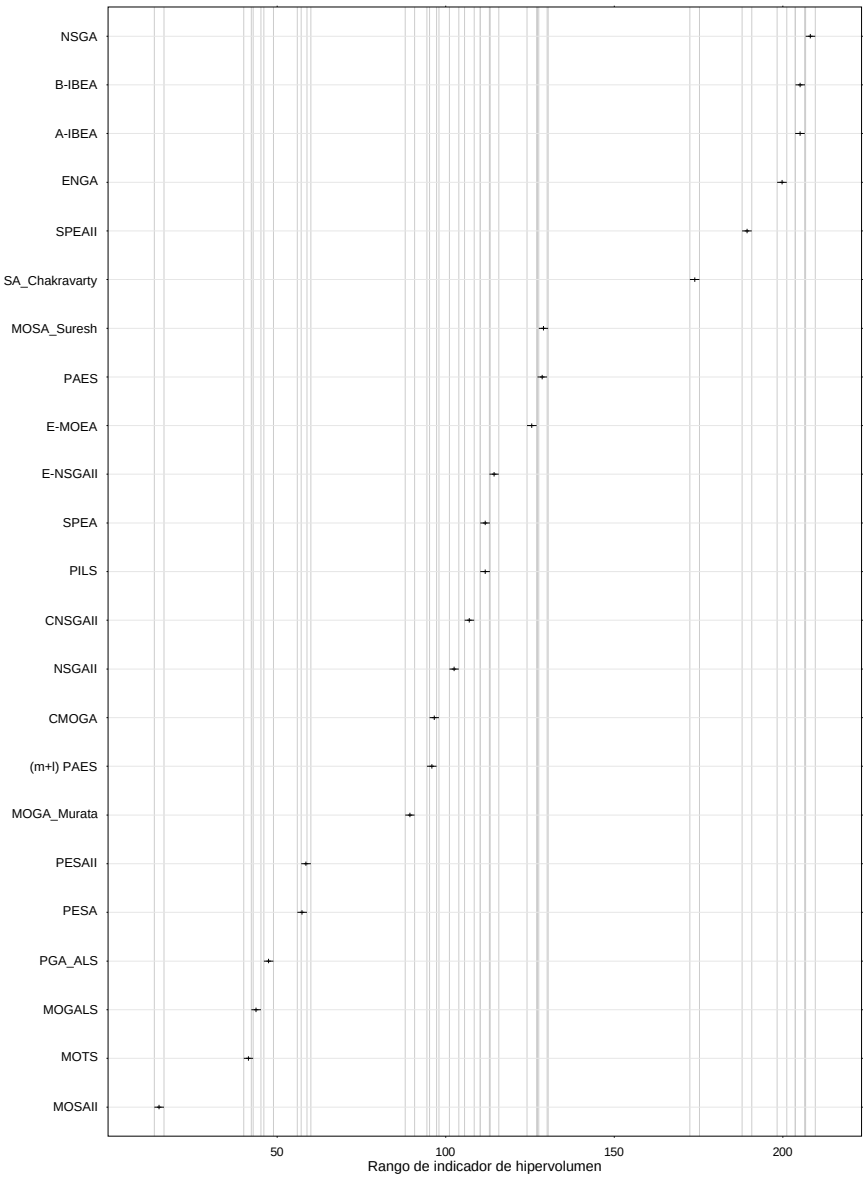


Figura 3.38: Gráfico de medias e intervalos de confianza ($\alpha_s = 0{,}01$, $\alpha = 0{,}05$) MSD para las pruebas de rangos de Friedman. Variable respuesta indicador de hipervolumen y criterio de parada $t = 200$.

En esta sección hemos realizado los mismos experimentos que en las secciones anteriores utilizando otra combinación de objetivos. Partiendo de un total de 75.900 aproximaciones a la frontera de Pareto obtenidas se pueden hacer las siguiente inferencias. Los resultados muestran un comportamiento similar a los experimentos anteriores en los algoritmos, con pequeñas diferencias. El primer y segundo lugar siguen siendo ocupados por los algoritmos MOSAII y MOGALS respectivamente. Algo a destacar es el hecho de que todos los algoritmos excepto NSGA y A-IBEA, en las posiciones 21 y 22 conservan su posición a través de los distintos valores de t. Mientras que el algoritmo B-IBEA es, de manera consistente, el algoritmo que obtiene los peores resultados en todas las pruebas.

3.3.4. Conclusiones de este capítulo

En este capítulo hemos presentado los resultados de los experimentos realizados con los 23 algoritmos implementados. Hemos utilizado una batería de pruebas compuesta de 110 instancias basadas en las instancias de Taillard. Estas 110 instancias pueden a su vez dividirse en 11 grupos de diez instancias cada uno. Hemos realizado las pruebas utilizando 3 combinaciones diferentes de objetivos. Para cada combinación de objetivos hemos realizado pruebas con 3 diferentes valores de t. En total, de todos estos experimentos hemos obtenido y revisado un total de 227.700 resultados.

Para cada combinación de objetivos hemos analizado el comportamiento de los 23 algoritmos implementados de una forma estadística. En general el comportamiento ha sido muy similar, obteniendo consistentemente el mejor desempeño el algoritmo MOSAII y en segundo lugar el algoritmo MOGALS. Los algoritmos MOTS, PGA_ALS, PESA y PESAII ocuparon las posiciones entre la tercera y la sexta, cambiando de posición entre los distintos experimentos. No obstante en la mayoría de los experimentos el desempeño de MOTS ha sido muy bueno, ocupando la tercera posición. Algo destacable de este algoritmo es que es capaz de obtener muy buenos resultados para instancias pequeñas,

pero su desempeño merma cuanto más grande es la instancia.
Claramente, se puede observar que los algoritmos que mantienen estructuras complejas o búsquedas locales exhaustivas tienen también una tendencia a tener un rendimiento pobre con instancias más grandes. Por otro lado, los algoritmos más simples obtienen un rendimiento más estable. El desafío, entonces, consiste en encontrar un algoritmo veloz, simple y con estructuras y búsquedas livianas y al mismo tiempo sea capaz de encontrar buenas soluciones. En los siguientes capítulos vamos a proponer, describir, desarrollar y probar un nuevo algoritmo. Realizaremos pruebas de rendimiento comparando el algoritmo propuesto contra los algoritmos que han obtenido mejores resultados en este capítulo de revisión.

CAPÍTULO 4

UN NUEVO MÉTODO ITERATIVO

En el Capítulo 3, luego de comprar 23 algoritmos, hemos determinado que los que obtienen los mejores resultados para la batería de pruebas propuesta son: en primer lugar el algoritmo de recocido simulado propuesto en Varadharajan y Rajendran (2005) denominado MOSAII, en segundo lugar se encuentra el algoritmo MOGALS. Éste es un algoritmo genético que hace uso de una búsqueda local intensiva, propuesto por Arroyo y Armentano (2005). Finalmente el tercer algoritmo en obtener los mejores resultados, ha resultado ser una búsqueda tabú denominada MOTS, propuesta por Armentano y Arroyo (2004). Algo interesante a destacar es la heterogeneidad de los algoritmos que obtienen los mejores resultados, ya que cada uno de ellos representa una metodología completamente diferente del otro.

Mientras llevábamos a cabo el estudio del Capítulo 3, nuevos algoritmos han ido apareciendo en la literatura. Por esta razón, al proponer nuestro propio algoritmo, incluiremos algunos de estos nuevos métodos para que la comparación se haga contra el estado del arte en cuanto a la resolución de problemas de flowshop multi-objetivo. Estos algoritmos se describen a continuación.

Yandra y Tamura (2007) proponen un algoritmo genético híbrido denominado hMGA, que trabaja con poblaciones heterogéneas. Básicamente hace una pequeña modificación sobre el algoritmo NSGAII de Deb (2002), pero deja de lado varios problemas bien conocidos de esta metodología. En consecuencia obtiene un algoritmo capaz de superar en resultados al NSGAII, pero que se queda muy por detrás de MOSAII.

Finalmente Framiñan y Leisten (2008) realizan, por primera vez, la adaptación de un algoritmo Iterated Greedy a problemas de flowshop multi-objetivo. Este algoritmo presenta algunas diferencias notorias respecto al algoritmo original presentado en Ruiz y Stützle (2007) mayormente debido a que en vez de trabajar con una única solución trabaja con una población de soluciones no dominadas. En este trabajo se realizan varias pruebas y se compara el algoritmo propuesto contra el MOSAII (original). Más adelante, compararemos este algoritmo con el algoritmo MOSAII$_M$, que es una versión adaptada y modificada de manera que pueda detenerse por tiempo. También ampliaremos la comparación a instancias grandes, con la finalidad de corroborar los resultados del algoritmo.

En la siguiente Sección haremos una breve introducción al algoritmo *Iterated Greedy* (IG). Luego de lo cual propondremos un nuevo algoritmo multi-objetivo, basado en el IG y que denominamos: *Restarted Iterated Pareto Greedy* o RIPG. El desarrollo del algoritmo será progresivo, comenzaremos por un algoritmo básico llamado *(Basic) Iterated Pareto Greedy* o IPG, y haremos varias pruebas y comparaciones, como así también mejoras. Se destacará en todo momento la importancia y relevancia del correcto diseño de un algoritmo, así como la importancia que esto puede tener en los resultados obtenidos.

4.1. El algoritmo *Restarted Iterated Pareto Greedy*

Antes de presentar nuestro algoritmo comenzaremos por la descripción del algoritmo voraz iterado, *iterated greedy* o simplemente IG, propuesto por Ruiz y Stützle (2007) para resolver problemas de flowshop de permutación con un solo objetivo.
El algoritmo voraz iterado básicamente consiste en, iterativamente, destruir varios elementos de una solución, para luego reconstruir una nueva solución mediante una técnica constructiva voraz. Esta nueva solución puede ser luego mejorada mediante una búsqueda local.
Dentro de este algoritmo hay dos partes o etapas perfectamente identificables: la etapa de destrucción-reconstrucción voraz y la etapa de búsqueda local. Durante la primera parte, algunos elementos son extraídos o removidos de la solución actual. Luego de ello, esos elementos vuelven a ser re-insertados de manera que se obtenga una nueva y posiblemente mejor solución completa.
El procedimiento de re-inserción de elementos está basado en la conocida heurística NEH de Nawaz, Enscore y Ham (1983). Esta heurística conforma un método voraz, donde básicamente se prueba insertar, uno a uno, los elementos extraídos en todas las posiciones posibles de la solución parcial actual. Cada elemento es colocado en la posición con la que se obtiene el mejor resultado en el valor de la función objetivo.

Desde hace algún tiempo han comenzado a aparecer varios estudios relativos a la aplicación de algoritmos IG a distintos tipos de problemas. Por ejemplo, Ruiz y Stützle (2007) han extendido los estudios sobre algoritmos IG a otros objetivos y a problemas de flowshop con tiempos de setup dependientes de la secuencia (SDST). Los autores Pan, Wang y Zhao (2008) desarrollan una adaptación del algoritmo IG al problema del taller de flujo sin esperas (*no wait*). También Framiñan y Leisten (2008) han adaptado el algoritmo IG al problema del taller de flujo multi-objetivo. Por otro lado Ying (2008) ha

aplicado algoritmos IG a problemas de flowshop híbridos con tareas multi-procesables. En otro artículo Toyama, Shoji y Miyamichi (2008) han aplicado un algoritmo IG a la ubicación de nodos dentro de redes de calles. Mientras que Zhi et al. (2008) lo han utilizado en la resolución de problemas de programación de horarios de trenes. Luego, Fanjul-Peyro y Ruiz (2010) aplican el IG al problema de las máquinas paralelas. Finalmente Ribas, Companys y Tort-Martorell (2011) han adaptado también el algoritmo IG al problema del taller de flujo con bloqueo.

En las siguientes secciones expondremos nuestras adaptaciones del algoritmo IG original para los problemas de flowshop de permutación multi-objetivo. En primer lugar y teniendo en cuenta que el algoritmo original fue pensado para trabajar con una única solución y que en problemas multi-objetivo Pareto es necesario trabajar con conjuntos de soluciones, muchas modificaciones y adaptaciones son necesarias. Por ejemplo, para trabajar con poblaciones también es recomendable utilizar métodos de selección que tengan en cuenta la distribución de soluciones dentro de la frontera de Pareto. Con esto se pretende dirigir la búsqueda de soluciones hacia ciertos valores, evitar los atascos en óptimos locales y obtener fronteras con valores más y mejor distribuidos.

4.1.1. El algoritmo *Iterated Pareto Greedy*

La contribución más destacable en este algoritmo se refiere al manejo de un conjunto de soluciones no dominadas a manera de conjunto de soluciones de trabajo (en adelante CST), en vez de utilizar una única solución como hace el algoritmo IG original. Para que esto sea posible, es necesario definir un método de selección que sea eficaz a la hora de dirigir la búsqueda. Teniendo en cuenta que el resultado de un algoritmo multi-objetivo es un conjunto de soluciones no dominadas, es muy importante que el mecanismo de selección sea eficaz en: dirigir la búsqueda de manera que los huecos en la frontera queden cubiertos y al mismo tiempo evitar que el algoritmo se quede atascado en “óptimos

locales". Para ello hemos desarrollado un mecanismo de selección que intenta dirigir la búsqueda hacia una frontera de Pareto bien distribuida, esto es, que las soluciones no estén ni demasiado cerca, ni demasiado lejos unas de otras. En cada paso una solución es elegida y procesada por la etapa voraz, en la que algunos elementos se extraen, como en el IG original. El procedimiento de reconstrucción de la solución en este caso cambia radicalmente. En el IG original en cada iteración se obtiene una nueva solución. En el algoritmo IPG, a cada iteración, se obtiene un *conjunto de soluciones no dominadas*. El conjunto de soluciones de trabajo es actualizado añadiendo las nuevas soluciones generadas en el paso de reconstrucción de la etapa voraz.
El conjunto de soluciones de trabajo o CST se actualiza luego de la etapa voraz y posiblemente se añaden nuevas soluciones. Esto significa que el valor de *crowding*, que se utiliza en el algoritmo de selección, de cada solución debe ser re-calculado. Además de ello, también existe la posibilidad de que la última solución elegida por el algoritmo de selección ya no se encuentre en el conjunto de soluciones de trabajo. Por estas razones, el algoritmo de selección debe ser aplicado una vez más para elegir una solución que será procesada por la etapa de búsqueda local.

La etapa de búsqueda local también ha variado radicalmente respecto a la búsqueda local del algoritmo original. El cambio más obvio es el manejo de un conjunto de soluciones en vez de una única solución. En este aspecto, la búsqueda local genera un conjunto de soluciones a partir de una única solución elegida por el algoritmo de selección. Luego se actualiza el conjunto de soluciones de trabajo a partir de las soluciones no dominadas generadas en la búsqueda local. La etapa voraz y la etapa de búsqueda local se aplican iterativamente hasta que se alcanza el criterio de parada. En conjunto estas etapas forman la versión básica del algoritmo IPG (*Iterated Pareto Greedy*).

El algoritmo IPG básico, como veremos más adelante, tiene un punto débil:

para problemas (o instancias) grandes, de más de 100 trabajos y 10 máquinas, este algoritmo es capaz de obtener resultados muy buenos, superando al resto de los algoritmos revisados en el Capítulo 2. La situación al procesar instancias pequeñas cambia y el algoritmo tiende a converger de manera prematura y estancarse en fronteras que podrían denominarse óptimos locales. Para solucionar el problema de la convergencia hemos extendido el algoritmo IPG básico añadiéndole un operador de reinicio. En la Sección 4.2 se demuestra, mediante experimentación y comparación con resultados anteriores, que la adición de esta nueva etapa incrementa la calidad de las soluciones para instancias o problemas pequeños y de tamaño medio, mientras que no afecta la calidad de soluciones para problemas de gran tamaño.

Otro punto a destacar es la importancia del conjunto inicial de soluciones. En este aspecto se debe prestar especial atención, debido a que un buen conjunto inicial de soluciones es preferible, pero en caso de algoritmos multiobjetivo también es importante que este conjunto de soluciones sea lo más diverso posible para evitar que el resultado esté muy orientado hacia alguno de los objetivos estudiados.
Luego de añadir la etapa de reinicio, el algoritmo propuesto se llamará RIPG o Restarted Iterated Pareto Greedy. En las siguientes secciones detallaremos cada una de las etapas que componen este algoritmo.

4.1.2. Inicialización y operador de selección

Experimentos iniciales mostraron que un buen conjunto de soluciones iniciales mejoran en gran medida los resultados obtenidos por el algoritmo RIPG. Esta es una situación lógica y esperable ya que lo mismo ocurre para el caso del flowshop de permutación con un único objetivo. Para este tipo de problemas, la mayoría de los algoritmos propuestos en la literatura recurren a la utilización de la conocida heurística NEH, mayormente debido a que es extremadamente veloz y a la vez obtiene muy buenos resultados para ciertos

objetivos muy usados como $C_{\max}$. En la versión multi-objetivo de este problema, donde al menos dos objetivos deben considerarse, lo ideal es tener para cada objetivo por lo menos una solución que proporcione un buen valor. Para ello, hemos seleccionado dos heurísticas bastante reconocidas, como son, la antes mencionada NEH de Nawaz, Enscore y Ham (1983) que provee buenas soluciones para $C_{\max}$ y flowtime, y en segundo lugar la heurística propuesta en Rajendran y Ziegler (1997), que da buenos resultados para el criterio de tardanza total.
Ambas heurísticas se aplican a cada uno de los objetivos obteniendo $M \times 2$ soluciones iniciales donde M es la cantidad de objetivos estudiados. En el caso de dos objetivos se obtendrán entonces cuatro soluciones iniciales. Igualmente, no existe garantía alguna que las soluciones obtenidas para el conjunto inicial de soluciones conformen una frontera bien distribuida.

La primer etapa del algoritmo, llamada etapa voraz, es capaz de mejorar enormemente las soluciones iniciales en los primeros pasos del algoritmo. A pesar que esto parece algo positivo, en la etapa inicial no es así. Al seleccionar una de las soluciones iniciales puede ocurrir que la etapa voraz sea capaz de generar soluciones que dominen a las demás soluciones iniciales. En estos casos el resultado es la pérdida de diversidad y cobertura en el CST, justo lo contrario que estábamos buscando al aplicar varias heurísticas para cada objetivo. Si todas las soluciones iniciales son eliminadas en la primera iteración del algoritmo, es muy probable que se descarten direcciones de búsqueda prometedoras. Para solucionar este problema, en el primer paso del algoritmo, todas las soluciones iniciales son procesadas por la etapa voraz una por una, sin utilizar el operador de selección, y por cada solución inicial se obtiene un conjunto de soluciones no dominadas. Finalmente, todos esos conjuntos de soluciones no dominadas obtenidos son colocados en el CST, luego de esto se quitan las soluciones dominadas.
Después de esta inicialización, el CST contiene sólo soluciones no dominadas

y está listo para ser procesado por las demás etapas del algoritmo.

En cada iteración del algoritmo, y cada vez que se actualiza el CST al quitar las soluciones dominadas, se asigna a cada una de las soluciones que componen el CST un valor de ponderación, que será utilizado luego para la selección. El elemento seleccionado en cada momento será el que tenga el valor de ponderación mayor. El método llamado *Modified Crowding Distance Assignment* (MCDA) es el encargado de asignar el valor de ponderación a cada solución en el CST. El MCDA está basado en el operador llamado *Crowding Distance Operator* (CDO) propuesto en Deb (2002).
El método original propuesto por Deb (2002) parte de un conjunto de soluciones dividido en distintos niveles de dominancia. En cada nivel de dominancia se encuentran soluciones que no son dominadas por otras soluciones del mismo nivel o niveles superiores y que al mismo tiempo son dominadas por las soluciones de niveles inferiores. Siguiendo este razonamiento, las soluciones del primer nivel de dominancia conforman la frontera de Pareto. La división de la población de soluciones en fronteras se hace mediante un método llamado *Fast Non Dominated Sorting* (FNDS) que es una mejora del método *Non Dominated Sorting* propuesto por Srinivas y Deb (1994), cuyo diagrama de flujo muestra la Figura 3.1 del Capitulo 3.

El método FNDS busca en primer lugar las soluciones que pertenecen al primer nivel (frontera de Pareto), luego quita esas soluciones del conjunto de búsqueda y vuelve a buscar, en este caso las soluciones de segundo nivel. Realiza este proceso de manera iterativa hasta que todas las soluciones han sido ubicadas en un nivel. Finalizado este proceso, el CDO asigna un valor a cada solución basándose en su nivel y en la distancia entre ella y las soluciones más cercanas que pertenezcan al mismo nivel. Este proceso facilita la selección de las soluciones más aisladas dentro de cada nivel o frontera. De esta manera se busca cerrar los huecos en la frontera de Pareto, obteniendo más diversidad

y mejor cobertura.
La principal desventaja de esta técnica es que no hace un seguimiento de las soluciones que ya han sido elegidas en pasos anteriores por lo cual, en el caso que no se encuentre una solución mejor que la elegida en una iteración anterior, la misma solución será elegida en la siguiente iteración.
Para subsanar el problema del estancamiento del CDO hemos añadido a cada solución un contador de selección, este contador de selección es utilizado luego para asignar la ponderación a las soluciones que se encuentran dentro del conjunto de soluciones de trabajo. La ponderación de cada solución será menor cuanto mayor sea el número de veces que ha sido seleccionada. De esta forma se asegura que el algoritmo no se estancará seleccionando una y otra vez la misma solución. Una cuestión a tener en cuenta es que nuestro método MCDA no necesita de una ordenación previa de las soluciones en fronteras o niveles debido a que el CST contiene solamente soluciones no dominadas.
El pseudo-código del método *Modified Crowding Distance Assignment* puede verse en el Algoritmo 2.

4.1.3. Etapa voraz

La etapa voraz, o *Greedy*, tiene dos pasos identificables: el primer paso se denomina destrucción. En él, un bloque compuesto por d elementos consecutivos se quita de la solución elegida por el método de selección MDCA. El punto inicial de dicho bloque es elegido de manera aleatoria. En este aspecto, nuestro algoritmo IPG difiere también del algoritmo IG original, ya que en este la remoción de elementos no se realiza en bloques, sino un elemento a la vez.
El resultado del paso de destrucción es una solución parcial a la que se le han quitado ciertos elementos y un conjunto de elementos que serán re-insertados en el siguiente paso.
El segundo paso de la etapa voraz se encarga de reconstruir la solución, a través de la re-inserción iterativa de los elementos removidos en el paso anterior. Uno a uno, los d elementos removidos serán re-insertados en la solución parcial. En

Algoritmo 2 *Modified Crowding Distance Assignment.*

$WorkingSet =$ Conjunto de soluciones de trabajo
para todo $m \in M$ **hacer**
  Ordenar $WorkingSet$ según el objetivo m
  Asignar ∞ a los elementos extremos de $WorkingSet$
  $f_m^{\text{máx}} :=$ valor máximo para el objetivo m en $WorkingSet$
  $f_m^{\text{mín}} :=$ valor mínimo para el objetivo m en $WorkingSet$
  //Calcular las distancias para todos los restantes elementos en $WorkingSet$
  para $i := 2, \ldots, WorkingSet_{LastElement-1}$ **hacer**
    $WorkingSet_i.distance := WorkingSet_i.distance \times \frac{(WorkingSet_{i+1}.Objetive_m - WorkingSet_{i-1}.Objetive_m)}{(f_m^{\text{máx}} - f_m^{\text{mín}})}$
  fin para
fin para
//Ajustar el valor de fitness mediante el valor de selección en cada solución
//Primero es necesario obtener los valores extremos (para normalizar)
$MaxDist :=$ máx $(WorkingSet.distance)$
$MinDist :=$ mín $(WorkingSet.distance)$
//Los valores que contienen las soluciones extremas deben reemplazarse por el máximo valor aceptable(1)
para todo $i \in WorkingSet$ **hacer**
  si $WorkingSet_i.distance = \infty$ **entonces**
    $WorkingSet_i.fitness := \frac{1}{(WorkingSet_i.SelectionCounter+1)}$
  sino
    $WorkingSet_i.fitness := \frac{\frac{WorkingSet_i.distance + MinDist}{MaxDist - MinDist}}{(WorkingSet_i.SelectionCounter+1)}$
  fin si
fin para

la primera iteración, el primero de los d elementos removidos es re-insertado en todas las posiciones de la solución parcial. Si la solución elegida contenía n elementos antes de su destrucción, esto genera una cantidad de $n-d+1$ nuevas soluciones parciales cuyo tamaño es $n-d+1$ elementos. En consecuencia, el siguiente de los elementos removidos va a ser re-insertado en todas las $n-d+1$ posiciones de todas las soluciones parciales generadas en la iteración anterior.

En la segunda iteración se genera entonces un nuevo conjunto de soluciones parciales de tamaño $(n-d+1) \times (n-d+2)$. Este proceso se reitera hasta que todos los elementos removidos hayan sido re-insertados y se genera un conjunto de soluciones completas.
Al finalizar este proceso, el número de soluciones completas generadas en la etapa voraz del algoritmo es igual a:

$$\prod_{i=1}^{d}(n-d+i) \geq \prod_{i=1}^{d}(n-d) + \prod_{i=1}^{d} i = (n-d)^d + d!$$

Cabe destacar, que el proceso de destrucción-reconstrucción planteado en el nuevo algoritmo es radicalmente diferente al proceso que se utiliza en el algoritmo IG original de Ruiz y Stützle (2007), donde sólo se mantiene una solución completa durante todo el proceso y se utiliza la heurística constructiva regular NEH. El método aquí propuesto mantiene un conjunto de soluciones parciales en las que se re-insertarán todos los elementos removidos.
El procedimiento propuesto tiene una desventaja notoria. Para valores grandes de n, el número de soluciones parciales generadas crece de manera exponencial. Por ejemplo, para $n = 25$ y $d = 5$ el número total de soluciones completas generadas sería mayor que 3×10^6. El generar una cantidad tan grande de soluciones ocasiona un problema fácilmente detectable: la velocidad del algoritmo se ve seriamente perjudicada debido a que una gran cantidad de soluciones debe ser evaluada en cada iteración. Para evitar este inconveniente, y obtener un algoritmo más veloz, cada vez que se genera un nuevo conjunto de soluciones parciales, solamente se mantienen las soluciones parciales no-dominadas, mientras que las soluciones parciales dominadas se descartan. En consecuencia, en el siguiente paso, el siguiente elemento removido se re-inserta sólo en las soluciones parciales no-dominadas.
El Algoritmo 3 describe la etapa voraz de manera más precisa mediante pseudo-código.

Algoritmo 3 Etapa voraz.

$SelectedSolution =$ Aplicar MCDA sobre la solución elegida
$PartialSolution =$ destruir $SelectedSolution$ mediante la remoción de un bloque de d elementos consecutivos
$DestructedElements =$ Conjunto de d elementos que han sido extraídos de $SelectedSolution$
$PartialSolutionsSet = PartialSoluction$ //Conjunto de soluciones parciales utilizadas en esta etapa
para todo $i \in DestructedElements$ **hacer**
  para todo $n \in PartialSolutionsSet$ **hacer**
    $Counter := 0$
    para todo j posición de $PartialSolutionsSet_n$ **hacer**
      $NewPartialSolutionsSet_{Counter}$ = Insertar el elemento i en la posición j de $PartialSolutionsSet_n$
      $Counter = Counter + 1$
    fin para
  fin para
  $PartialSolutionsSet$:= quitar las soluciones parciales dominadas del conjunto $NewPartialSolutionsSet$
fin para
retornar: $PartialSolutionsSet$

El conjunto de soluciones no dominadas obtenido por la etapa voraz se añade al conjunto de soluciones de trabajo, luego de ello se quitan las soluciones dominadas, conformando un nuevo CST. Finalmente, el método MCDA se aplica nuevamente al CST y se elige una nueva solución para la búsqueda local.

4.1.4. Etapa de búsqueda local

Para refinar y profundizar la búsqueda en el espacio cercano a la solución elegida utilizaremos una etapa de búsqueda local simple. Si bien existen en la literatura métodos de búsqueda local complejos, y que en general consumen mucho tiempo de proceso, nuestra meta es conseguir un algoritmo rápido

y eficaz. Por ello nos inclinamos hacia la utilización de búsquedas locales simples y más veloces.

La etapa de búsqueda local consiste en repetidamente seleccionar, de manera aleatoria, un elemento de la solución elegida, quitarlo y luego re-insertarlo dentro de las n_{neigh} posiciones adyacentes a la izquierda y a la derecha de su posición original. Este proceso se repite un número de veces que depende de un contador de selección ($SelectionCounter$). Cuanto mayor es el valor del contador de selección, más intensiva será la búsqueda local. Esto asegura que en los primeros pasos del algoritmo, cuando $SelectionCounter$ tiene valores pequeños, la búsqueda local utiliza poco tiempo de proceso. Y al mismo tiempo permite que la búsqueda se amplíe conforme las mismas soluciones hayan sido elegidas ya varias veces. Sin embargo, para evitar que al final del algoritmo las búsquedas locales se hagan demasiado exhaustivas, hemos puesto un límite máximo a la cantidad de elementos que se extraen.

Igual que ocurre con la etapa voraz, la búsqueda local mantiene un conjunto de soluciones no dominadas. En cada iteración, el elemento seleccionado aleatoriamente se re-inserta en una de las posibles posiciones y se genera una nueva solución. Al finalizar el proceso de insertar el elemento seleccionado en todas las n_{neigh} posiciones a la izquierda y a la derecha de su posición original, se obtiene un conjunto de soluciones de tamaño $n_{neigh} \times 2$.
Finalmente se quitan las soluciones dominadas del conjunto de soluciones generadas por la búsqueda local, y estas soluciones se añaden al conjunto de soluciones de trabajo del algoritmo.

Luego de la búsqueda local, se ejecuta la etapa de reinicio, siempre que se cumplan los requisitos para ello. En este caso, se añade el CST actual al archivo de soluciones no dominadas y se genera un nuevo CST. Si la etapa de reinicio no se ejecuta, el algoritmo vuelve a comenzar con la etapa voraz

previa selección de una nueva solución.

4.1.5. Etapa de reinicio

Como comentamos anteriormente, al realizar experimentos previos con el algoritmo IPG básico detectamos que para instancias pequeñas y medianas este algoritmo tendía a estancarse en óptimos locales y era incapaz de avanzar en la búsqueda de nuevas soluciones. Para solucionar este inconveniente decidimos añadirle una etapa de reinicio. Esta etapa consiste archivar el conjunto de soluciones de trabajo actual y luego crear uno nuevo conformado por soluciones generadas de manera aleatoria. Este seguramente es el esquema de reinicio más simple que se puede hacer, permitiendo que el algoritmo escape de situaciones de estancamiento en óptimos locales.
Una de las grandes dificultades a la hora de diseñar un método de reinicio es determinar el momento más idóneo para su ejecución. Un enfoque simplista consiste en reiniciar el algoritmo cuando el CST no ha cambiado durante un cierto numero de iteraciones. El problema en este caso es que determinar si el CST ha cambiado no es una cuestión trivial, incluso puede ser un proceso que consume mucho tiempo y además puede hacerse de varias maneras.
Siguiendo la premisa de diseño de nuestro algoritmo, que debe ser simple, eficaz y veloz, hemos elegido ejecutar la etapa de reinicio cuando el tamaño de el conjunto de soluciones de trabajo no ha cambiado luego de transcurrir un número específico de iteraciones. Más aún, algunas pruebas iniciales mostraron que el número de iteraciones que se deben esperar antes de reiniciar el algoritmo debe depender del tamaño de la instancia. En este aspecto hemos determinado que la etapa de reinicio se ejecutará luego de transcurridas $n \times 2$ (donde n es el número de trabajos en la instancia) iteraciones sin cambios en el tamaño del CST. Este valor ha sido obtenido mediante la ejecución de un experimento de calibración que presentaremos más adelante, en la Sección 4.2. En la siguiente sección explicaremos la manera en la que se llevan a cabo los experimentos y como se compararan sus resultados.

4.2. Análisis experimental

Como vimos en la Sección 1.1.2, la comparación de resultados en la optimización multi-objetivo es mucho más compleja que cuando se tratan problemas de un solo objetivo. Realmente, el único caso en el cual los resultados de dos algoritmos multi-objetivo son claramente diferenciables o comparables es cuando el conjunto de soluciones no dominadas resultante de uno de ellos domina completamente al conjunto resultante del otro algoritmo. Esto solamente ocurre cuando todas las soluciones de la segunda frontera de Pareto son dominadas por al menos una de las soluciones de la primera.

Para los experimentos de este capítulo utilizaremos los dos indicadores de calidad utilizados en el Capítulo 3, que han sido probados como Pareto-compatibles. El primero de ellos es el indicador de hipervolumen I_H, para el cual utilizaremos un punto de referencia obtenido al incrementar el peor de los valores de cada objetivo en un 20 %. Debido a que los valores de los objetivos están normalizados, el máximo valor de hipervolumen I_H se puede obtener mediante el producto de los valores de los puntos de referencia: $1{,}2 \times 1{,}2 = 1{,}44$.
El segundo método de medición que utilizaremos es indicador épsilon I_ε. Los valores de los objetivos deben ser normalizados y escalados antes del cálculo del I_ε, de esta manera los valores del indicador varían entre 1 y 2. Un valor más cercano a 1 indica que la frontera de Pareto dada es cercana a la frontera de referencia, mientras que un valor cercano a 2 indica el caso contrario.
La ventaja de usar dos indicadores de calidad al mismo tiempo es que esto permite detectar situaciones en las que dos fronteras son incomparables, cuando estos indicadores dan resultados contradictorios. Este enfoque fue utilizado exitosamente con anterioridad en la comparación de los más de 20 algoritmos presentes en el Capítulo 3 y en el artículo de Minella, Ruiz y Ciavotta (2008a). El conjunto de instancias del banco de pruebas es el mismo utilizado en el

Capítulo 3, que contiene 110 instancias, divididas en 11 grupos de 10 instancias cada uno. Cada grupo contiene diferentes combinaciones de número de trabajos n y número de máquinas m. Para obtener más detalles del conjunto de instancias de banco de pruebas, el lector puede dirigirse al Capítulo 3.

Para utilizar el indicador I_ε es necesario obtener previamente el conjunto de soluciones de referencia. Para ello hemos recurrido a las soluciones generadas en los experimentos del Capítulo 3. Para cada instancia se han recogido todas las soluciones no dominadas generadas por todos los algoritmos en todas sus ejecuciones.

La siguiente sección trata de la calibración del algoritmo propuesto. En ésta se detalla la metodología seguida para diseñar y parametrizar el algoritmo resultante. Cabe destacar que gracias a esta manera ordenada, precisa y científica de afrontar el desarrollo del algoritmo, hemos sido capaces de mejorar la calidad de sus resultados, no solo mediante su correcta parametrización, sino también a través de la inclusión de una nueva etapa: la etapa de reinicio.

4.2.1. Calibración del algoritmo

Antes de comparar el nuevo algoritmo propuesto contra los mejores algoritmos de la literatura, llevamos a cabo un experimento de calibración. El objetivo final de la calibración es determinar el mejor valor para cada uno de los parámetros del algoritmo. También hemos utilizado este experimento de calibración para determinar la importancia de cada etapa o parte del algoritmo y su colaboración en la obtención de la solución final.

Empleamos la técnica de Diseño de Experimento (*Design Of Experiment* o DOE), donde cada uno de los factores que afectan el rendimiento del algoritmo RIPG se prueban en un diseño factorial completo, que luego analizamos mediante un análisis de varianza multi-factor (ANOVA). En total, probamos

5 factores: 1) La inicialización del algoritmo, a dos niveles: utilizando una inicialización aleatoria, o el método propuesto por nosotros mismos. 2) El tamaño del bloque de destrucción dentro de la etapa voraz d a dos niveles: 5 y 10. 3) El tamaño del vecindario en la búsqueda local o n_{neigh}, también probado a dos niveles: 3 y 5. 4) El número de iteraciones de búsqueda local, probado a dos niveles: 5 y $SelectionCounter$ (como ya explicamos antes $SelectionCounter$ es un contador que indica la cantidad de veces que una solución ha sido seleccionada con anterioridad). 5) Operador de reinicio, probado a tres niveles: sin reinicio, reiniciar luego de 10 iteraciones sin mejoras y reiniciar luego de $n \times 2$ iteraciones sin mejorar, donde n representa el número de trabajos de la instancia a procesar. Para facilitar el experimento, hemos combinado los factores 3) y 4) en un único factor con cinco niveles, donde el último nivel es la desactivación de la búsqueda local.
Finalmente nuestro diseño de experimento consta de 4 factores con 2, 2, 5 y 3 niveles respectivamente. Esto da un total de 60 configuraciones diferentes de algoritmos. Cada configuración se ha probado contra 110 instancias, realizando 10 ejecuciones por cada instancia. Hemos obtenido un número total de 66.000 resultados de estos experimentos. Hemos codificado el algoritmo RIPG utilizando Delphi 2007, al igual que todos los algoritmos del Capítulo 3. Hemos ejecutado los experimentos en un cluster de 12 ordenadores con un procesador Intel Core 2 Duo E6000 de 2.4 GHz de velocidad y con 1 Gbyte de memoria RAM. Sólo un procesador es utilizado en cada ordenador. Cada configuración se ejecutó durante un tiempo exacto de CPU que depende del tamaño de la instancia y se calcula a través de esta fórmula: $T_{CPU} = n \cdot m/2 \cdot t$ ms. El tiempo total consumido por la calibración asciende a 41,57 días de CPU. En los experimentos ANOVA hemos utilizado el Indicador de Hipervolumen I_H y el Indicador Épsilon I_ε como variables respuesta.

Debido a que el ANOVA es una herramienta de inferencia estadística paramétrica, en primer lugar es necesario comprobar las tres principales hipótesis

que son normalidad, homoscedasticidad e independencia de los residuos. Al analizar los resultados de los residuos de los datos obtenidos por el experimento, las tres hipótesis son satisfechas.

Luego de la calibración observamos que la inicialización aleatoria tiene un rendimiento muy pobre. En cambio la inicialización propuesta por nosotros tiene resultados muy superiores. El mejor tamaño del bloque de destrucción resulta ser $d = 5$. Mientras que los factores que afectan a la búsqueda local que han obtenido los mejores resultados son: para el tamaño del vecindario de inserción $n_{neigh} = 5$ y el número de iteraciones de búsqueda local se iguala al contador de selección de la solución $SelectionCounter$. Finalmente, la activación de la etapa de reinicio se llevará a cabo cuando el numero de iteraciones sin mejorar (cambiar el tamaño de la población) sea igual a $n \times 2$.

Antes mencionamos que durante este experimento también se comprobó la relevancia de cada etapa del algoritmo propuesto. Algunos resultados obtenidos de esta prueba se pueden ver en la Figura 4.1 para el indicador épsilon (a) y para el indicador de hipervolumen (b), respectivamente. Las configuraciones mostradas en la imagen corresponden a: GP (*Greedy Phase*) indica que se ha probado solamente la parte voraz, sin búsqueda local y sin la etapa de reinicio, GP+LS (*Greedy Phase* + *Local Search*)representa el algoritmo básico propuesto en primera instancia IPG, que consta de la etapa voraz y la etapa de búsqueda local. La siguiente configuración GP+R (*Greedy Pahse* + *Restart*) se conforma de la etapa voraz y la etapa de reinicio, sin búsqueda local. Finalmente la configuración GP+LS+R conforma el algoritmo denominado RIPG que consta de todas las etapas propuestas.
Podemos observar que cada etapa propuesta para el algoritmo mejora los resultados iniciales de la etapa voraz ya que la configuración GP es la que obtiene peores resultados. Muy destacable es el hecho que el impacto de la etapa de reinicio dentro del algoritmo es muy elevado. Como puede verse la

combinación GP+R produce resultados un poco superiores a la combinación GP+LS. Este resultado no sólo justifica la inclusión de esta etapa en el algoritmo, sino que nos impulsa a seguir el desarrollo de este tipo de métodos de mejora de soluciones.

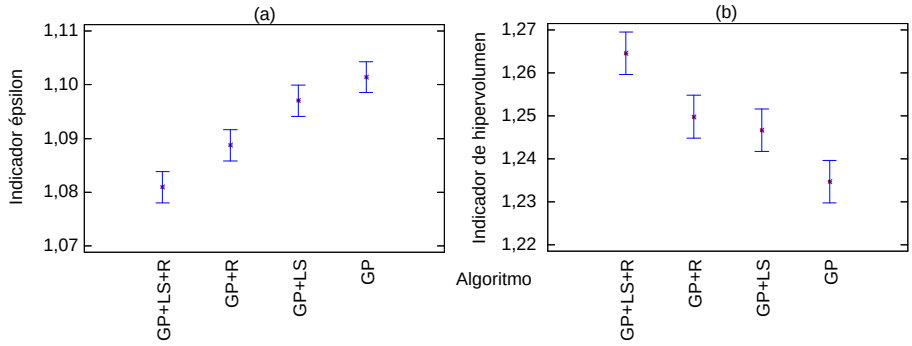


Figura 4.1: Gráfico de medias e intervalos de confianza de Tukey de 99 % para el indicador épsilon (a) y el indicador de hipervolumen (b) en el experimento ANOVA para varias configuraciones del algoritmo RIPG.

4.2.2. Evaluación computacional

Comparamos ahora el algoritmo propuesto contra los tres mejores algoritmos de acuerdo con el artículo de revisión de Minella, Ruiz y Ciavotta (2008a), además de esto incluimos dos métodos publicados en los años posteriores a la aparición de nuestro artículo. La tabla 4.1 muestra la lista de todos los algoritmos que han sido utilizados en esta comparación. Algunos de ellos fueron re-implementados para los experimentos del Capítulo 3, mientras que los más nuevos han sido re-implementados para los experimentos del capítulo actual. Todos estos algoritmos han sido re-implementados utilizando el lenguaje Delphi 2007 siguiendo las instrucciones de los artículos originales. La única modificación que se ha llevado a cabo es la inclusión de aceleraciones en

la evaluación incluidas en el algoritmo MOIGS de Framiñan y Leisten (2008), mejorando su rendimiento.

Algoritmo	Tipo	Autor/es	Año
MOTS	Búsqueda Tabú	Armentano y Arroyo	2004
MOGALS	Algoritmo Genético	Arroyo y Armentano	2005
MOSAII	Recocido Simulado	Varadharajan y Rajendran	2005
hMGA	Algoritmo Genético	Yandra y Tamura	2007
MOIGS	Algoritmo Voraz Iterativo	Framiñan y Leisten	2008
MOSAII_M	Recocido Simulado	adaptación de MOSAII	2008
RIPG	Algoritmo Voraz Iterativo	Minella, Ruiz y Ciavotta	2011

Tabla 4.1: Lista de algoritmos re-implementados o adaptados incluidos en este trabajo.

Las pruebas de los algoritmos fueron realizadas de igual manera y en el mismo *cluster* de ordenadores que las pruebas del Capítulo 3. En todas nuestras pruebas realizamos dos combinaciones multi-objetivo diferentes: $C_{\text{máx}}$-tiempo total de flujo y $C_{\text{máx}}$-tardanza total. También hemos cambiado el criterio de parada de acuerdo con el tamaño de la instancia de la misma manera que explicamos en el Capítulo 3. Todos los experimentos se llevaron a cabo en el mismo grupo de ordenadores mencionados en la sección 4.2.1. Para el criterio de parada utilizamos dos valores diferentes de t: 100 y 200. Cada algoritmo se ejecutó 10 veces (réplicas) para cada combinación de criterio de parada y par de objetivos.

Para la ejecución de cada réplica de cada algoritmo, se ha elegido un ordenador aleatoriamente y los resultados se han recogido al finalizar el experimento. Un total de 30.800 resultados (o aproximaciones a la frontera de Pareto) han sido recogidos del experimento final, resultado de la ejecución de 10 réplicas de los 7 algoritmos, para 2 combinaciones diferentes de objetivos y 2 criterios de parada.

Hemos analizado los resultados obtenidos de las pruebas utilizando el test paramétrico multi-factor de análisis de varianza (ANOVA) en conjunción con el test no paramétrico basado en rangos de Friendman para los dos indicadores propuestos (I_H y I_ε). Llevamos a cabo los dos tests estadísticos para cada combinación de objetivos, indicadores de rendimiento y criterio de parada, obteniendo un total de 16 tests estadísticos. Las hipótesis de normalidad, homogeneidad de varianza e independencia de los residuos fueron comprobadas en los datos antes de aplicar los tests ANOVA.
Debido a que ejecutamos cuatro tests estadísticos diferentes sobre el mismo conjunto de resultados ha sido necesario aplicar una corrección a los niveles de confianza, ya que los mismos datos están siendo utilizados para hacer más de una inferencia. Para contrarrestar este problema potencial hemos empleado el ajuste de Bonferroni, y hemos cambiado el nivel de significancia ajustado α_s a $\frac{\alpha}{4} = \frac{0{,}05}{4} \simeq 0{,}01$. Esto significa que todos los tests se han llevado a cabo con un nivel de confianza ajustado de $0,01$, con lo que hemos obtenido un nivel de confianza real de $0,05$. Los resultados de estos tests se muestran en forma de gráficas de ANOVA con intervalos de confianza del $99\,\%$ (nivel de confianza ajustado de $95\,\%$).

Por razones de claridad, sólo comentaremos en este capítulo las gráficas para los tests ANOVA. Los resultados de test de rangos se pueden consultar en el Apéndice A. En las subsiguientes sub-secciones discutiremos los resultados obtenidos para algunas de las combinaciones de objetivos y veremos, también, las tablas y gráficos ANOVA de estos experimentos.

4.2.2.1. Resultados para $C_{\max}$ y tiempo total de flujo

La Tabla 4.2 muestra los valores promedio para los indicadores de hipervolumen y épsilon con la combinación de objetivos $C_{\max}$ y tiempo total de flujo. Los criterios de parada utilizados han sido: $t = 100$ y $t = 200$. Cada valor que se muestra en las tablas es un promedio obtenido de los resultados de evaluación de 110 instancias y 10 réplicas por instancia (un total de 1.100

valores). Los valores en las tablas se muestran en orden descendente con respecto al indicador de hipervolumen, lo que significa que el primer algoritmo que se muestra es el que obtiene el mejor rendimiento para este indicador. Los mejores valores para ambos indicadores se muestran resaltados en negrita.

Al observar la Tabla 4.2 se puede ver que los resultados del RIPG mejoran los

	Tiempo	100			200	
#	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
1	RIPG	**1,270630**	**1,078920**	RIPG	**1,295480**	**1,066000**
2	MOSAII$_M$	1,227270	1,114630	MOSAII$_M$	1,248160	1,105500
3	MOIGS	1,170090	1,135420	MOIGS	1,212110	1,111510
4	MOSAII	1,154200	1,150990	MOSAII	1,146800	1,153970
5	MOGALS	1,130070	1,169700	MOGALS	1,151050	1,156690
6	MOTS	1,041530	1,230810	MOTS	1,066500	1,212490
7	hMGA	0,569154	1,547800	hMGA	0,592918	1,528700

Tabla 4.2: Resultados para la combinación de objetivos de $C_{\max}$ y tiempo total de flujo con tiempos de parada $t = 100$ y $t = 200$. Los algoritmos están ordenados de acuerdo a I_H.

resultados de los demás algoritmos para ambos indicadores y para ambos criterios de parada. La segunda posición es ocupada por el algoritmo MOSAII$_M$ que es la versión modificada del algoritmo MOSAII. Si bien esta tabla solamente muestra las medias totales para todas las instancias, puede llevarse a cabo un análisis más detallado de los valores que muestra. Por ejemplo, Framiñan y Leisten (2008) mostraron que el algoritmo MOIGS propuesto por ellos tiene un mejor rendimiento que el algoritmo MOSAII. Sin embargo, los autores solamente consideraron instancias pequeñas y medianas de hasta 100 trabajos. En nuestros experimentos, con instancias de hasta 200 trabajos, el algoritmo MOSAII muestra un rendimiento superior cuando se trata de instancias con muchos trabajos. La Figura 4.2 muestra los resultados para el experimento comparativo de MOIGS Y MOSAII, para el indicador de hipervolumen y el indicador épsilon para el criterio de parada de $t = 100$ y la combinación de objetivos de $C_{\max}$ y tardanza total. El algoritmo MOIGS obtiene mejores

resultados para instancias de 20 y 50 trabajos, tanto para I_H como para I_ε. En cambio para instancias con 100 trabajos el rendimiento para I_H es igual en ambos algoritmos y para I_ε sigue siendo superior MOIGS. La situación cambia rotundamente para 200 trabajos, en estas condiciones el rendimiento del algoritmo MOIGS cae por debajo de MOSAII para ambos indicadores.

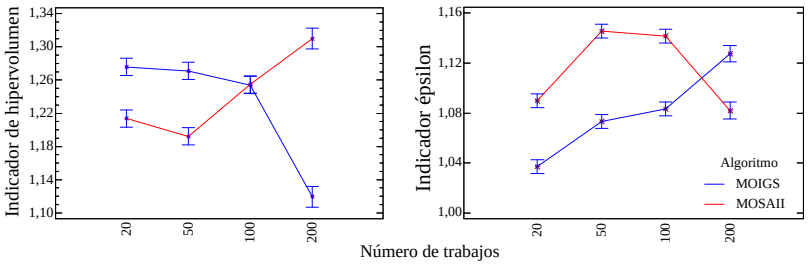


Figura 4.2: Gráfico de medias e intervalos de confianza de Tukey al 99 % para el indicador épsilon (a) y para el indicador de hipervolumen (b) en el test ANOVA para MOIGS y MOSAII.

La Figura 4.3 muestra los resultados del test ANOVA para $t = 100$ y para I_ε, mientras que la Figura 4.4 muestra los resultados para $t = 100$ y para I_H. Ambas figuras se refieren a los resultados para la combinación de objetivos de $C_{\text{máx}}$ y tiempo total de flujo. De manera similar, las Figuras 4.5 y 4.6 muestran los resultados para la misma configuración de experimento pero con $t = 200$.

Puede deducirse de los resultados que todos los algoritmos han mostrado diferencias estadísticamente significativas (salvo algunas pequeñas excepciones) en todas las pruebas. También puede observarse como el rendimiento del algoritmo RIPG es claramente superior en todas las pruebas.
Los resultados previamente mostrados son promedios totales. Un aspecto interesante, que deja de lado, es estudiar el comportamiento de los diferentes algoritmos de acuerdo al tamaño de la instancia. Las Figuras 4.7 y 4.8 muestran los distintos resultados obtenidos para cada grupo de instancias para los 3

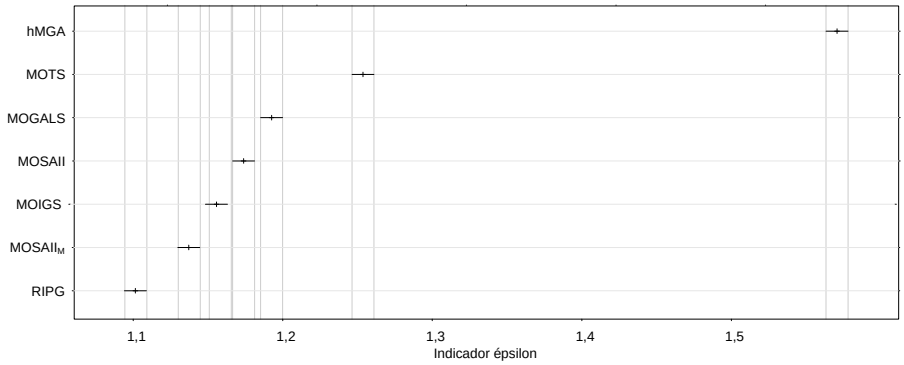


Figura 4.3: Resultados del test ANOVA para el indicador épsilon con $t = 100$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de Tukey al 99 % (nivel de confianza ajustado al 95 %).

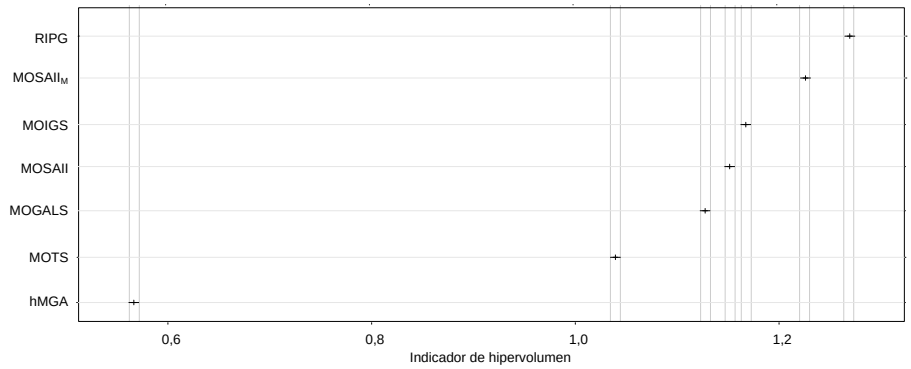


Figura 4.4: Resultados del test ANOVA para el indicador de hipervolumen y $t = 100$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de Tukey al 99 % (nivel de confianza ajustado al 95 %).

algoritmos que han dado consistentemente los mejores resultados: MOIGS, MOSAII$_M$ y RIPG. En instancias pequeñas todos los algoritmos obtienen resultados similares. Por otro lado, cuando se trata de instancias grandes, puede observarse una clara diferencia entre el algoritmo RIPG y los demás,

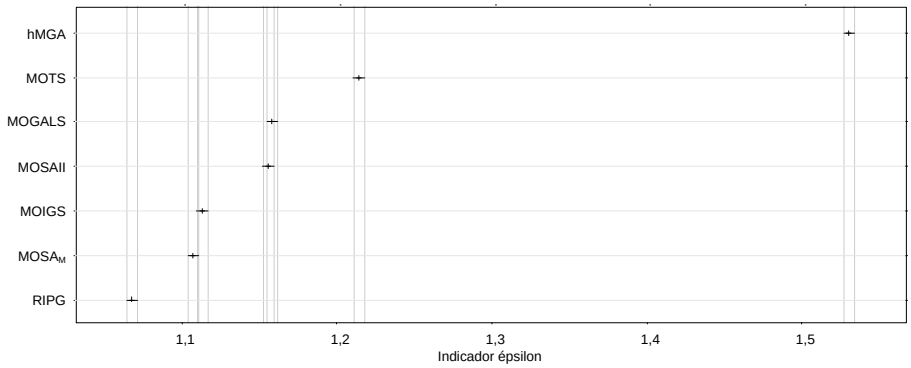


Figura 4.5: Resultados del test ANOVA para el indicador épsilon con $t = 200$ para los objetivos de $C_{\text{máx}}$ y tiempo total de flujo con intervalos de Tukey al 99 % (nivel de confianza ajustado al 95 %).

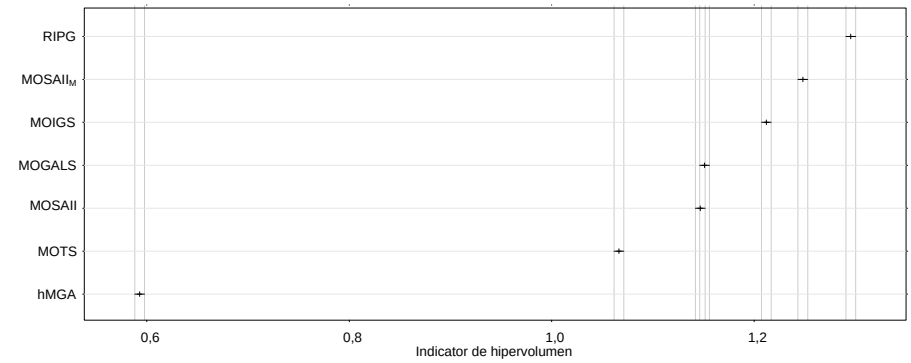


Figura 4.6: Resultados del test ANOVA para el indicador de hipervolumen y $t = 200$ para los objetivos de $C_{\text{máx}}$ y tiempo total de flujo con intervalos de Tukey al 99 % (nivel de confianza ajustado al 95 %).

que obtienen un rendimiento inferior. También el rendimiento del algoritmo MOIGS se ve afectado negativamente en las instancias más grandes.

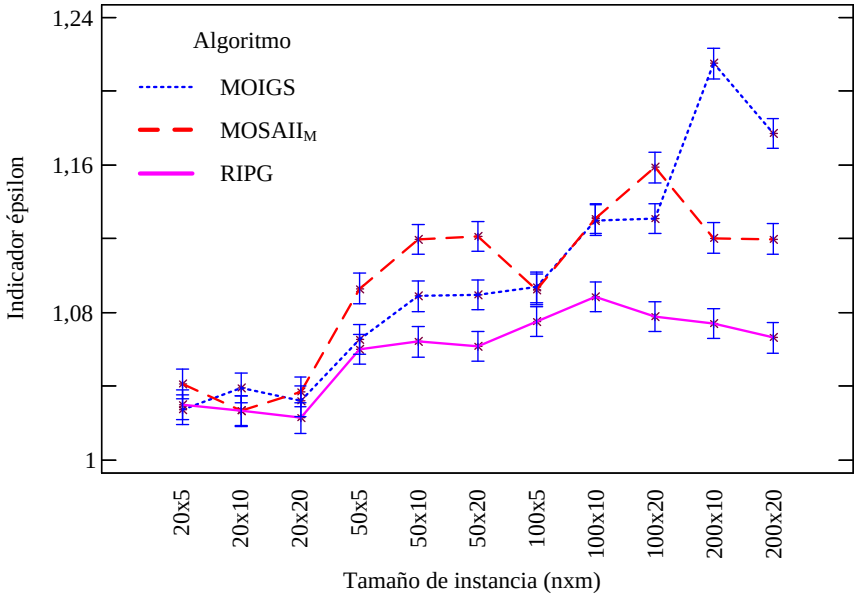


Figura 4.7: Gráfico de medias e intervalo de confianza de Tukey al 99 % para el experimento de ANOVA con el indicador épsilon para los tres algoritmos contra el tamaño de la instancia, con $C_{\max}$ y tiempo total de flujo como combinación de objetivos.

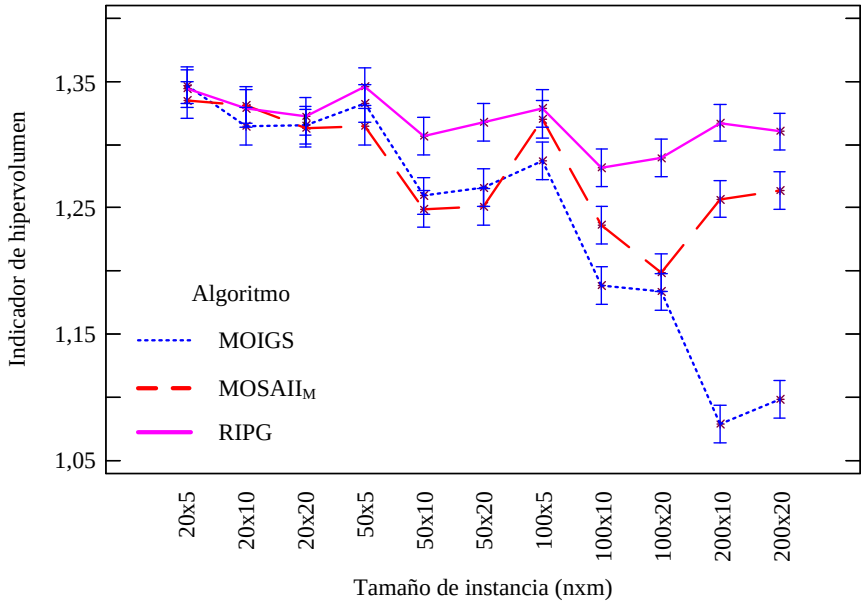


Figura 4.8: Gráfico de medias e intervalo de confianza de Tukey al $99\,\%$ para el experimento de ANOVA con el indicador de hipervolumen para los tres los algoritmos contra el tamaño de la instancia, con $C_{\text{máx}}$ y tiempo total de flujo como combinación de objetivos.

4.2.2.2. Resultados para $C_{\max}$ y tardanza total

La Tabla 4.3 muestra los resultados para la combinación de objetivos de $C_{\max}$ y tardanza total. Nuevamente, el algoritmo RIPG presenta los mejores resultados en términos de I_H y de I_ε, el algoritmo MOSAII$_M$ ocupa el segundo lugar, mientras hMGA obtiene el peor rendimiento. Las mismas conclusiones se desprenden al observar los tests estadísticos cuyos resultados se muestran en las figuras que van desde la 4.9 a la 4.12.

	Tiempo	100			200	
#	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
1	RIPG	**1,256220**	**1,08086**	RIPG	**1,280370**	**1,066090**
2	MOSAII$_M$	1,214630	1,11139	MOSAII$_M$	1,234290	1,102120
3	MOIGS	1,153630	1,13636	MOIGS	1,194390	1,113260
4	MOSAII	1,147280	1,14568	MOSAII	1,136630	1,149310
5	MOGALS	1,026660	1,24139	MOGALS	1,046410	1,227070
6	MOTS	0,890548	1,33638	MOTS	0,917784	1,318060
7	hMGA	0,611876	1,51659	hMGA	0,636000	1,497120

Tabla 4.3: Resultados para la combinación de objetivos de $C_{\max}$ y tardanza total con tiempos de parada de $t = 100$ y $t = 200$. Los algoritmos están ordenados de acuerdo a I_H.

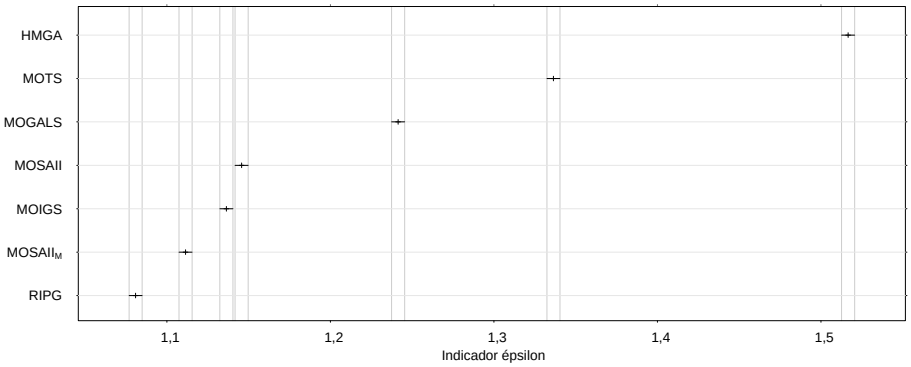


Figura 4.9: Resultados del test ANOVA para el indicador épsilon con $t = 100$ para los objetivos $C_{\max}$ y tardanza total con un intervalo de confianza de Tukey al 99 % (nivel de confianza ajustado al 95 %).

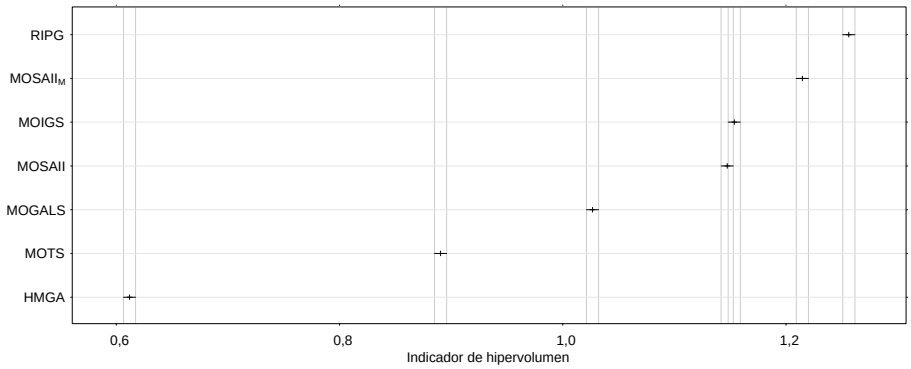


Figura 4.10: Resultados del test ANOVA para el indicador de hipervolumen y $t = 100$ para los objetivos $C_{\max}$ y tardanza total con un intervalo de confianza de Tukey al 99 % (nivel de confianza ajustado al 95 %).

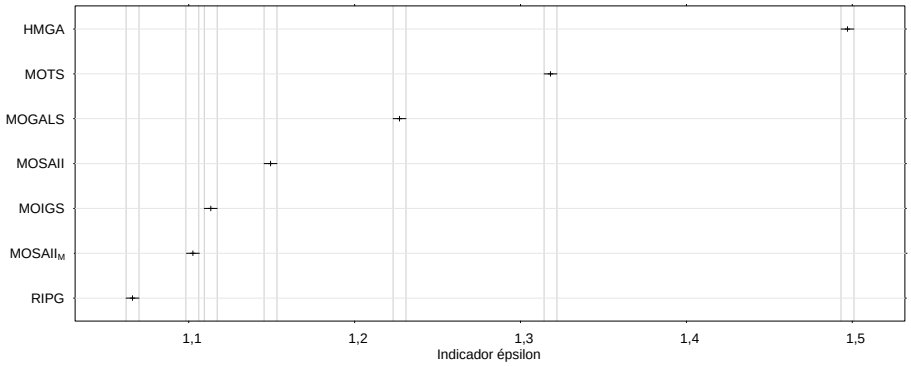


Figura 4.11: Resultados del test ANOVA para el indicador épsilon con $t = 200$ para los objetivos $C_{\max}$ y tardanza total con un intervalo de confianza de Tukey al 99 % (nivel de confianza ajustado al 95 %).

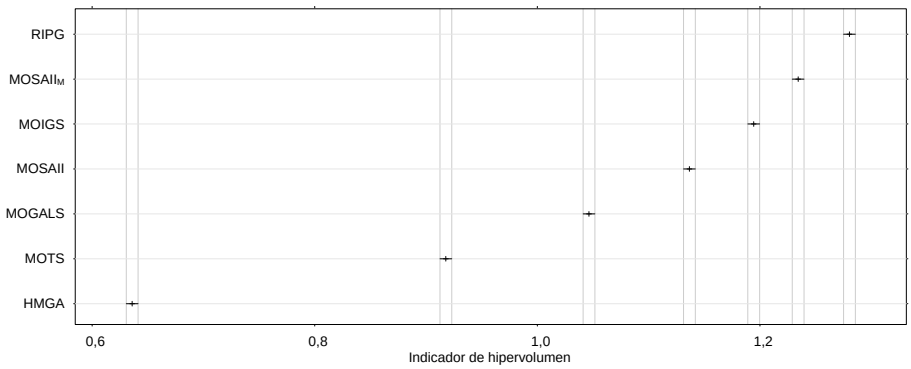


Figura 4.12: Resultados del test ANOVA para el indicador hipervolumen con $t = 200$ para los objetivos $C_{\max}$ y tardanza total con un intervalo de confianza de Tukey al 99 % (nivel de confianza ajustado al 95 %).

Como podemos ver, la mayoría de las gráficas indican que existe una diferencia fuerte y estadísticamente significativa entre los algoritmos. Cabe destacar que para esta combinación de objetivos las diferencias entre los algoritmos se agudizan. De manera similar al experimento anterior, también mostramos las diferencias de rendimiento de los algoritmos de acuerdo al tamaño de la instancia mediante las Figuras 4.13 y 4.14. Es interesante destacar que los tres algoritmos se comportan de manera similar a como se comportan en los experimentos previos con $C_{\text{máx}}$ y tiempo total de flujo.

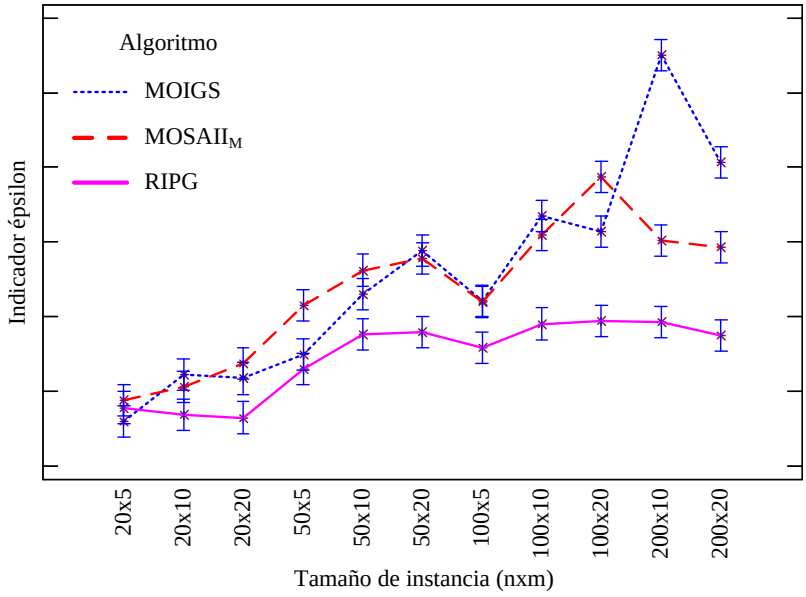


Figura 4.13: Gráfico de medias e intervalo de confianza de Tukey al 99 % para el indicador épsilon en el experimento ANOVA para los tres algoritmos contra el tamaño de instancia. Objetivos de $C_{\text{máx}}$ y tardanza total.

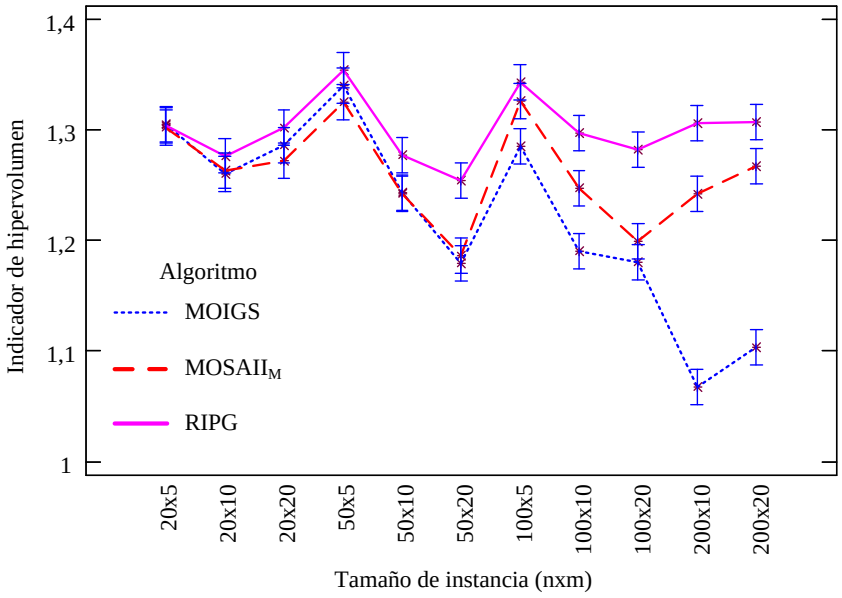


Figura 4.14: Gráfico de medias e intervalo de confianza de Tukey al 99 % para el indicador de hipervolumen en el experimento ANOVA para los tres algoritmos contra el tamaño de instancia. Objetivos de $C_{\max}$ y tardanza total.

4.2.3. *Differential Empirical Attainment Functions*

Para completar el análisis de los resultados previamente presentados, comparamos mediante el uso de gráficas de *empirical attainment functions* o EAF, el algoritmo RIPG propuesto al principio de este capítulo contra el algoritmo MOSAII$_M$, que obtuvo el segundo mejor rendimiento de acuerdo a nuestros experimentos. Estas gráficas muestran, para cada instancia seleccionada, qué área es más probable que sea dominada por cada algoritmo en una ejecución. Utilizamos gradientes de colores para diferenciar las zonas con mayor probabilidad de ser dominadas por un algoritmo (color más intenso) de las zonas en las que es poco probable que el algoritmo obtenga soluciones dominantes (zonas más claras) y zonas donde el algoritmo no es capaz de generar soluciones (zonas blancas).

Si bien las gráficas de EAF son herramientas poderosas que nos dan una descripción espacial sobre el comportamiento de un algoritmo, su mayor desventaja es que no permiten la comparación entre varios algoritmos, ya que cada gráfica corresponde a los resultados de un único algoritmo para una única instancia. Para resolver este problema utilizamos las gráficas de *Differential Empirical Attainment Functions* o Diff-EAF propuestas en López-Ibáñez, Paquete y Stützle (2006). La Diff-EAF muestra una función que expresa, para cada punto del espacio de soluciones, la probabilidad de ser dominado por sólo uno de los dos algoritmos comparados. Esta función se obtiene tras calcular la diferencia entre dos EAF pertenecientes, cada una, a un algoritmo diferente. Utilizamos dos gradientes de colores (azul y rojo) para señalar los distintos valores en cada punto de la gráfica. Cada color se asigna a uno de los algoritmos comparados. De esta forma podemos mostrar dos EAF como una única imagen que permite identificar que algoritmo prevalece en cada zona del espacio de objetivos. La intensidad del color asignado a cada algoritmo muestra la probabilidad de que ese algoritmo domine un punto en el espacio sobre el otro algoritmo. Los puntos con poca intensidad de color muestran diferencias

menores entre los algoritmos. Mientras que las zonas en blanco indican que los algoritmos comparados no pueden generar soluciones dominantes o que ambos algoritmos tienen la misma probabilidad de generar soluciones (la diferencia es nula). La Figura 4.15 muestra las gráficas de dos EAF y la diferencia diff-EAF de los algoritmos RIPG y MOSAII$_M$, calculadas a partir de 100 ejecuciones de cada algoritmo contra una instancia de gran tamaño, con 200 trabajos y 10 máquinas (Ta091), considerando los objetivos de $C_{\max}$ y tardanza total. La gráfica de Diff-EAF muestra claramente que el algoritmo RIPG domina consistentemente al algoritmo MOSAII$_M$ en todo el espacio de objetivos para la instancia dada. De manera similar, la Figura 4.16 muestra una gráfica similar para una instancia de tamaño mediano (Ta021) con 20 trabajos y 20 máquinas.

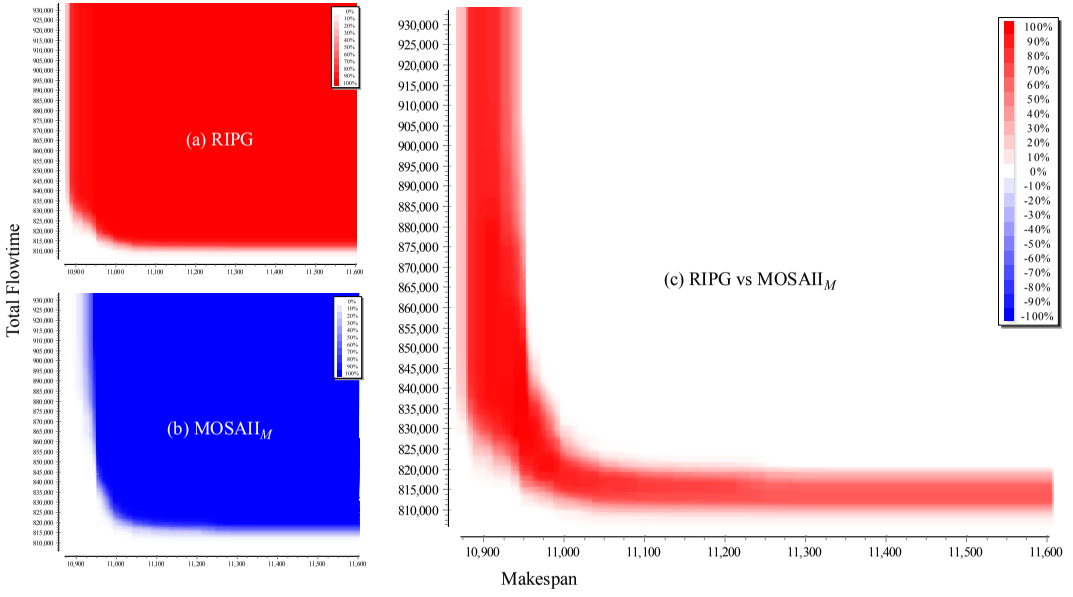


Figura 4.15: Gráfica de *Empirical Attainment Function*. $t = 100$ para $C_{\max}$ y tiempo total de flujo para los algoritmos MOSAII$_M$ y RIPG. (a) para $C_{\max}$ y tiempo total de flujo para el algoritmo RIPG, (b) para el algoritmo MOSAII$_M$. (c) representa la gráfica de Diff-EAF entre (a) y (b). Instancia Ta091 con 200 trabajos y 10 máquinas.

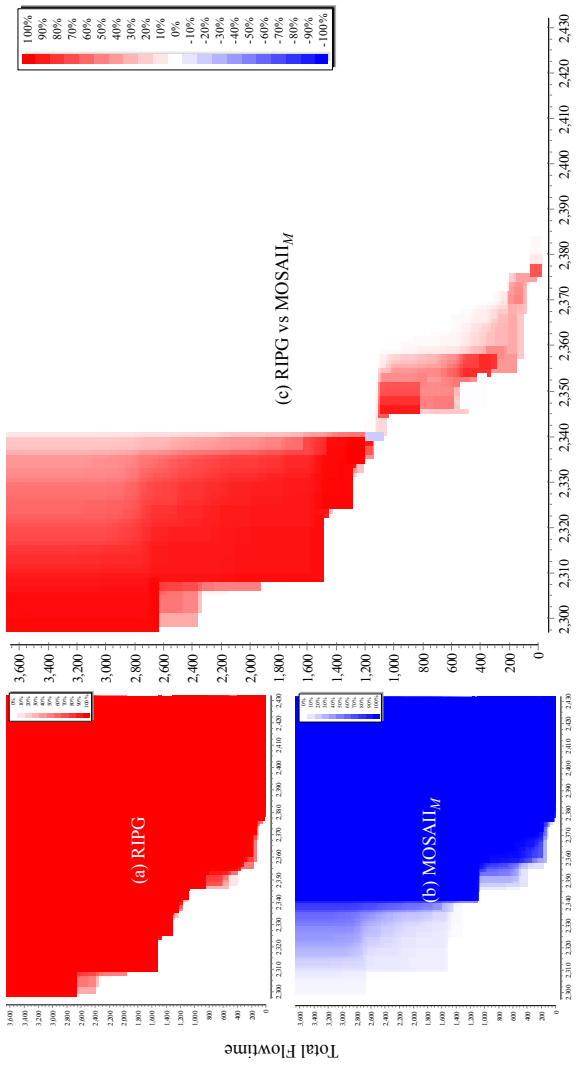


Figura 4.16: Gráfica de Empirical Attainment Function. $t = 100$ para $C_{\max}$ y tiempo total de flujo para los algoritmos MOSAII$_M$ y RIPG. (a) para $C_{\max}$ y tiempo total de flujo para los algoritmo RIPG, (b) para el algoritmo MOSAII$_M$. (c) representa la gráfica de Diff-EAF entre (a)-(b) or Diff-EAF.Instancia Ta021 con 20 trabajos y 20 máquinas.

4.3. Conclusiones del capítulo

En este Capítulo hemos presentado el algoritmo *Restarted Iterated Pareto Greedy* para resolver problemas de taller de flujo de permutación multiobjetivo. Este algoritmo, está basado en el algoritmo voraz iterativo de Ruiz y Stützle (2007), aunque presenta grandes diferencias en cuanto a su estructura y funcionamiento. La diferencia más importante es la utilización de un conjunto de soluciones no dominadas en vez de una única solución. También la fase voraz utiliza y evalúa un conjunto de soluciones parciales no dominadas. Se ha desarrollado un operador de selección llamado *Modified Crowding Distance Assignment* que permite aumentar la diversidad y distribución de las soluciones. La búsqueda local propuesta varía de forma dinámica el vecindario de búsqueda permitiendo la ejecución de búsquedas más exhaustivas cuando el algoritmo ha avanzado mucho y es más difícil encontrar mejores fronteras. Además, se ha añadido un sistema de reinicio que permite al algoritmo aumentar su capacidad de búsqueda librándolo de los óptimos locales en las instancias pequeñas y medianas.

Hemos llevado a cabo un profundo análisis estadístico y computacional de cada una de sus fases con el objetivo de poner en relevancia la importancia que tiene el enfoque científico en el diseño de un algoritmo. Hemos comparado el algoritmo RIPG con el estado del arte de la literatura de taller de flujo multiobjetivo. Para ello hemos elegido los tres algoritmos que obtuvieron mejor rendimiento según nuestros experimentos del Capítulo 3. También hemos añadido otros dos algoritmos que aparecieron en la literatura luego de que publicáramos los resultados de ese capítulo.

Para los experimentos computacionales empleamos los mismos indicadores de calidad que utilizamos en la comparativa del Capítulo 3, el indicador de hipervolumen en combinación con el indicador épsilon. Para refinar la comparación utilizamos también las Diff-EAF entre los dos algoritmos con mejor rendimiento, de manera tal, que se pueda observar claramente las diferencias

de calidad de resultados de los mismos. En todas las pruebas el algoritmo propuesto RIPG ha mostrado un rendimiento y calidad de soluciones superior a los demás algoritmos comparados y en muchos casos, por un amplio margen.

En los siguientes capítulos aplicaremos el algoritmo RIPG a problemas de taller de flujo más realistas y por lo tanto más complejos. Como por ejemplo el problema de taller de flujo con tiempos de cambios dependientes de la secuencia (SDST) o el taller de flujo híbrido. En cada caso realizaremos las adaptaciones necesarias al algoritmo RIPG para los tipos de problemas propuestos y realizaremos una batería de pruebas comparándolo con los mejores algoritmos encontrados en la literatura para cada tipo de problema.

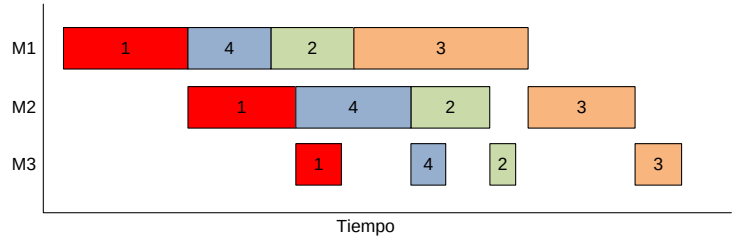
CAPÍTULO 5

EL TALLER DE FLUJO DE PERMUTACIÓN MULTI-OBJETIVO CON TIEMPOS DE CAMBIO

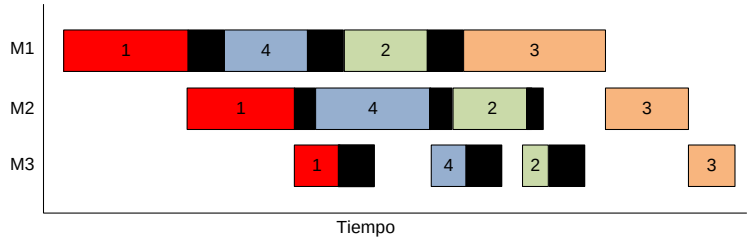
El problema de taller de flujo de permutación ha sido muy criticado debido a que en muchas ocasiones no es aplicable a la realidad de las organizaciones productivas. De acuerdo a la idiosincrasia de cada industria, ciertas características son desestimables o no, y en muchos casos, un taller de flujo de permutación es una simplificación aceptable del problema a resolver, mientras que en otras ocasiones es necesario tener en cuenta alguna característica particular del problema para llegar a resultados aceptables, incluso cuando se habla del taller de flujo multi-objetivo. Una de las características comunes en muchas de estas organizaciones es la existencia de tiempos de cambio.

El tiempo de cambio es un periodo de tiempo no productivo que transcurre entre dos tareas productivas consecutivas y que se debe, en general, a adaptaciones que deben realizarse en los recursos productivos cuando se varía el tipo de producto a fabricar. Por ejemplo en las empresas de fabricación de cerámicas al cambiar de un artículo (por ejemplo un azulejo) de un color

oscuro a otro artículo de color más claro, es importante la limpieza profunda de todas las máquinas para evitar las mezclas de colores. Entonces, el tiempo ocupado para la limpieza de la maquinaria será el tiempo de cambio. Los procesos de tiempo de cambio pueden incluir también cambios de piezas o ajustes en las máquinas, limpieza de parte o toda la máquina, reemplazo de piezas que se gastan, etc. Si bien a veces se puede incluir el tiempo de cambio como parte de la tarea productiva, por lo general no es posible ignorarlo.



(a) Sin tiempos de cambio.



(b) Con tiempos de cambio.

Figura 5.1: En la figura superior (a) un programa de producción sin tiempos de cambio. En la figura inferior (b) puede verse como afectan los tiempos de cambio al mismo programa de producción.

Se pueden categorizar los tiempos de cambio en dos grandes grupos de

acuerdo a su dependencia o no con el estado actual de la producción. En primer lugar esta el tiempo de cambio no dependiente de la secuencia (*Sequence Independent Setup Time* o SIST). En este caso, el tiempo de cambio solamente depende del producto que está a punto de comenzar a producirse. Mientras que el tiempo de cambio dependiente de la secuencia (*Sequence Dependent Setup Time* o SDST) tiene en cuenta tanto el producto que está por comenzar a producirse como el producto que está actualmente en producción. Esta segunda categorización de los tiempos de cambio es mucho más compleja que la primera y además de ello la incluye como un caso particular.
Por otro lado, los tiempos de cambio pueden ser anticipatorios o no anticipatorios. En el primer caso la configuración de la máquina se puede llevar a cabo en cuanto ésta queda disponible y antes que el siguiente trabajo en la secuencia de fabricación esté disponible para comenzar su procesamiento. En este enfoque el tiempo de cambio dependiente de la secuencia en la máquina i al procesar el trabajo k luego de haber procesado el trabajo j se describe como $S_{ijk}, \forall i \in M, \forall j, k \in N, j \neq k$.

Siguiendo el esquema de notación propuesto por Graham et al. (1979) y su posterior extensión para problemas multi-objetivo propuesta por T'Kindt y Billaut (2002), el problema estudiado en este Capítulo se puede representar con la siguiente notación: $F/prmu, S_{ijk}/\#(\gamma_1, \gamma_2)$ donde γ_1 y γ_2 son los dos objetivos considerados en el enfoque de Pareto.

Poco se ha investigado en el campo de flowshop multi-objetivo con tiempos de cambio dependientes de la secuencia de fabricación. La mayor parte de la investigación concerniente a tiempos de cambio se refiere a problemas de un solo objetivo.
Del mismo modo que en otros problemas relativos al taller de flujo, los objetivos más estudiados suelen ser los ya nombrados *makespan* o $C_{\max}$ y tiempo total de flujo o TFT.

Otro objetivo muy común en los problemas de taller de flujo es la tardanza total (*Total Tardiness* o TT). En todo caso, la utilización de este objetivo intenta orientar el resultado hacia la satisfacción del cliente más que hacia la productividad. Sin embargo, debido a que no todos los clientes o pedidos de clientes tendrán la misma relevancia en un entorno industrial real, se puede añadir una pequeña modificación al cálculo de este objetivo, ponderando cada trabajo mediante un multiplicador o ponderación. Esta característica permite otorgar mayor o menor relevancia a cada trabajo que ha de programarse. En este capítulo entonces, para acercarnos más a la realidad de las organizaciones productivas, utilizaremos como criterio de optimización, además de $C_{\text{máx}}$ y tiempo total de flujo, la tardanza total ponderada (*Total Weighted Tardiness* o TWT). Las combinaciones de objetivos que consideraremos a lo largo de este capítulo son $(C_{\text{máx}}, TFT)$ y $(C_{\text{máx}}, TWT)$.

El problema estudiado en este capítulo contiene características que lo hacen aplicable a múltiples configuraciones industriales. Para nuestro conocimiento, hasta la fecha este tipo de problemas no ha sido estudiado. En este capítulo presentamos una adaptación del algoritmo propuesto en el Capítulo 4 al problema del taller de flujo multi-objetivo con tiempos de cambio.

A continuación revisaremos el estado de la literatura en cuanto al estudio del taller de flujo para un único objetivo con tiempos de cambio.

5.1. Revisión bibliográfica del problema de taller de flujo de permutación con tiempos de cambio para un solo objetivo

En comparación con la basta literatura existente para el taller de flujo de permutación, pocas publicaciones estudian la variante de este problema que incluye tiempos de cambio. Más aún, hasta la fecha no hemos encontrado

ninguna referencia para el taller de flujo de permutación multi-objetivo con tiempos de cambio, aunque si se pueden encontrar referencias a otro tipo de problemas con tiempos de cambio, como el taller de flujo híbrido.
Las técnicas exactas propuestas para el taller de flujo de permutación con tiempos de cambios han obtenido resultados muy limitados. La última referencia existente, y a la vez, el estudio más avanzado referente a este problema es el propuesto por Ríos-Mercado y Bard (2003). En él se estudia la estructura poliédrica de un modelo de programación entera mixta para el taller de flujo con tiempos de cambio con la finalidad de generar cortes para ser usados en un esquema de *branch & cut*.
En un trabajo anterior, Ríos-Mercado y Bard (1998) estudiaron la resolución de problemas de taller de flujo con $F/prmu, S_{ijk}/C_{\text{máx}}$ presentando una modificación a la conocida heurística NEH (presentada por Nawaz, Enscore y Ham (1983) para resolver el taller de flujo regular) que tiene en cuenta tiempos de cambio. En el mismo artículo también propusieron un algoritmo GRASP (*Greedy Randomized Adaptative Search*).
Más adelante, los mismos autores, (Ríos-Mercado y Bard, 1999), presentaron una modificación a la heurística propuesta por Simons (1992), creando un nuevo método al que denominaron HYBRID.

Ruiz, Maroto y Alcaraz (2005) propusieron un algoritmo memético junto con un algoritmo genético para el problema $F/prmu, S_{ijk}/C_{\text{máx}}$. En este trabajo llevaron a cabo un profundo estudio experimental comparando los métodos propuestos contra varios métodos originalmente propuestos para el problema $F/prmu/C_{\text{máx}}$.

En el caso del problema $F/prmu, S_{ijk}/\sum_{j=1}^{n} w_j T_j$ poco ha sido propuesto. Parthasarathy y Rajendran (1997b) y Parthasarathy y Rajendran (1997a) presentaron, en ambos casos, una heurística basada en la metodología de recocido simulado (*simulated annealing*) con el objetivo de minimizar la máxima

tardanza ponderada y la tardanza ponderada total en cada caso.

Rajendran y Ziegler (1997) presentaron una nueva heurística e introdujeron un algoritmo conformado por la heurística propuesta y una búsqueda local. En este trabajo estudiaron el objetivo de tiempo de permanencia en planta ponderado. Más adelante Rajendran y Ziegler (2003) propusieron una estructura similar en donde consideraron una combinación de los objetivos tardanza total y tiempo total de permanencia en planta ponderado.
Luego, Ruiz y Stützle (2008) propusieron dos algoritmos voraces iterados (*Iterated Greedy* o IG) para el problema del taller de flujo de permutación con tiempos de cambio dependientes de la secuencia. El primero siguió los lineamientos del IG original planteado por Ruiz y Stützle (2007), mientras que el segundo algoritmo incorporó una búsqueda local simple.

Podemos encontrar en la literatura algunas revisiones bibliográficas para el problema del taller de flujo de permutación con tiempos de cambio. Entre ellas la de Ruiz, Maroto y Alcaraz (2005), quienes realizan una profunda y detallada revisión de literatura, que luego es actualizada y extendida por Allahverdi et al. (2008).

Debido a que, hasta la actualidad no hemos encontrado algoritmos para resolver específicamente el problema de taller de flujo multi-objetivo con tiempos de cambio, adaptaremos los algoritmos implementados y probados en el Capítulo 3 a este problema, para realizar una comparativa extensa y profunda. En las siguientes secciones presentaremos un estudio comparativo de los algoritmos antes citados, adaptados al problema de taller de flujo con tiempos de cambio dependientes de la secuencia, describiendo la metodología, el banco de pruebas utilizado y los resultados obtenidos.

5.2. Fase experimental

Para evitar evaluar nuevamente todos los algoritmos ya evaluados en el Capítulo 3 hemos tomado los 10 algoritmos que mostraron tener mejor rendimiento. Siete de los algoritmos finalmente seleccionados fueron específicamente diseñados para resolver problemas de taller de flujo multi-objetivo sin tiempos de cambio, mientras que los tres restantes son algoritmos multi-objetivo de propósito general.

Del mismo modo que hicimos en el Capítulo 4, hemos añadido a la evaluación varios algoritmos propuestos luego de la publicación de nuestro artículo Minella, Ruiz y Ciavotta (2008a) y también una adaptación del algoritmo MOSAII, propuesto en Varadharajan y Rajendran (2005) ya nombrado en el Capítulo 4, al que denominamos MOSAII$_M$. También hemos añadido otros dos algoritmos ya nombrados en capítulos anteriores. Uno de ellos es el algoritmo genético propuesto por Yandra y Tamura (2007) y el otro es el algoritmo voraz iterado (IG) adaptado al entorno multi-objetivo propuesto en Framiñan y Leisten (2008).
Además, hemos añadido a esta comparación cuatro algoritmos desarrollados por Rajendran y Ziegler (2009). Los autores proponen una serie de 20 variaciones sobre un algoritmo de colonia de hormigas. En cada caso se permutan, añaden o modifican los parámetros y búsquedas locales. De estas 20 variaciones, hemos seleccionado cuatro de ellas que han mostrado ser muy competitivas.
Finalmente, hemos adaptado el algoritmo RIPG propuesto en el Capítulo 4 para el problema del taller de flujo multi-objetivo con tiempos de cambio y lo hemos añadido a la comparación. En resumen hemos evaluado dieciocho algoritmos en este capítulo, algunos de ellos específicamente propuestos para el PFSP multi-objetivo y otros algoritmos multi-objetivo de uso general. La relación de estos algoritmos se resume en la Tabla 5.1.

Algoritmo	Año	Autor/es	Tipo
MOGA_Murata	1996	Murata, Ishibuchi y Tanaka	Algoritmo genético. Específico
PESA	2000	Corne, Knowles y Oates	Algoritmo genético. General
PESAII	2001	Corne et al.	Algoritmo genético. General
CMOGA	2001	Murata, Ishibuchi y Gen	Algoritmo genético. Específico
MOTS	2004	Armentano y Arroyo	Búsqueda tabú. Específico
$\varepsilon-$NSGAII	2005	Kollat y Reed	Algoritmo genético. General
MOGALS	2005	Arroyo y Armentano	Algoritmo genético. Específico
MOSAII	2005	Varadharajan y Rajendran	Recocido simulado. Específico
PGA_ALS	2006	Pasupathy, Rajendran y Suresh	Algoritmo genético. Específico
PILS	2007	Geiger	Búsqueda local iterada. Específico
hMGA	2007	Yandra y Tamura	Algoritmo genético. Específico
MOIGS	2008	Framiñan y Leisten	Algoritmo voraz iterado. Específico
MOACA17_M	2009	Rajendran y Ziegler	Algoritmo de colonia de hormigas. Específico
MOACA18_M	2009	Rajendran y Ziegler	Algoritmo de colonia de hormigas. Específico
MOACA19_M	2009	Rajendran y Ziegler	Algoritmo de colonia de hormigas. Específico
MOACA20_M	2009	Rajendran y Ziegler	Algoritmo de colonia de hormigas. Específico
MOSAII_M	2011	Minella, Ruiz y Ciavotta	Versión mejorada de MOSAII
RIPG	2011	Minella, Ruiz y Ciavotta	Algoritmo voraz iterado. Específico

Tabla 5.1: Métodos adaptados al problema del taller de flujo multi-objetivo con tiempos de cambio.

5.2.1. Descripción del banco de pruebas

Para los experimentos de este capítulo utilizamos tres diferentes conjuntos de instancias, todos ellos basados en las instancias propuestas en Taillard (1993) (comúnmente conocido como instancias de Taillard). Estas instancias fueron originalmente propuestas para el taller de flujo normal (sin tiempos de cambio) y luego adaptadas para el problema PFSP con tiempos de cambio en los trabajos de Ruiz, Maroto y Alcaraz (2005) y Ruiz y Stützle (2008). Cada conjunto contiene instancias con varias combinaciones de trabajos (n) y máquinas (m). Las diferentes combinaciones de $n \times m$ son $\{20, 50, 100\} \times \{5, 10, 20\}$ y $200 \times \{10, 20\}$ de lo que se obtiene un total de 11 diferentes grupos de instancias.

Los primeros dos conjuntos de instancias, a los que nos referiremos como SSD50 y SSD125 respectivamente, contienen 10 instancias para cada combinación de n (número de trabajos) y m (cantidad de máquinas), resultando en un total de 110 instancias. Los tiempos de cambio en SSD50 y SSD125 han sido generados de manera tal que representen el $50\,\%$ y el $125\,\%$ del tiempo de proceso (p_{ij}) respectivamente. Esto significa que, como los tiempos de proceso p_{ij} en las instancias de Taillard son generados utilizando una distribución uniforme en el rango $[1, 99]$, en el primer conjunto de instancias los tiempos de cambio están distribuidos uniformemente en el rango $[1, 49]$. Mientras que en el segundo conjunto el rango de distribución de los tiempos de cambio es $[1, 124]$.

Además, cada conjunto de instancias tiene asignado unas ponderaciones (*weights*) y fechas de entrega (*due dates*). Las ponderaciones tienen una distribución uniforme $U[1, 10]$, donde los valores mayores representan una mayor importancia o ponderación. Los valores de las fechas de entrega, por otro lado, son generados mediante la expresión: $d_j = P_j \times (1 + random \cdot 3)$ donde $P_j = \sum_{i=1}^{m} p_{ij}$ es la suma del tiempo de proceso en todas las máquinas para los trabajos $j \in N$ y $random$ es un número aleatorio uniformemente distribuido en el rango $[0, 1]$.

Finalmente hemos creado de cero el tercer conjunto de instancias, utilizando la misma estructura usada por los otros dos conjuntos anteriormente citados. Este conjunto es usado para la calibración de nuestro algoritmo y contiene solamente cuatro instancias en cada grupo, la mitad de ellas tienen tiempos de cambio en $[1, 49]$ y la otra mitad en $[1, 124]$. Llamaremos a este conjunto de instancias SSDTest.

5.2.2. Calibración del algoritmo RIPG

El algoritmo RIPG es el resultado de un exhaustivo proceso de ingeniería, en el cual, todas las partes que constituyen el algoritmo han sido probadas y comparadas contra varias alternativas posibles. La finalidad de estas pruebas es crear un algoritmo eficiente, efectivo y adaptado al problema a resolver. Para ello hemos utilizado la misma metodología utilizada ya en el Capítulo 4 denominada diseño de experimentos (DoE). El diseño de experimentos básicamente consiste en la ejecución de varias pruebas con diferentes configuraciones del algoritmo, utilizando un conjunto de instancias reducido y representativo, aunque diferente a las instancias con las que se harán las pruebas finales. De esta manera, se puede determinar el impacto de cada configuración en la solución final y al mismo tiempo se determina la configuración mejor adaptada para el problema en cuestión. Para llevar a cabo la calibración hemos utilizado el conjunto de instancias SSDTest nombrado anteriormente.

Inicialmente estudiamos la influencia de la búsqueda local y el mecanismo de reinicio en la calidad de las soluciones producidas por el algoritmo. Para ello, al igual que hicimos en la Sección 4.2.1 del Capítulo 4, creamos 4 algoritmos llamados IPG_B, IPG_B+LS, IPG_B+R e IPG_B+R+LS, respectivamente. Donde B, LS y R significan *Basic* (Básico), *Local Search* (búsqueda local) y *Restart* (reinicio), respectivamente. Ejecutamos cada algoritmo diez veces (réplicas) contra las 48 instancias que componen el conjunto SSDTest. De cada una de esas ejecuciones hemos extraído información para calcular tanto el I_H

como el I_ε, obteniendo un total de $4 \times 44 \times 10$ = 1760 aproximaciones a las fronteras de Pareto.

Debido a que estas son pruebas reducidas, sólo hemos considerado la combinación de objetivos $(C_{\max}, TWT)$. Además de esto, hemos considerado un único criterio de parada con $t = 100$. Hemos aplicado los test ANOVA a los resultados obtenidos para las variables respuesta I_H y I_ε y el tipo de algoritmo, que representa un factor controlado con cuatro niveles (ya que presentamos cuatro algoritmos, como indicamos anteriormente). Las Figuras 5.2a y 5.2b muestran los gráficos de medias con intervalos de Tukey con un nivel de confianza del 95 % para los tests ANOVA.

En resumen, el resultado de los experimentos ha mostrado que el método con reinicio y búsqueda local IPG_B+R+LS (o simplemente RIPG) es, una vez más, el que mejores resultados obtiene. Estos resultados, son estadísticamente mejores a los resultados de los demás algoritmos incluso tomando el promedio total de resultados. La fase de la búsqueda local mejora los resultados obtenidos en todos los tamaños de instancias, mientras que la fase de reinicio sólo afecta positivamente a los resultados para las instancias pequeñas, sin afectar a las instancias grandes.
En la siguiente sección detallamos los experimentos llevados a cabo y analizamos en profundidad cada uno los resultados obtenidos. Hemos configurado el algoritmo RIPG presentado utilizando los resultados obtenidos en el DoE presentado en esta sección.

5.2.3. Análisis computacional

En esta sección presentamos el detalle de la campaña de experimentos llevados a cabo. Luego analizamos también, utilizando tests estadísticos ANOVA, los resultados obtenidos por cada algoritmo.
En nuestros experimentos ejecutamos los 18 algoritmos probados diez veces

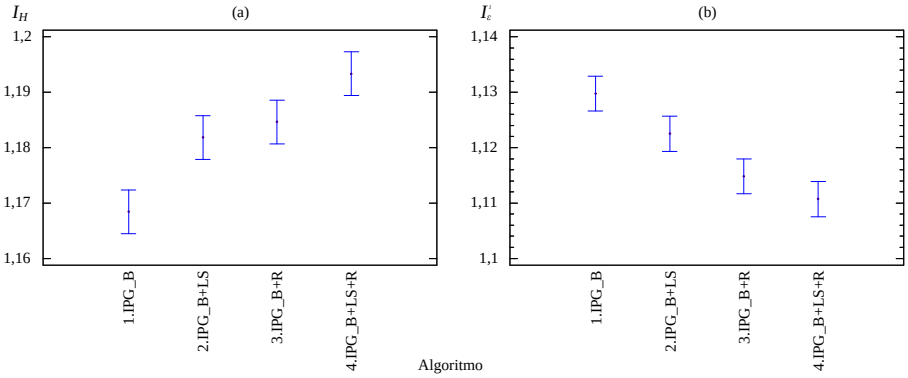


Figura 5.2: Gráfica de medias e intervalos de confianza de Tukey ($\alpha = 0{,}05$) en el test ANOVA para la calibración del algoritmo RIPG. Combinación de criterios $C_{\max}$ y tardanza total ponderada con criterio de parada $t = 100ms$.

(réplicas) para cada una de las 220 instancias presentadas en los conjuntos de instancias SSD50 y SSD125, y cada combinación de objetivos probada. Además realizamos dos ejecuciones con distintos criterios de parada: $t = 150$ms y $t = 200$ms. De estos experimentos hemos obtenido un total de 158.400 fronteras de Pareto.

Todos los métodos que participaron en las pruebas fueron codificados utilizando el lenguaje Delphi 2009 con todas las opciones de optimización activadas. Los experimentos se llevaron a cabo en un conjunto de 20 máquinas virtuales ejecutando el sistema operativo Windows XP, con 2Gb de memoria RAM y 2 procesadores. Estas máquinas virtuales se han ejecutado en un *cluster* de 30 *blade servers* con 16 GB de memoria y dos procesadores Intel XEON E5420 corriendo a 2.5 GHz con 4 núcleos cada uno (8 en total). Cada ejecución de una prueba (instancia, t, combinación de objetivos y número de réplica) ha sido ejecutada en una de las 20 máquinas virtuales seleccionada al azar.

5.2.3.1. Resultados para los experimentos con $C_{\max}$ y tardanza total ponderada

Las Tablas 5.2 y 5.3 resumen los resultados obtenidos en los experimentos con la combinación de objetivos $C_{\max}$ y tardanza total ponderada, para los dos indicadores considerados (I_H y I_ε) y para los grupos de instancias SSD50 y SSD125 respectivamente. Los resultados están ordenados respecto de los valores del indicador I_H de manera descendente. Con lo cual, el mejor algoritmo para I_H queda en la primera posición.

Hemos realizado un test paramétrico ANOVA, así como un test no-paramétrico basado en rangos de Friedman para ambos conjuntos de instancias, ambos indicadores de calidad, los dos pares de objetivos y para los dos criterios de parada probados en los experimentos. En total hemos realizado 16 tests ANOVA multi-factor y otros 16 test de rangos donde el tamaño de la instancia y el algoritmo son los factores controlados. Hemos llevado a cabo la mitad de los experimentos utilizando el valor de hipervolumen como variable respuesta, mientras que para la otra mitad hemos repetido los mismos experimentos pero utilizando el valor de indicador épsilon como variable respuesta. En todas las pruebas hemos utilizando un nivel de confianza ajustado del 95 % ($\alpha = 0{,}05$) igual que hicimos en los Capítulos 3 y 4.
Hemos comprobado las hipótesis principales del test ANOVA en un paso previo, en el que hemos analizado los residuos resultantes del experimento.
Igual que hicimos anteriormente en el Capítulo 4, hemos utilizado tanto test paramétricos como tests no-paramétricos para reforzar la solidez de las conclusiones.
El algoritmo RIPG resulta ser el que obtiene, de forma consistente, los mejores resultados para cada combinación de indicadores, criterio de parada y conjunto de instancias.

La Figura 5.3 muestra los resultados para el primer grupo de instancias

Grupo de instancias SSD50					
t	150ms			200ms	
Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
RIPG	**1,296**	**1,067**	RIPG	**1,313**	**1,057**
MOSAII$_M$	1,272	1,102	MOSAII$_M$	1,282	1,096
MOSAII	1,232	1,127	MOSAII	1,232	1,127
MOIGS	1,186	1,164	MOIGS	1,202	1,150
MOGALS	1,179	1,132	MOGALS	1,189	1,127
MOTS	1,151	1,136	MOTS	1,163	1,130
PESAII	1,106	1,201	PESAII	1,123	1,189
PESA	1,104	1,202	PESA	1,121	1,191
MOACA17$_M$	1,087	1,189	MOACA17$_M$	1,095	1,185
MOACA18$_M$	1,087	1,188	MOACA18$_M$	1,095	1,185
MOACA19$_M$	1,083	1,192	MOACA19$_M$	1,091	1,187
MOACA20$_M$	1,082	1,191	MOACA20$_M$	1,091	1,187
PGA_ALS	1,024	1,250	PGA_ALS	1,034	1,246
MOGA_Murata	0,980	1,276	MOGA_Murata	1,004	1,263
$\varepsilon-$NSGAII	0,969	1,266	$\varepsilon-$NSGAII	0,989	1,255
CMOGA	0,897	1,332	CMOGA	0,930	1,313
hMGA	0,804	1,356	hMGA	0,815	1,348
PILS	0,741	1,441	PILS	0,791	1,401

Tabla 5.2: Resultados para el grupo de instancias SSD50 con la combinación de objetivos $C_{\max}$ y tardanza total ponderada.

con $t = 150$ y para I_ε. En esta ocasión los primeros cinco algoritmos son RIPG, MOSAII$_M$, MOSAII, MOGALS y en quinta posición MOTS. También cabe destacar que la segunda y tercera posición son ocupadas por el algoritmo MOSAII$_M$ y MOSAII, respectivamente, que son básicamente el mismo algoritmo.

La Figura 5.4 muestra los resultados para I_H del primer grupo de instancias con $t = 150$. El orden de los algoritmos ahora cambia, mostrando que las diferencias existentes entre los algoritmos MOIGS, MOGALS y MOTS son

muy pequeñas y dejando estos algoritmos como incomparables para este experimento.

En los resultados para $t = 200$ (Figuras 5.5 y 5.6) para la combinación de objetivos $C_{\max}$ y tardanza total ponderada, el orden de los algoritmos para el primer grupo de instancias varía para I_ε y para I_H.
En este caso, para I_ε el primer lugar es ocupado por el algoritmo RIPG, el segundo por el MOSAII$_M$, el tercero por el MOGALS, el cuarto por MOSAII y el quinto por el algoritmo MOTS, quedando el algoritmo MOIGS desplazado a la sexta posición. Para I_H el orden de los algoritmos es el mismo que para el experimento con $t = 150$, quedando RIPG en primera posición, MOSAII$_M$ y MOSAII en segundo y tercer lugar respectivamente y luego MOIGS en la cuarta y MOGALS ocupando la quinta posición.

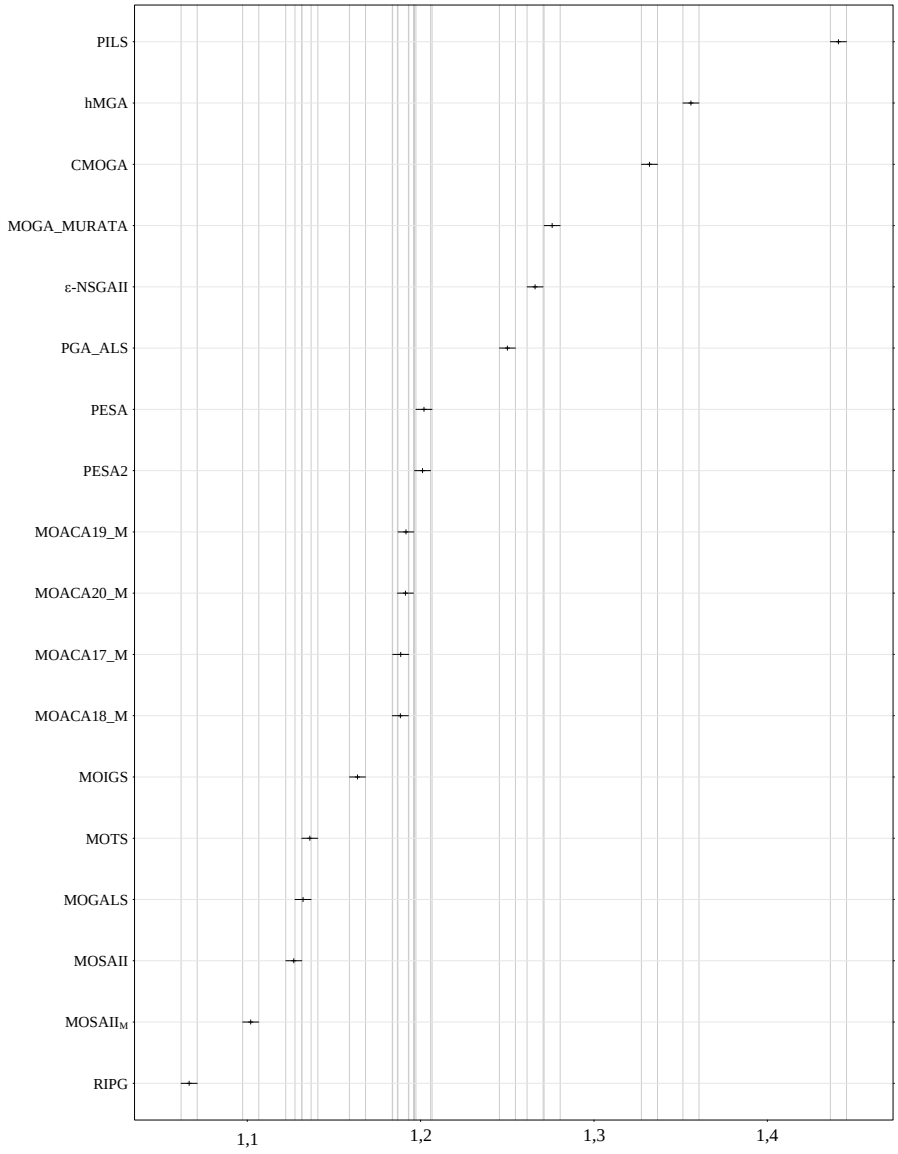


Figura 5.3: Resultados de tests ANOVA para el grupo de instancias SSD50, variable respuesta indicador épsilon y $t = 150$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

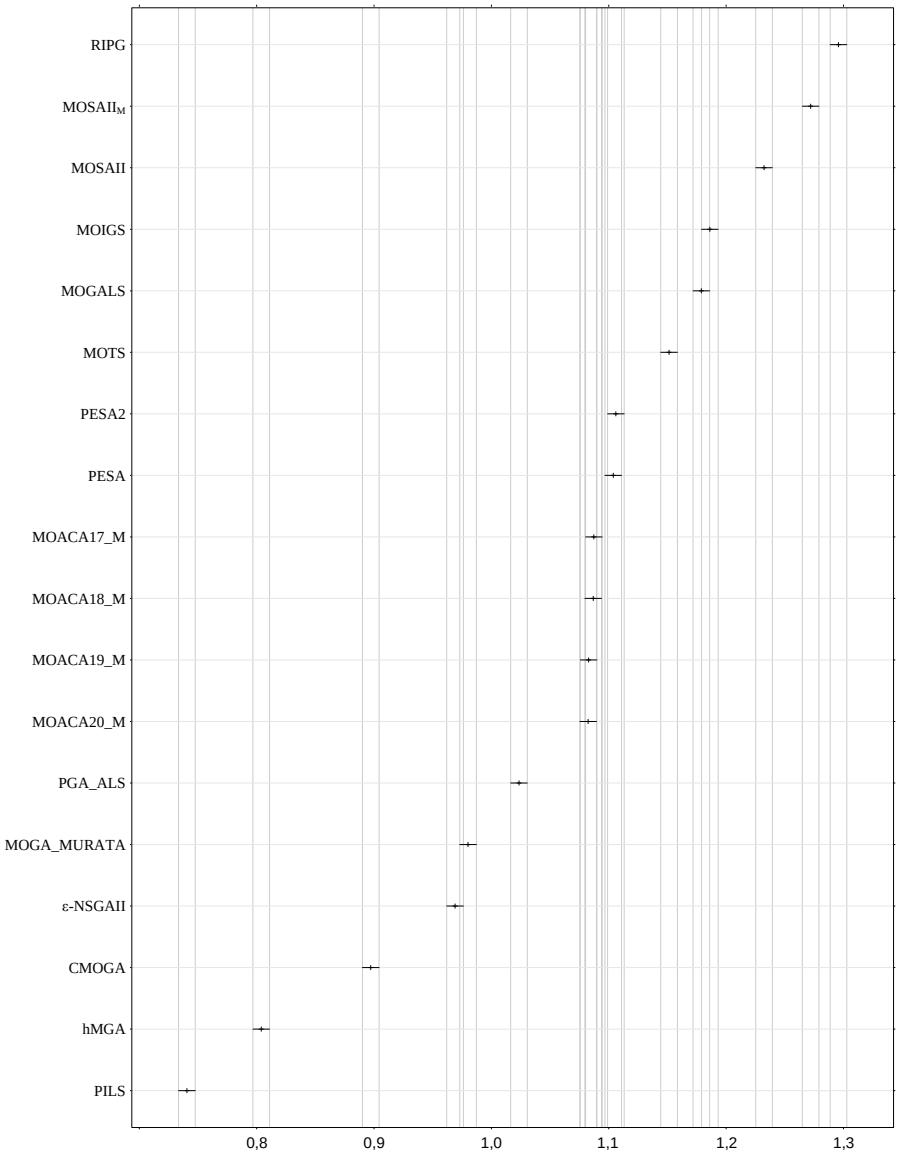


Figura 5.4: Resultados de tests ANOVA para el grupo de instancias SSD50, variable respuesta indicador de hipervolumen y $t = 150$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

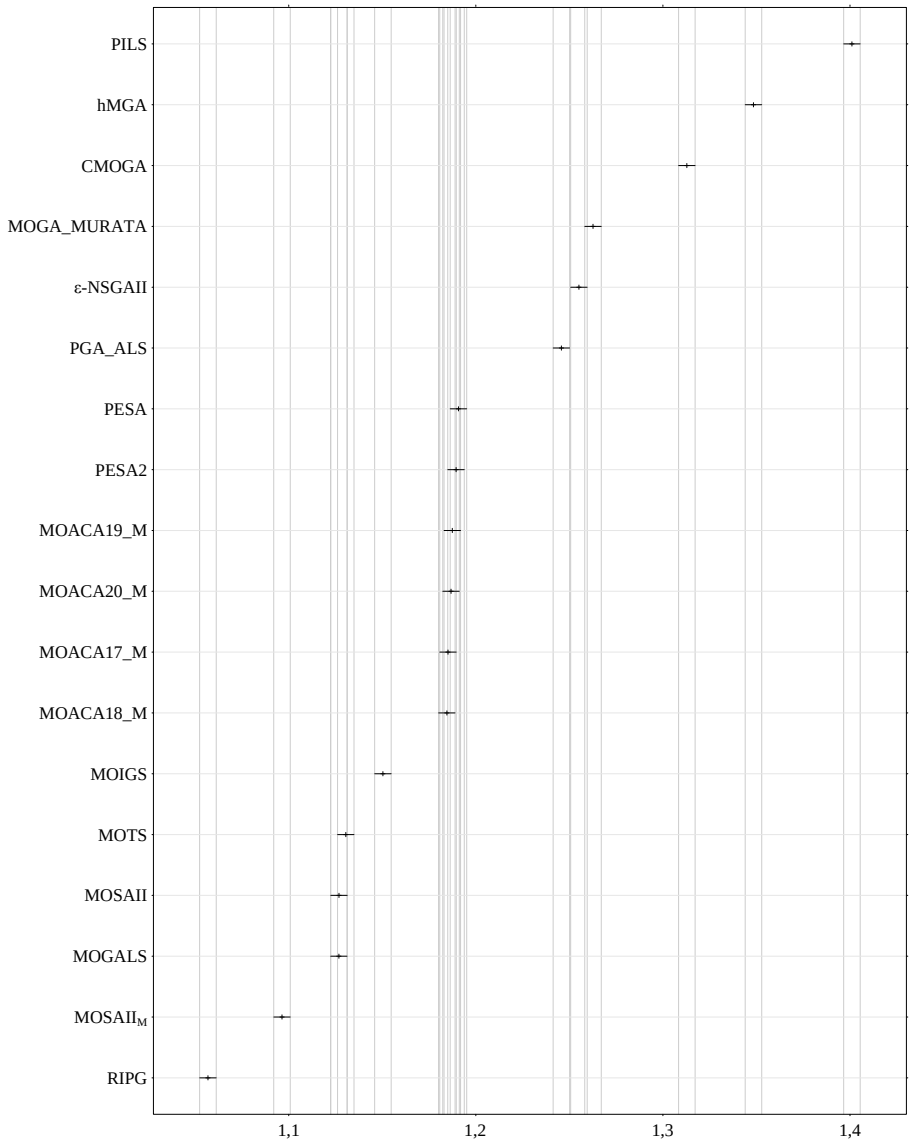


Figura 5.5: Resultados de tests ANOVA para el grupo de instancias SSD50, variable respuesta indicador épsilon y $t = 200$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

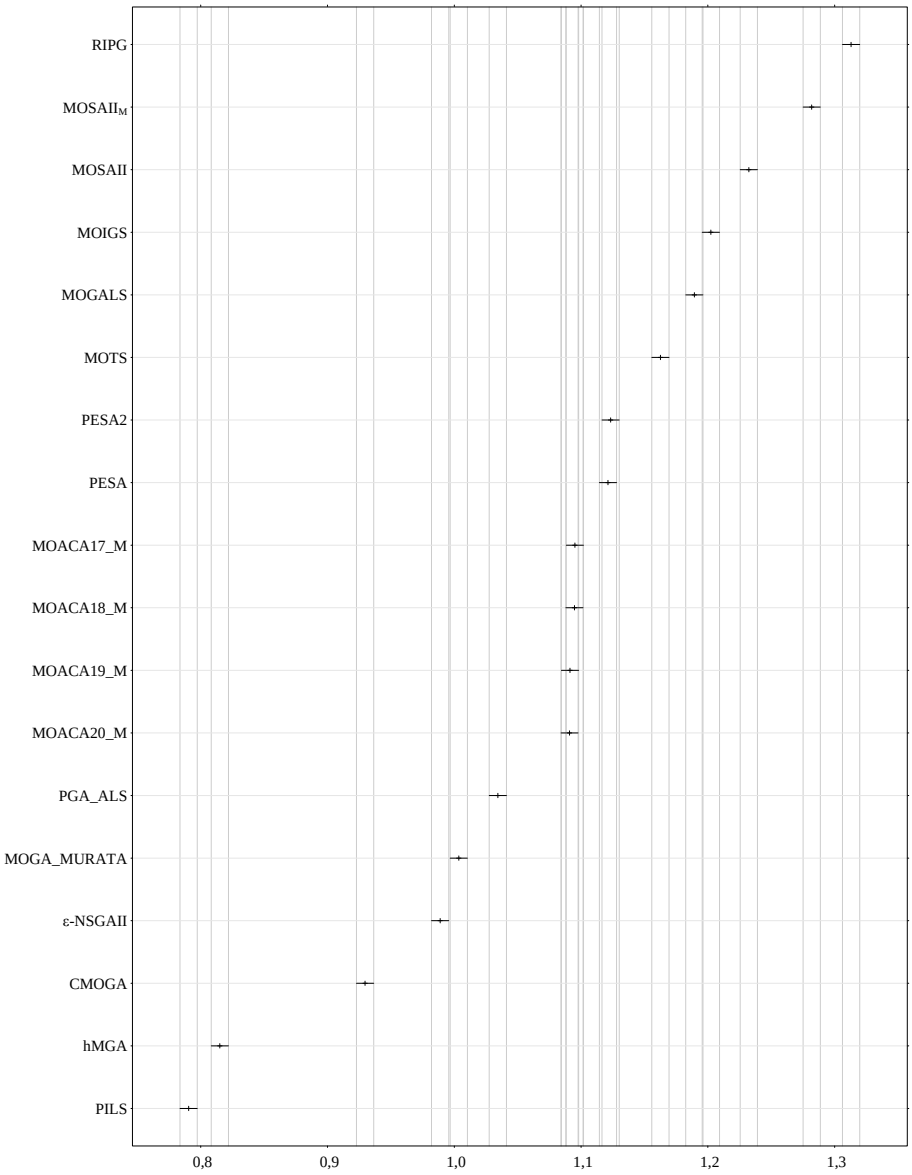


Figura 5.6: Resultados de tests ANOVA para el grupo de instancias SSD50, variable respuesta indicador de hipervolumen y $t = 200$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

Grupo de instancias SSD125					
t	150ms			200ms	
Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
RIPG	**1,318**	**1,064**	RIPG	**1,336**	**1,054**
MOSAII$_M$	1,237	1,146	MOSAII$_M$	1,248	1,140
MOIGS	1,225	1,147	MOIGS	1,241	1,134
MOSAII	1,182	1,178	MOSAII	1,182	1,178
MOGALS	1,154	1,157	MOGALS	1,163	1,153
MOTS	1,126	1,162	MOTS	1,135	1,158
PESA	1,075	1,216	PESA	1,092	1,205
PESAII	1,067	1,221	PESAII	1,086	1,208
MOACA17$_M$	1,060	1,228	MOACA17$_M$	1,065	1,226
MOACA18$_M$	1,059	1,227	MOACA18$_M$	1,065	1,225
MOACA20$_M$	1,056	1,231	MOACA20$_M$	1,063	1,227
MOACA19$_M$	1,056	1,232	MOACA19$_M$	1,062	1,228
PGA_ALS	0,955	1,323	PGA_ALS	0,967	1,318
$\varepsilon-$NSGAII	0,913	1,296	MOGA_Murata	0,934	1,297
MOGA_Murata	0,908	1,313	$\varepsilon-$NSGAII	0,934	1,284
CMOGA	0,810	1,380	CMOGA	0,843	1,360
PILS	0,729	1,448	PILS	0,774	1,410
hMGA	0,699	1,425	hMGA	0,709	1,417

Tabla 5.3: Resultados para el grupo de instancias SSD125 con la combinación de objetivos $C_{\max}$ y tardanza total ponderada.

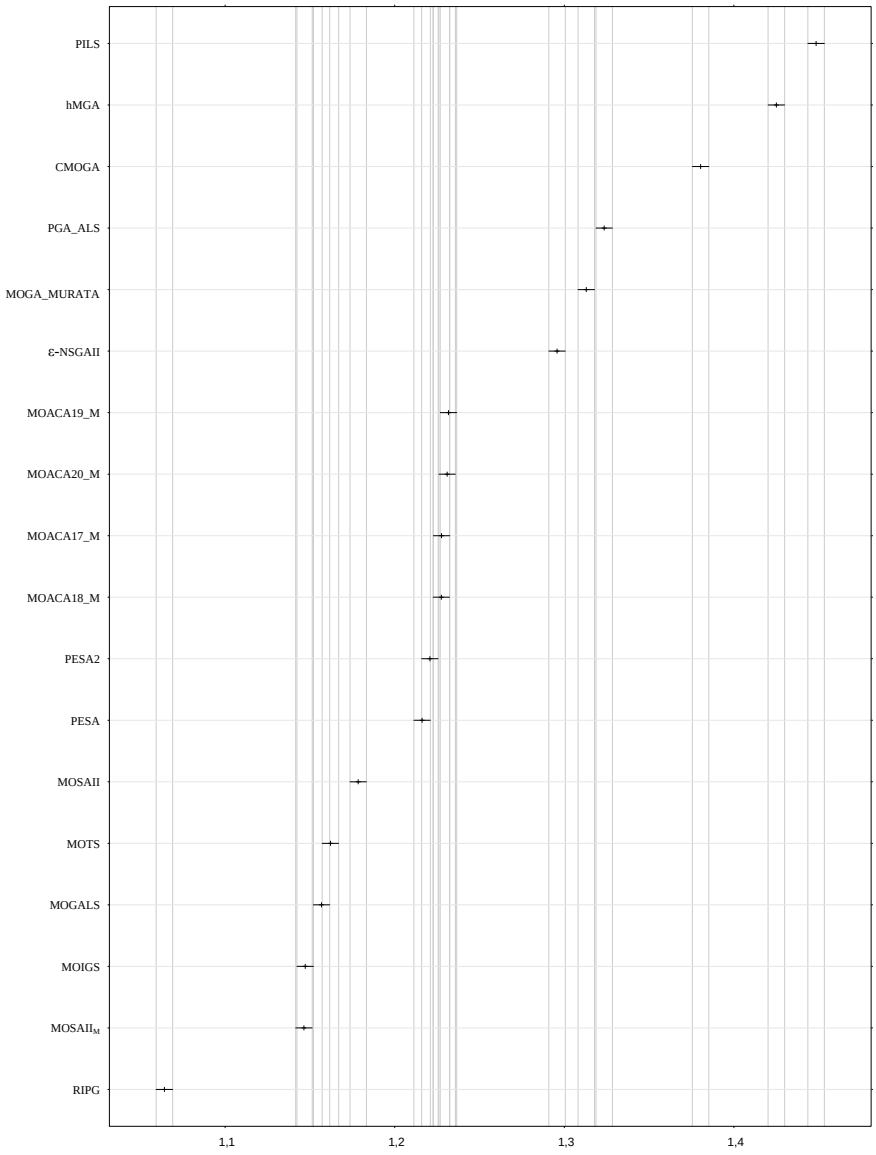


Figura 5.7: Resultados de tests ANOVA para el grupo de instancias SSD125, variable respuesta indicador épsilon y $t = 150$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

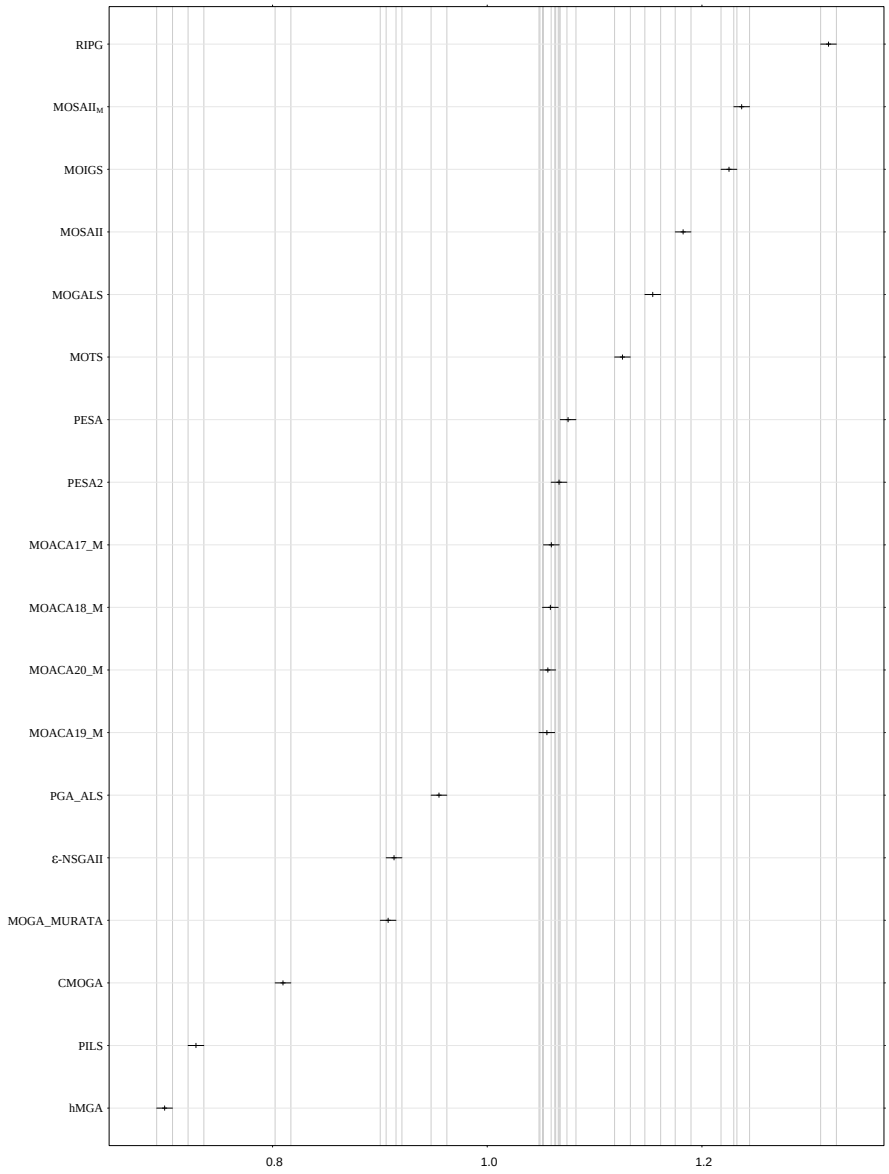


Figura 5.8: Resultados de tests ANOVA para el grupo de instancias SSD125, variable respuesta indicador de hipervolumen y $t = 150$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

En las Figuras 5.7 y 5.8, que presentan los resultados para el grupo SSD125 con $t = 150$ el orden relativo de los algoritmos también cambia como en los experimentos del primer grupo de instancias. En este caso, para I_H, el primer lugar es ocupado por el algoritmo RIPG, seguido por el MOSAII$_M$, luego el MOIGS, en cuarta posición el MOSAII y finalmente en la quinta y sexta posición el MOGALS y MOTS, respectivamente. En el caso del indicador épsilon el orden de los algoritmos se mantiene en las primeras tres posiciones y cambia para la cuarta posición, ocupada por el algoritmo MOGALS, la quinta posición es ocupada por el algoritmo MOTS y el algoritmo MOSAII queda en la sexta posición. Recordemos que el algoritmo MOSAII no se puede detener por tiempo, sino que siempre realiza un número determinado de iteraciones, con lo cual al añadir más tiempo a la ejecución este algoritmo va perdiendo posiciones. Esa es la razón por la hemos creado el algoritmo MOSAII$_M$ partiendo de MOSAII, cuya diferencia es que se puede detener por tiempo. Igual a como ocurría con los experimentos anteriores, el RIPG ha quedado en la primera posición con una diferencia importante sobre el segundo algoritmo.

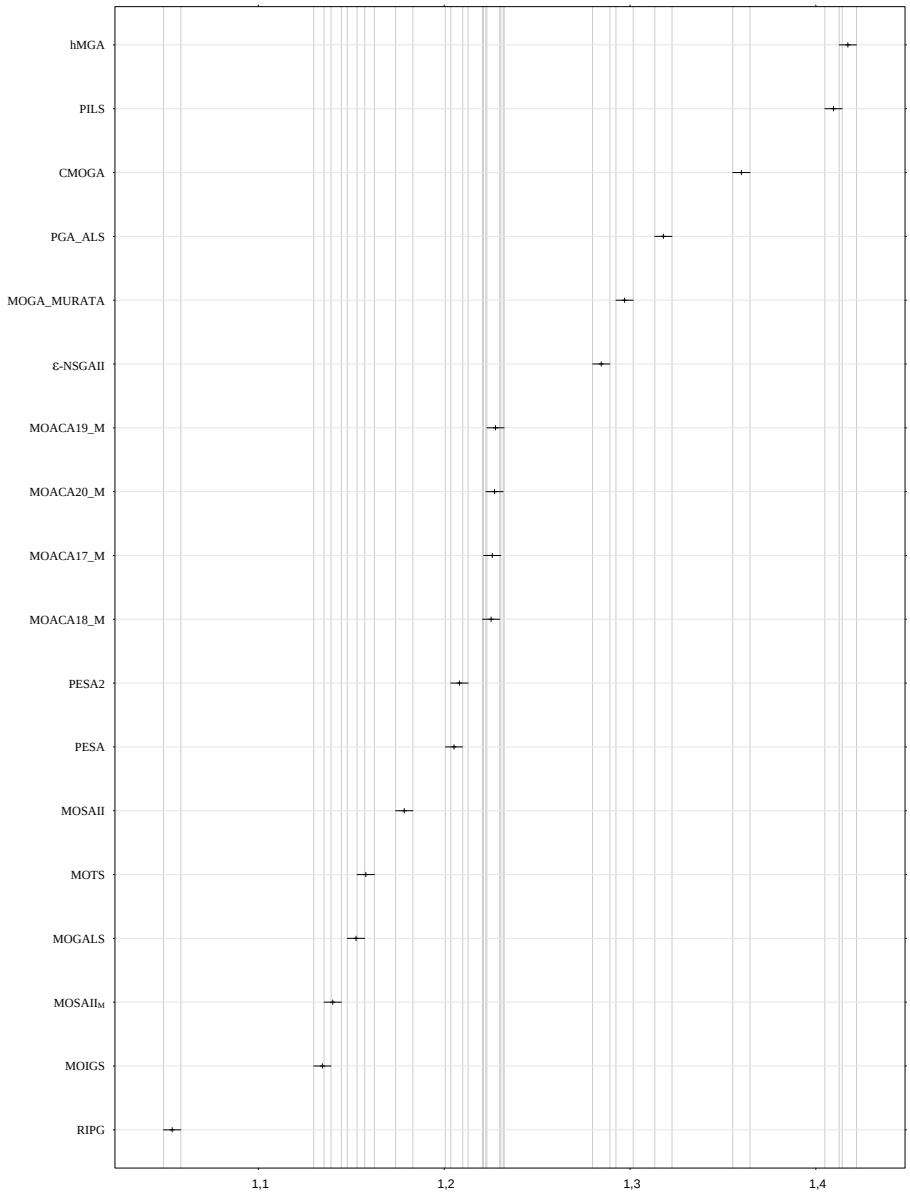


Figura 5.9: Resultados de tests ANOVA para el grupo de instancias SSD125, variable respuesta indicador épsilon y $t = 200$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

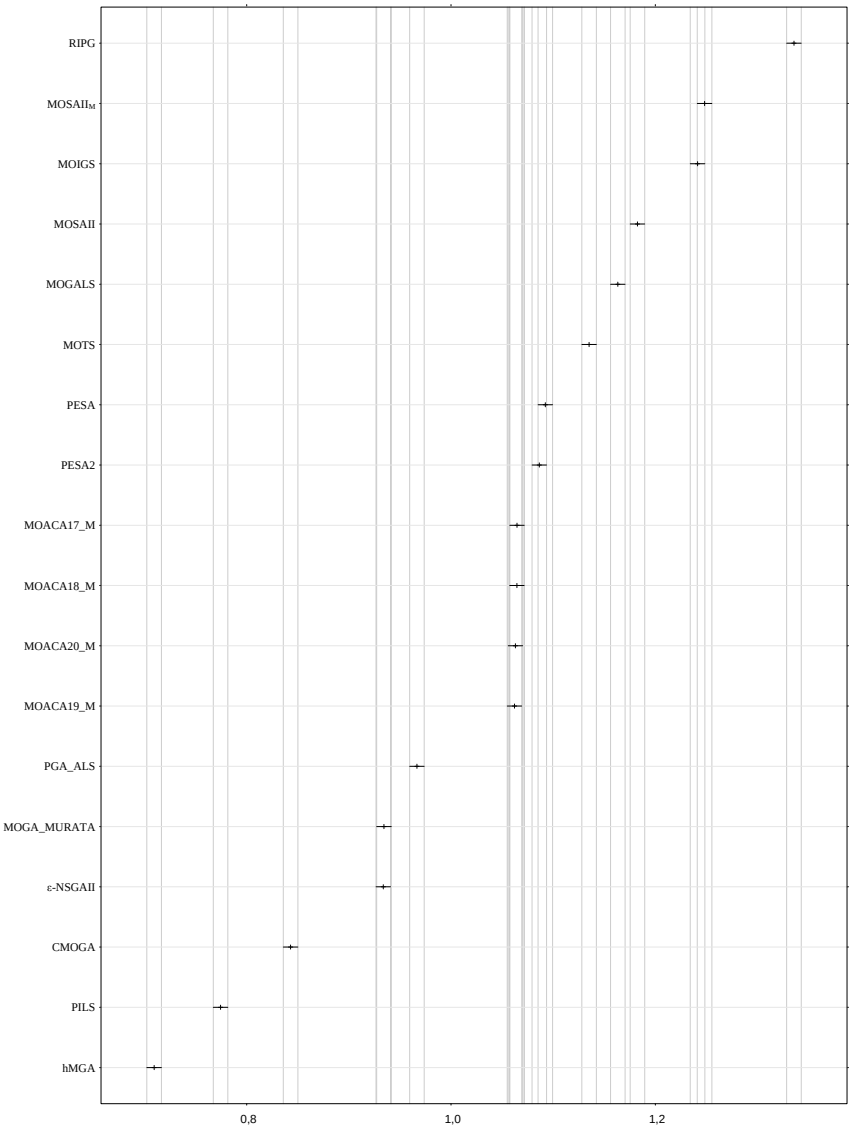


Figura 5.10: Resultados de tests ANOVA para el grupo de instancias SSD125, variable respuesta indicador de hipervolumen y $t = 200$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

En los resultados para $t = 200$ para la combinación de objetivos $C_{\max}$ y tardanza total ponderada, con el segundo grupo de instancias el orden de los algoritmos es el mismo que para $t = 150$ en el caso del indicador I_H. Para I_ε el primer lugar es ocupado por el algoritmo RIPG, el segundo por el MOSAII$_M$, el tercero por el MOIGS, el cuarto por MOGALS y el quinto por el algoritmo MOTS, quedando el algoritmo MOSAII desplazado a la sexta posición una vez más.
Cabe destacar que el ranking global de los algoritmos se mantiene sin cambios sustanciales para los diferentes criterios de parada y los diferentes conjuntos de instancias. Sin embargo, pequeñas diferencias aparecen si se compara el ranking obtenido para I_H y para I_ε. En estos casos, como ya hemos nombrado anteriormente, se considera que los algoritmos son incomparables.

5.2.3.2. Resultados para los experimentos con $C_{\max}$ y tiempo total de flujo

Las Tablas 5.4 y 5.5 muestran los resultados concernientes a la combinación de objetivos $C_{\max}$ y tiempo total de flujo. Nuevamente el algoritmo RIPG obtiene, de manera consistente, los mejores resultados. Mientras que los algoritmos MOSAII$_M$, MOIGS, MOGALS, MOTS y MOSAII ocupan las posiciones siguientes, de la segunda a la sexta en todos los experimentos.

Grupo de instancias SSD50					
t	150ms			200ms	
Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
RIPG	**1,314**	**1,062**	RIPG	**1,333**	**1,053**
MOSAII$_M$	1,217	1,138	MOSAII$_M$	1,232	1,133
MOIGS	1,197	1,121	MOIGS	1,219	1,109
MOSAII	1,151	1,170	MOSAII	1,151	1,170
MOGALS	1,131	1,166	MOGALS	1,143	1,159
MOTS	1,093	1,188	MOTS	1,108	1,181
PESA	1,023	1,212	PESA	1,044	1,201
PESAII	1,002	1,222	PESAII	1,023	1,211
MOACA18$_M$	0,995	1,253	MOACA18$_M$	0,999	1,249
MOACA17$_M$	0,992	1,254	MOACA17$_M$	0,998	1,249
MOACA20$_M$	0,990	1,255	MOACA20$_M$	0,996	1,251
MOACA19$_M$	0,989	1,256	MOACA19$_M$	0,993	1,253
PGA_ALS	0,921	1,306	PGA_ALS	0,934	1,300
MOGA_Murata	0,821	1,336	MOGA_Murata	0,850	1,319
$\varepsilon-$NSGAII	0,799	1,345	$\varepsilon-$NSGAII	0,827	1,328
CMOGA	0,753	1,374	CMOGA	0,790	1,350
PILS	0,661	1,476	PILS	0,713	1,429
hMGA	0,585	1,505	hMGA	0,598	1,495

Tabla 5.4: Resultados para el conjunto de instancias SSD50 con la combinación de objetivos $C_{\max}$ y tiempo total de flujo.

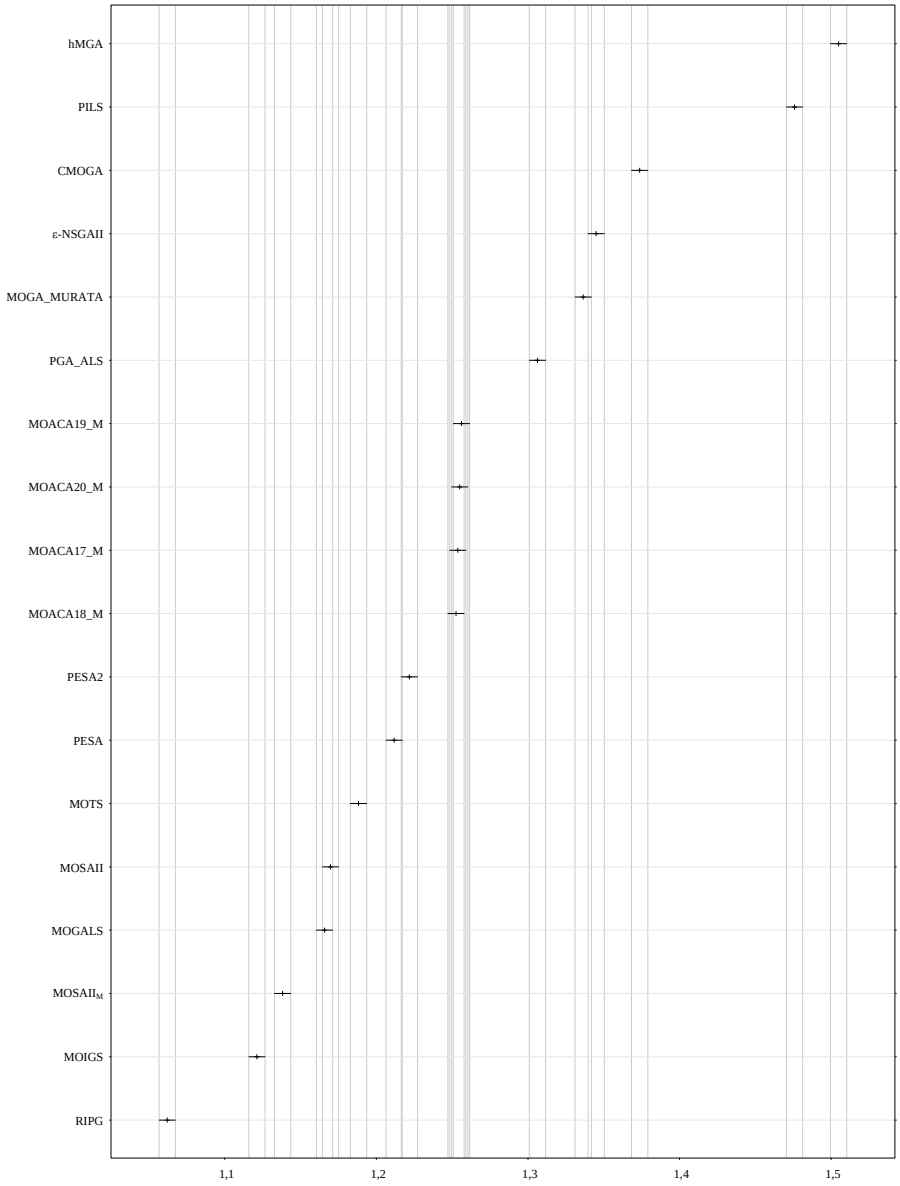


Figura 5.11: Resultados de tests ANOVA para el grupo de instancias SSD50, variable respuesta indicador épsilon y $t = 150$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

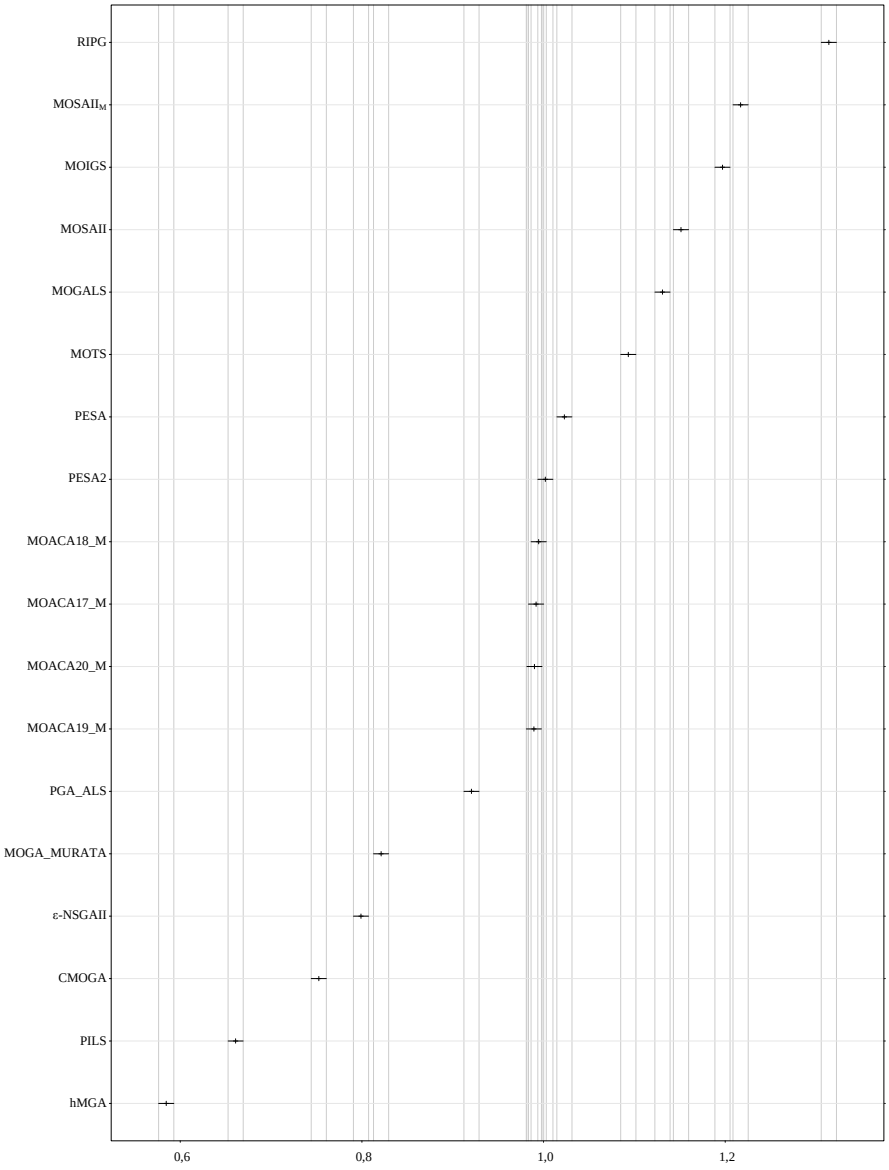


Figura 5.12: Resultados de tests ANOVA para el grupo de instancias SSD50, variable respuesta indicador de hipervolumen y $t = 150$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

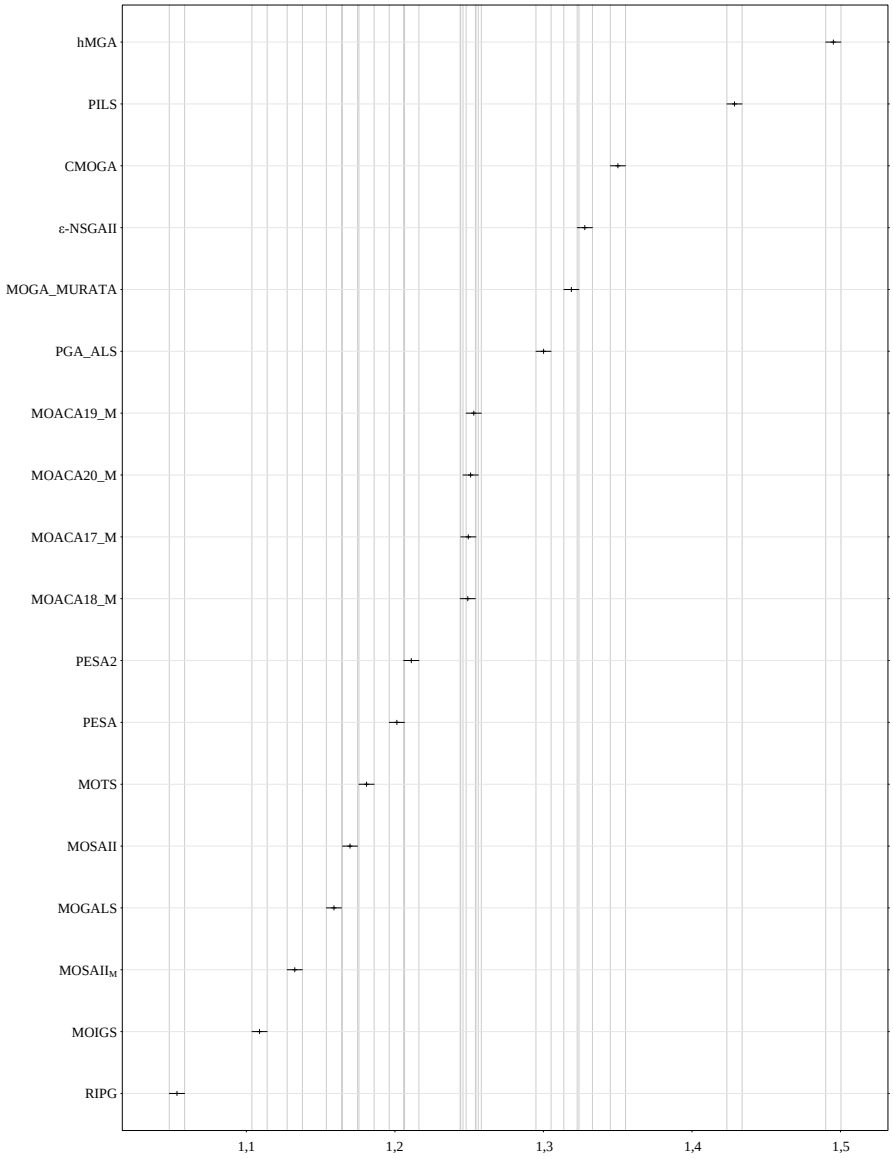


Figura 5.13: Resultados de tests ANOVA para el grupo de instancias SSD50, variable respuesta indicador épsilon y $t = 200$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

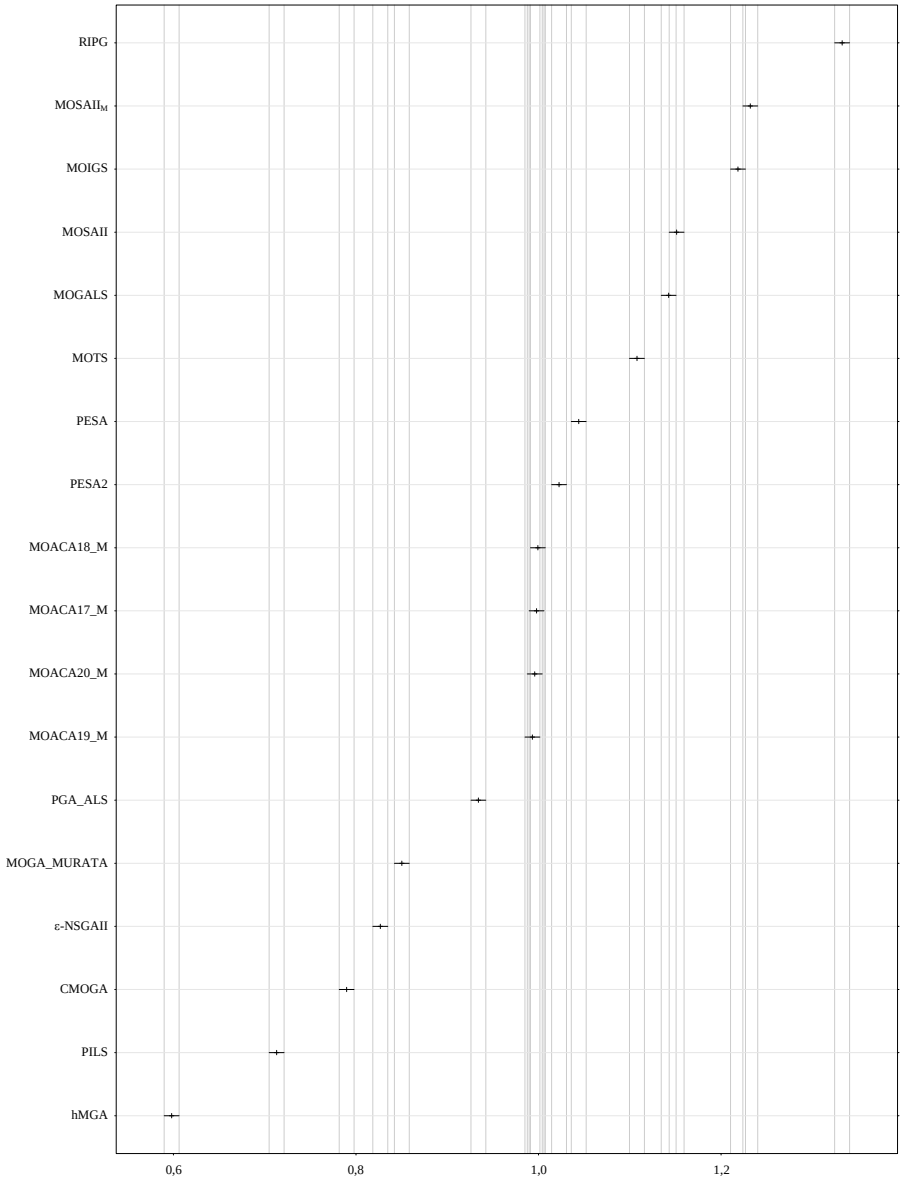


Figura 5.14: Resultados de tests ANOVA para el grupo de instancias SSD50, variable respuesta indicador de hipervolumen y $t = 200$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

En los experimentos para la combinación de objetivos $C_{\max}$ y tiempo total de flujo con criterio de parada $t = 150$ el orden de los algoritmos cambia un poco. Para el indicador I_H el orden de los algoritmos es el siguiente: en primer lugar se encuentra el algoritmo RIPG, luego el algoritmo MOSAII$_M$, en tercer lugar el MOIGS, en cuarto lugar MOSAII, en quinto lugar MOGALS y finalmente en la sexta posición el MOTS.
En el mismo experimento, pero para el indicador I_ε, el orden de los algoritmos no cambia como ocurría con la combinación de objetivos $C_{\max}$ y tardanza total ponderada.

Grupo de instancias SSD125					
t	150ms			200ms	
Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
RIPG	**1,322**	**1,063**	RIPG	**1,339**	**1,055**
MOIGS	1,196	1,124	MOIGS	1,218	1,114
MOSAII$_M$	1,148	1,168	MOSAII$_M$	1,166	1,161
MOSAII	1,067	1,208	MOSAII	1,067	1,208
MOGALS	1,043	1,210	MOGALS	1,056	1,203
MOTS	1,029	1,220	MOTS	1,041	1,213
MOACA18$_M$	0,946	1,280	PESA	0,961	1,248
MOACA17$_M$	0,943	1,281	MOACA18$_M$	0,952	1,277
MOACA20$_M$	0,941	1,283	MOACA17$_M$	0,950	1,277
PESA	0,940	1,259	MOACA20$_M$	0,948	1,279
MOACA19$_M$	0,937	1,284	MOACA19$_M$	0,945	1,281
PESAII	0,916	1,272	PESAII	0,937	1,261
PGA_ALS	0,835	1,349	PGA_ALS	0,847	1,342
$\varepsilon-$NSGAII	0,705	1,400	MOGA_Murata	0,731	1,385
MOGA_Murata	0,702	1,404	$\varepsilon-$NSGAII	0,729	1,385
CMOGA	0,626	1,450	CMOGA	0,663	1,426
PILS	0,607	1,508	PILS	0,650	1,467
hMGA	0,468	1,584	hMGA	0,479	1,575

Tabla 5.5: Resultados para el conjunto de instancias SSD125 con la combinación de objetivos $C_{\max}$ y tiempo total de flujo.

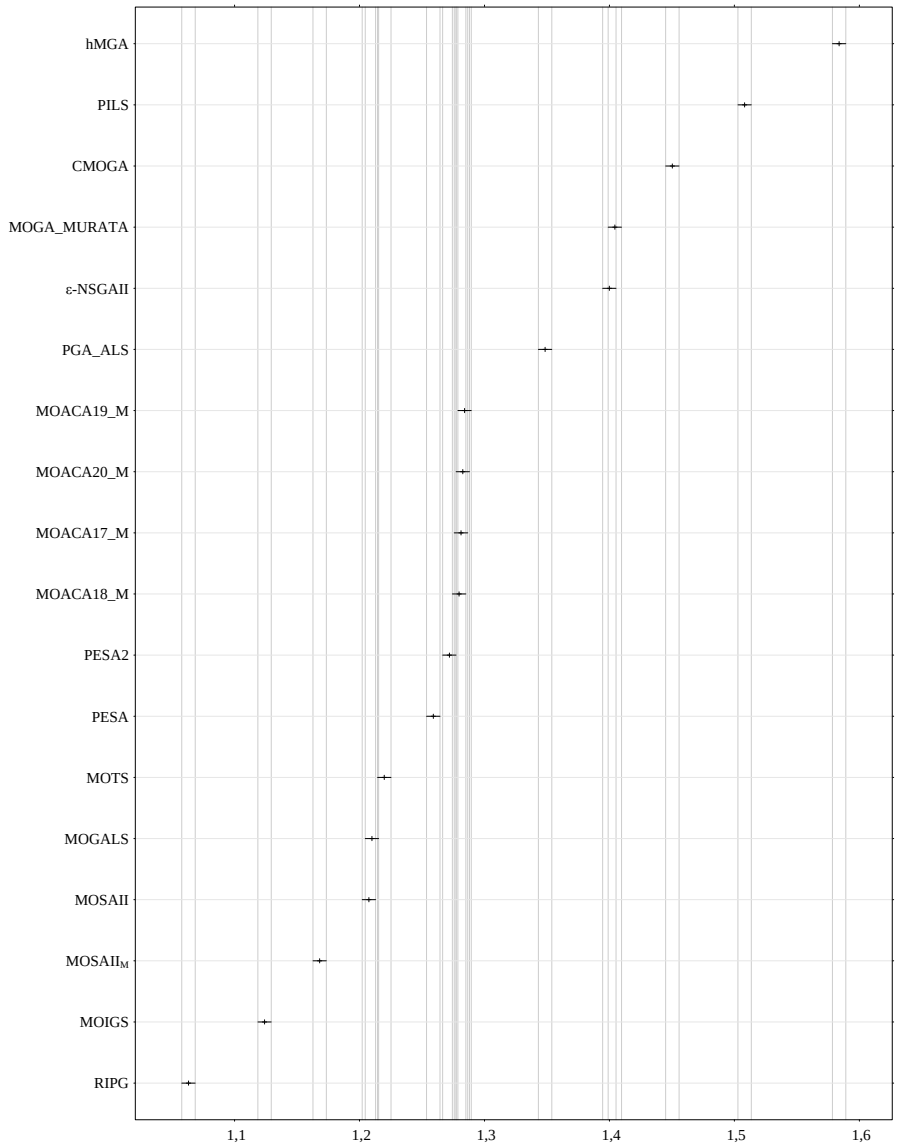


Figura 5.15: Resultados de tests ANOVA para el grupo de instancias SSD125, variable respuesta indicador épsilon y $t = 150$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

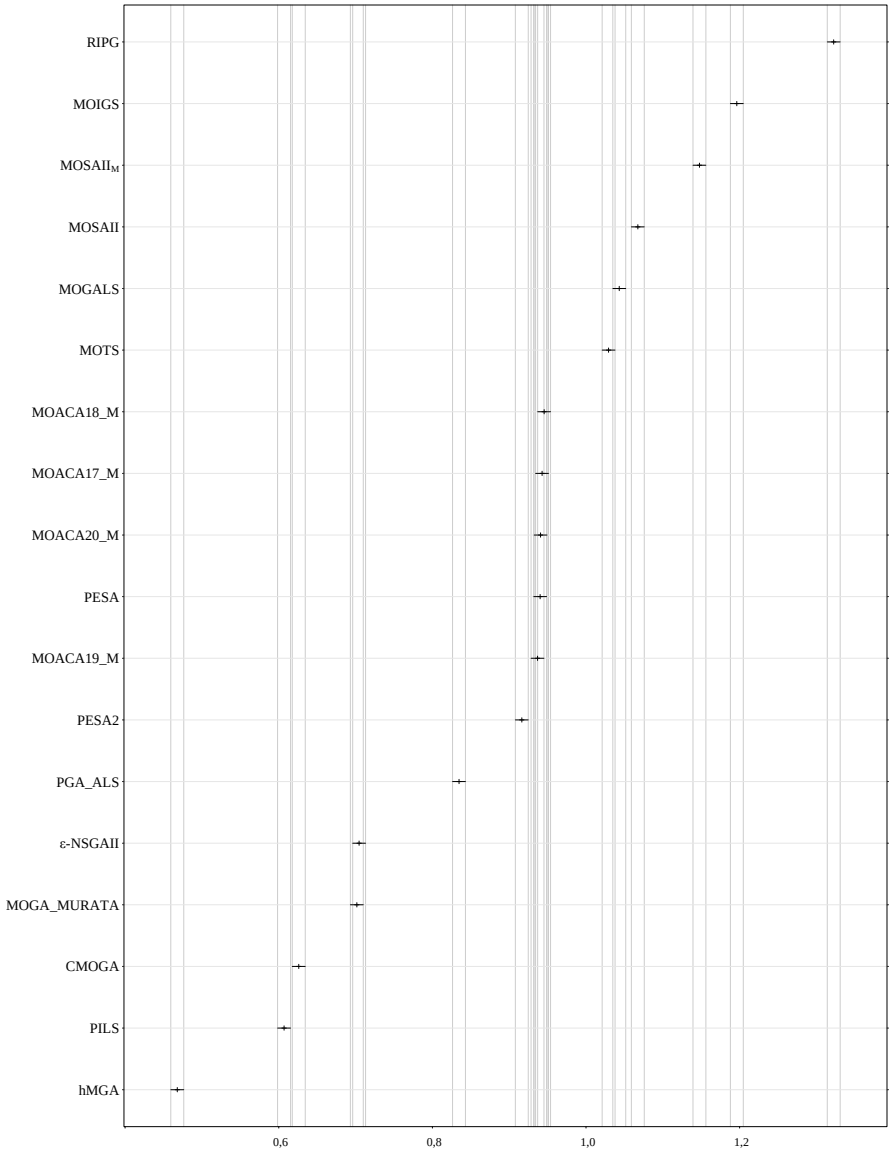


Figura 5.16: Resultados de tests ANOVA para el grupo de instancias SSD125, variable respuesta indicador de hipervolumen y $t = 150$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

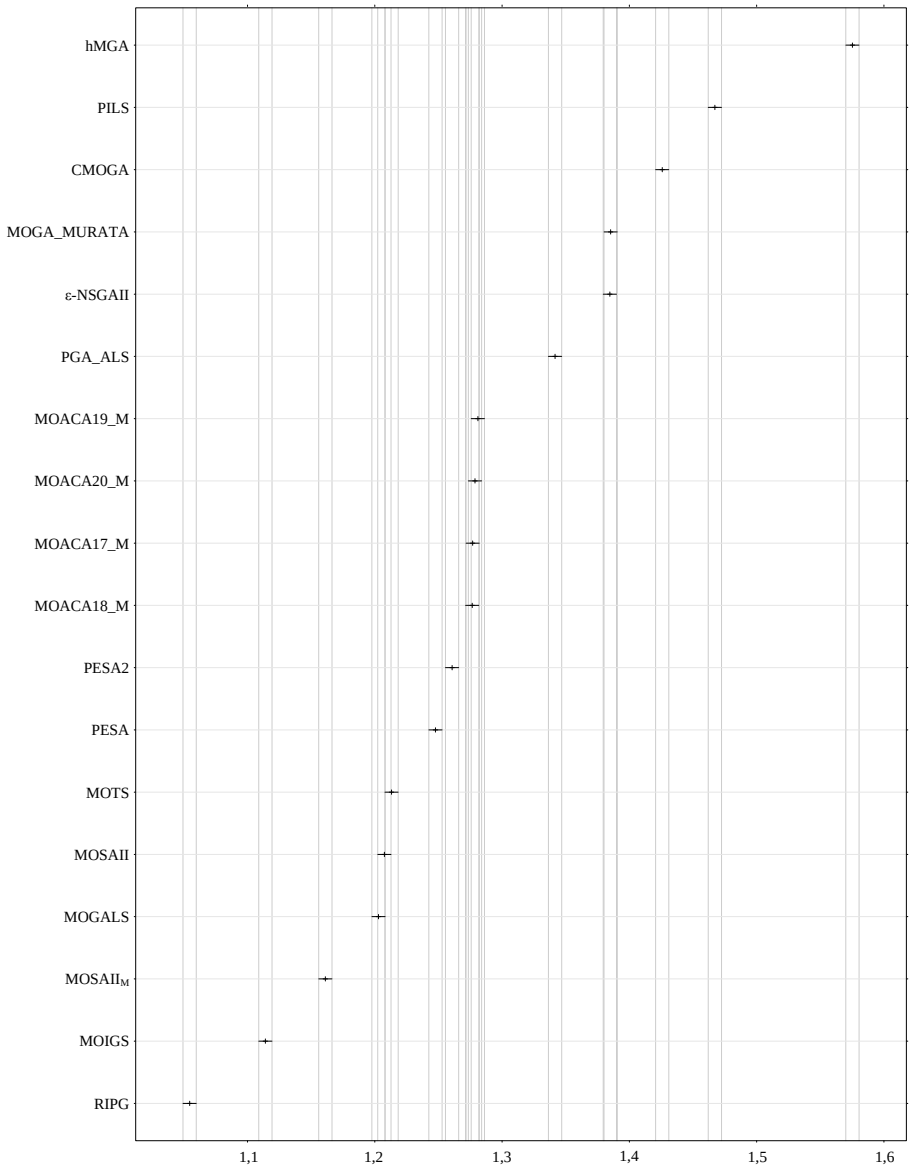


Figura 5.17: Resultados de tests ANOVA para el grupo de instancias SSD125, variable respuesta indicador épsilon y $t = 200$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

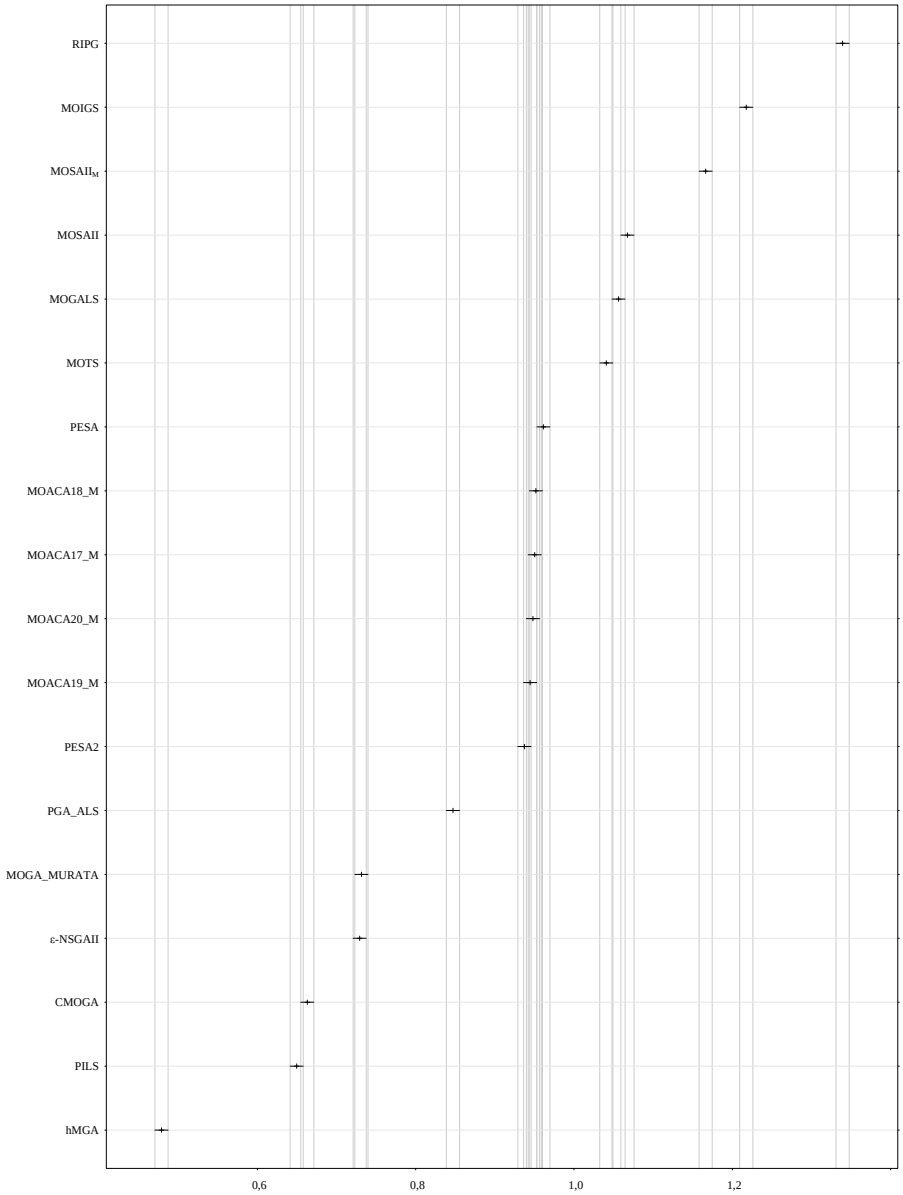
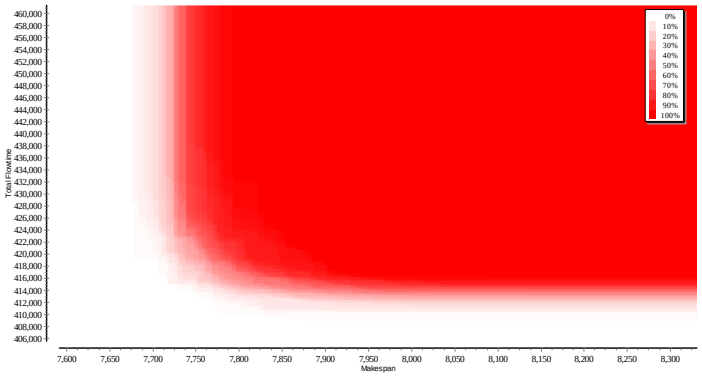


Figura 5.18: Resultados de tests ANOVA para el grupo de instancias SSD125, variable respuesta indicador de hipervolumen y $t = 200$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

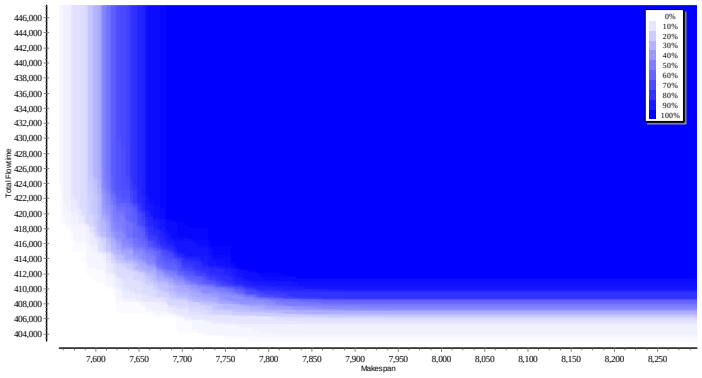
Para el experimento con la combinación de objetivos de $C_{\max}$ y tiempo total de flujo y el criterio de parada $t = 200$ el rendimiento de los algoritmos mantiene el mismo orden que para el experimento con $t = 150$ en el caso de el indicador I_H. Con el indicador I_ε solamente cambia la posición del algoritmo MOSAII de la cuarta a la quinta posición. Igual que pasa con los experimentos para $C_{\max}$ y tardanza total, este algoritmo va perdiendo posiciones al aumentar el tiempo de ejecución de los demás algoritmos debido a que siempre realiza un número fijo de iteraciones.

El Anexo C contiene las 16 gráficas que representan el análisis de Friedman. Estas gráficas están divididas de acuerdo al grupo de instancias y el criterio de parada t. El orden de los algoritmos se mantiene similar al del análisis ANOVA presentado en este capítulo. El algoritmo RIPG obtiene de manera consistente los mejores resultados, mientras que algunos algoritmos como MOIGS, MOSAII$_M$ y MOGALS se mantienen entre los primeros 5 algoritmos.

Finalmente presentamos nueve figuras que representan las gráficas de EAF y Diff-EAF para los algoritmos MOSAII$_M$, MOIGS y RIPG y para la instancia 71 del conjunto SSD50 (con 100 trabajos y 10 máquinas) y los objetivos *makespan* o $C_{\max}$ y tardanza total ponderada. Cada imagen ha sido elaborada utilizando los resultados de la ejecución de 50 réplicas para cada algoritmo. Cada gráfica de Diff-EAF conforma una comparación por diferencia de dos gráficas de EAF para dos algoritmos diferentes. Las gráficas de EAF han confirmado que el algoritmo RIPG obtiene resultados ampliamente mejores a los obtenidos por los demás algoritmos para la instancia probada.



(a) EAF para el algoritmo MOIGS.



(b) EAF para el algoritmo RIPG.

Figura 5.19: EAFs para los algoritmos MOIGS y RIPG, para la instancia ta071 con 100 trabajos y 10 máquinas. Resultados generados para $C_{\max}$ y tardanza total ponderada.

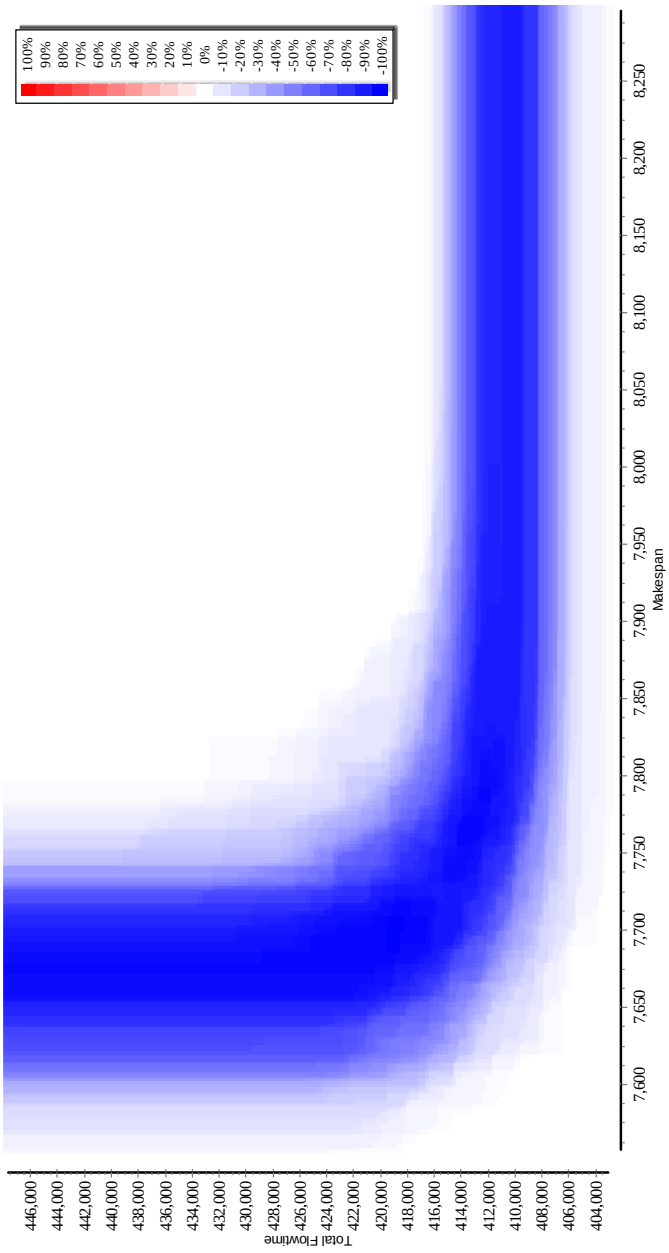
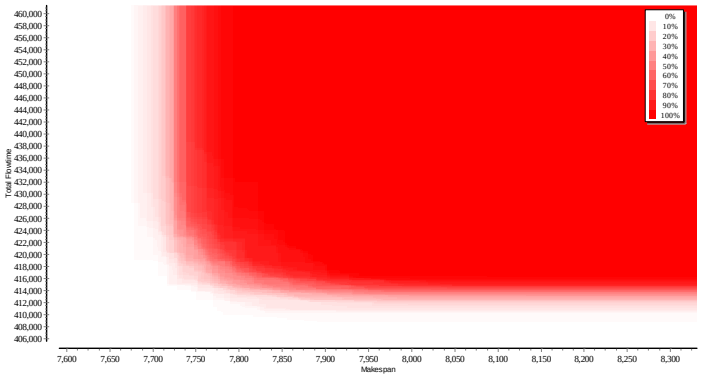
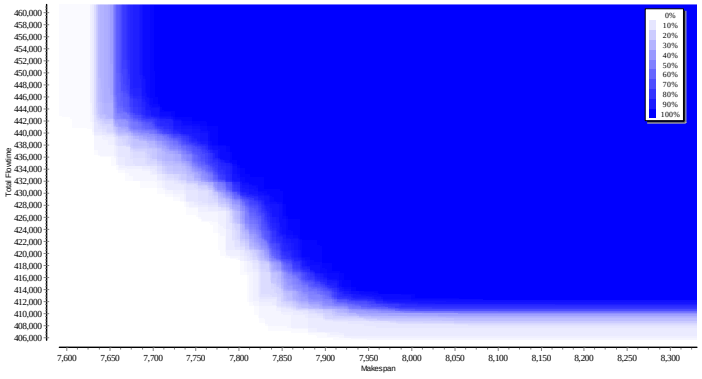


Figura 5.20: Diff-EAF para los algoritmos MOIGS y RIPG, para la instancia ta071 con 100 trabajos y 10 máquinas. Resultados generados para $C_{\max}$ y tardanza total ponderada.



(a) EAF para el algoritmo MOIGS.



(b) EAF para el algoritmo MOSAII$_M$.

Figura 5.21: EAFs para los algoritmos MOIGS y MOSAII$_M$, para la instancia ta071 con 100 trabajos y 10 máquinas. Resultados generados para $C_{\max}$ y tardanza total ponderada.

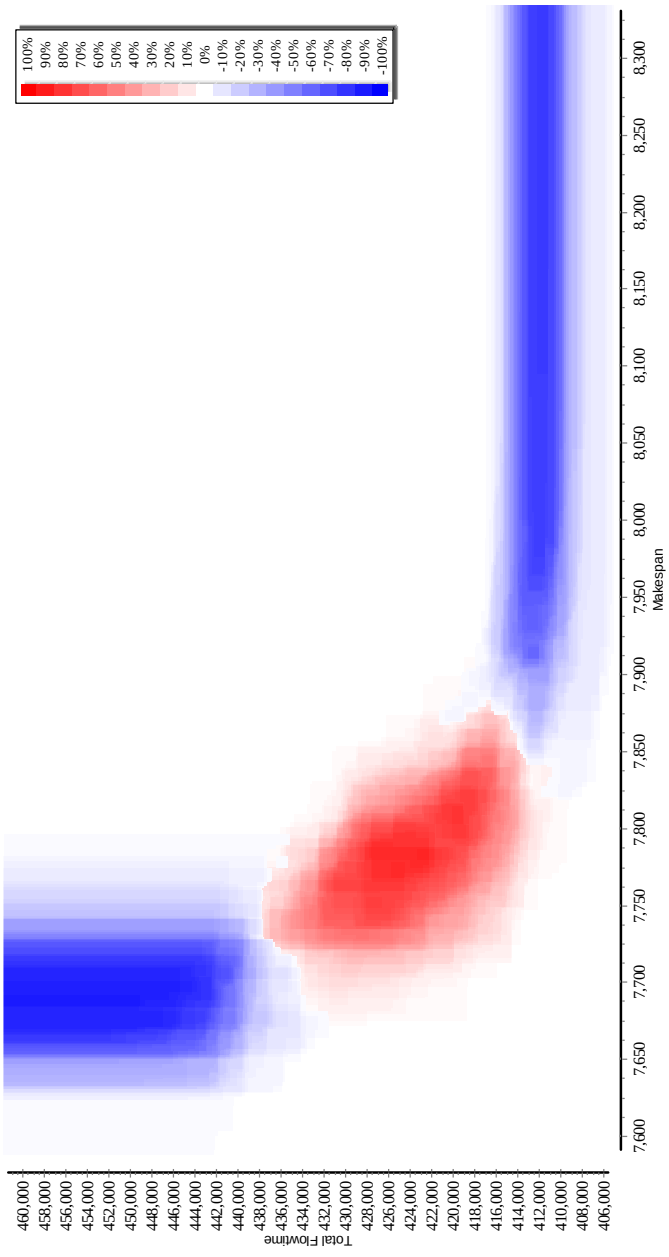
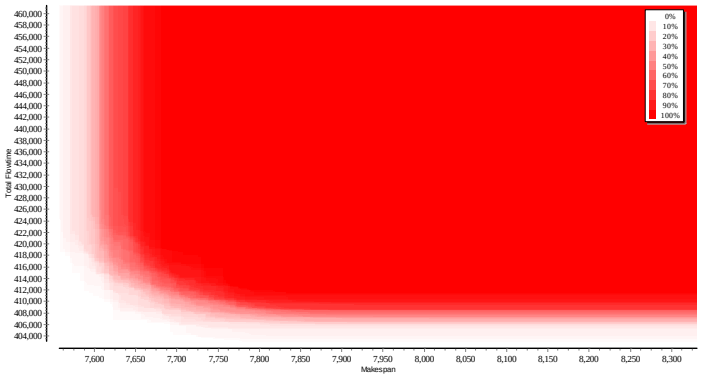
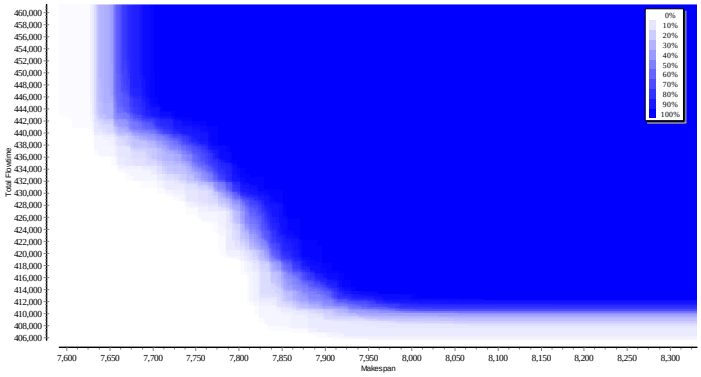


Figura 5.22: Diff-EAF para los algoritmos MOIGS y MOSAII$_M$, para la instancia ta071 con 100 trabajos y 10 máquinas. Resultados generados para $C_{\max}$ y tardanza total ponderada.



(a) EAF para el algoritmo RIPG



(b) EAF para el algoritmo MOSAII$_M$

Figura 5.23: EAFs para los algoritmos RIPG y MOSAII$_M$, para la instancia ta071 con 100 trabajos y 10 máquinas. Resultados generados para $C_{\max}$ y tardanza total ponderada.

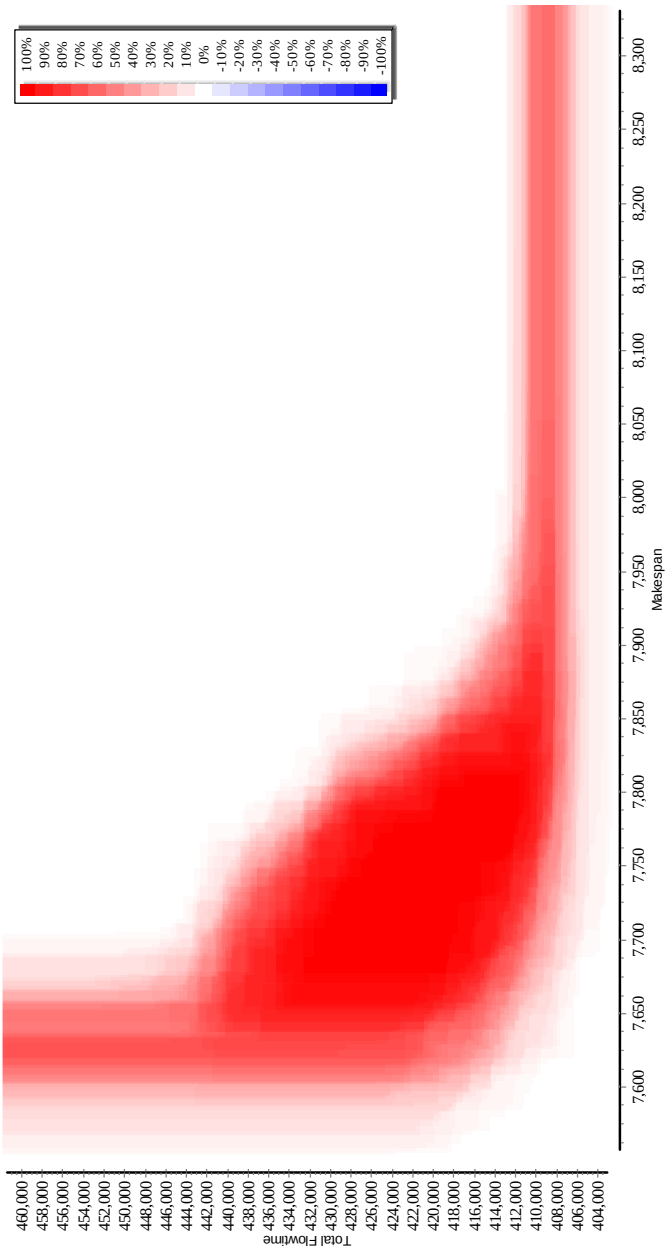


Figura 5.24: Diff-EAF para los algoritmos RIPG y MOSAII$_M$, para la instancia ta071 con 100 trabajos y 10 máquinas. Resultados generados para $C_{\max}$ y tardanza total ponderada.

5.3. Conclusiones del capítulo

A pesar de la importancia de los tiempos de cambio en las aplicaciones de la vida real, hasta la actualidad muy poco se ha hecho para resolver el taller de flujo multi-objetivo con la restricción de tiempos de cambio dependientes de la secuencia.

En este Capítulo realizamos la adaptación de los mejores métodos desarrollados para resolver el taller de flujo multi-objetivo, ampliando su funcionalidad para resolver el problema con la restricción de los tiempos de cambio anticipativos dependientes de la secuencia de producción. Llevamos a cabo un estudio de esos algoritmos para determinar su comportamiento ante las nuevas restricciones, determinando una clasificación según su rendimiento para todos ellos.

En un segundo paso hemos realizado la adaptación del algoritmo RIPG, propuesto en el Capítulo 4 para resolver estos problemas. La adaptación del nuevo algoritmo ha sido reforzada mediante la aplicación de la metodología de ingeniería de algoritmos utilizada denominada diseño de experimentos.
Mediante esta técnica hemos parametrizado el algoritmo RIPG propuesto de manera que se adapte mejor al problema estudiado.

Con la finalidad de comprobar el rendimiento de cada algoritmo adaptado y del algoritmo RIPG hemos utilizado conjunto de instancias basadas en las instancias de Taillard tanto para la batería de experimentos como para el diseño de experimentos.
Finalmente, hemos llevado a cabo una amplia campaña de experimentos para determinar la clasificación de los algoritmos. De estos experimentos, concluimos que nuestro algoritmo es el estado del arte en cuanto a la resolución de problemas de taller de flujo multi-objetivo con tiempos de cambio dependien-

tes de la secuencia para el conjunto de instancias utilizado.

Siguiendo nuestro camino hacia una representación más exacta de la realidad, en el siguiente capítulo estudiaremos el taller de flujo híbrido. En este tipo de problemas puede existir una secuencia de trabajos diferente para cada etapa de producción. Debido a que la resolución de este tipo de problemas es muy compleja, no tendremos en cuenta los tiempos de cambio.

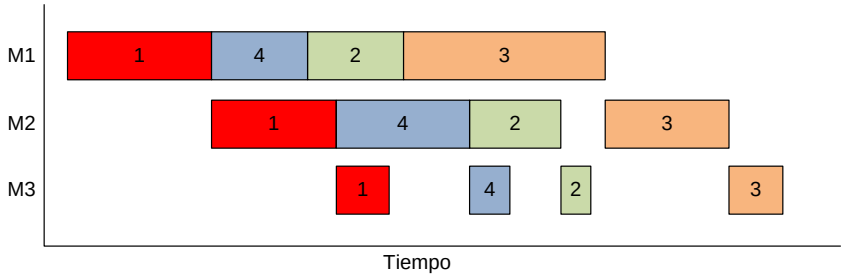
CAPÍTULO 6

RESOLVIENDO EL TALLER DE FLUJO DE HÍBRIDO MULTI-OBJETIVO

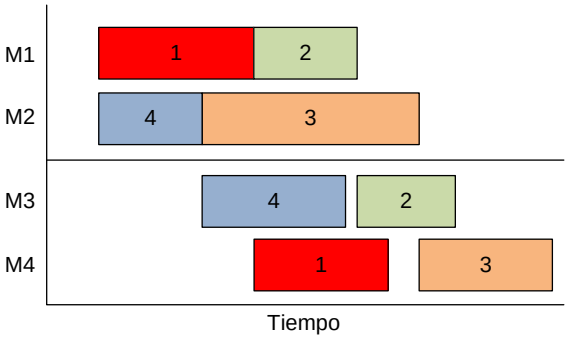
En el capítulo anterior hemos introducido una extensión al problema de taller de flujo de permutación: los tiempos de cambio. En un contexto industrial, esta extensión puede considerarse de gran importancia ya que ayuda a representar de manera más fiel a la realidad. Sin embargo existen muchos otros tipos de problemas que tienen mayor complejidad e intentan representar la realidad de una manera más precisa. El taller de flujo híbrido es una forma alternativa de representar la realidad que añade gran complejidad. En este capítulo estudiaremos el taller de flujo híbrido y presentaremos una revisión del estado del arte actual del problema. También presentaremos una modificación del algoritmo RIPG estudiado en capítulos anteriores para adecuarlo al problema del taller de flujo híbrido. Finalmente realizaremos una comparación de los métodos encontrados en la literatura y nuestro algoritmo propuesto.

El problema del taller de flujo híbrido consiste en una configuración con un conjunto M de m etapas, donde cada etapa $i \in M$ contiene a su vez un

conjunto M_i de m_i máquinas paralelas no relacionadas. Dado un conjunto N de n trabajos, donde existe para cada trabajo j una fecha de entrega d_j. Cada uno de estos trabajos debe ser procesados por exactamente una máquina l en cada etapa i. Para cada trabajo j, máquina l y etapa i se define un tiempo de proceso p_{ilj}.



(a) Representación de un taller de flujo de permutación.



(b) Representación de un taller de flujo híbrido.

Figura 6.1: (a) Un programa de producción representando un taller de flujo de permutación. (b) El mismo programa pero representado con un taller de flujo híbrido.

Como veremos en la siguiente sección, existen muy pocos estudios que

hayan abarcado este problema o alguna variedad del mismo.
Encontramos en la literatura dos artículos que hacen una revisión profunda del problema del taller de flujo híbrido. Aunque ninguno de ellos se especializa en problemas multi-objetivo, ambos nombran a este tipo de problemas.
El primero de ellos es el trabajo de Ruiz y Vázquez-Rodríguez (2010) en el cual se hace un profundo estudio de las aplicaciones, heurísticas y metaheurísticas existentes que tratan diferentes problemas de taller de flujo híbridos.En cuanto al taller de flujo híbrido multi-objetivo, este trabajo indica que a la fecha de su publicación existían apenas un par de artículos que trataban el problema del taller de flujo híbrido multi-objetivo, pero con aproximación "a priori" en vez de Pareto.
El otro trabajo de revisión de literatura que encontramos es el artículo de Ribas, Leisten y Framiñan (2010). En este trabajo no se estudia de una forma específica al problema del taller de flujo híbrido, sino problemas híbridos de producción en general. Los problemas se clasifican con distintos criterios, uno de los criterios de clasificación es el objetivo o los objetivos estudiados. Los autores encuentran que la mayoría de los artículos estudiados se centran el objetivo de $C_{\text{máx}}$. En cuanto a multi-objetivo los autores señalan que solamente han podido encontrar unos pocos artículos con esta aproximación y señalan también la importancia del estudio de este tipo de problemas en aras de acercar la investigación a problemas más realistas y de aplicación en empresas.

En cuanto a artículos que se dediquen al tema del taller de flujo híbrido multi-objetivo, la publicación más similar es el trabajo de Rashidi, Jahandar y Zandieh (2010), quienes estudian el problema del taller de flujo híbrido bi-objetivo Pareto con máquinas paralelas no relacionadas, tiempos de cambios dependientes de la secuencia y bloqueo de procesadores (*processor blocking*).

Este capítulo está estructurado de la siguiente manera: en primer lugar, en la Sección 6.1 presentaremos la revisión bibliográfica y estudiaremos el estado

del arte en cuanto al problema del taller de flujo híbrido multi-objetivo, luego en la Sección 6.2 describiremos el algoritmo propuesto para la resolución del problema, así como su calibración. Más adelante en la Sección 6.3 presentamos una comparativa del algoritmo propuesto contra algoritmos estado del arte propuestos en la literatura. También presentamos en esta misma sección los resultados obtenidos. Finalmente presentamos las conclusiones de este capítulo en la Sección 6.4.
Este Capítulo ha sido desarrollado en estrecha colaboración con Thijs Urlings, quien ha publicado también algunos de estos resultados en su trabajo de tesis doctoral (Urlings, 2011).

6.1. Revisión bibliográfica

Si cuando estudiamos el problema del taller de flujo de permutación multi-objetivo con tiempos de cambio observamos que las publicaciones que tratan el tema son escasas, en el caso del taller de flujo híbrido multi-objetivo esto se acentúa aún más, hasta el punto de que apenas pueden encontrarse un puñado de publicaciones que tratan este tema.

Recién en los últimos años han surgido algunas publicaciones que tratan variaciones del flowshop híbrido multi-objetivo. En 2009 Behnamian, Ghomi y Zandieh presentan una metaheurística híbrida multi-fase para el problema del taller de flujo híbrido multi-objetivo con tiempos de cambio dependientes de la secuencia, a la que denominan ECHM (ϵ-*Constraint covering Hybrid Metaheuristic*). En este trabajo proponen un algoritmo compuesto, que incluso llega a cambiar la representación de la solución varias veces durante su ejecución.
La población inicial se genera aleatoriamente. En el primer paso o fase se descompone la población inicial en varias sub-poblaciones. Cada una de ellas será procesada mediante un algoritmo genético propuesto por Kurz y Askin (2004) que representa la solución mediante claves aleatorias o *random-keys*. En

cada una de las sub-poblaciones se asignan pesos o ponderaciones para cada objetivo. El algoritmo genético se ejecuta contra cada sub-población como si fuera un problema de un solo objetivo (ya que los objetivos tendrán un peso fijado). Con esta fase se pretende aumentar la diversidad en las soluciones encontradas. Al finalizar cada iteración se actualiza el archivo global de soluciones a partir de todas las sub-poblaciones.
En la segunda fase del algoritmo se suman las sub-poblaciones en una única población llamada archivo de soluciones y se ejecuta una metaheurística de recocido simulado (*simulated annealing*) con búsqueda local. La segunda fase también es, a su vez, una metaheurística híbrida que suma las características del recocido simulado con la búsqueda de vecindario variable (*Variable Neighborhood Search*) propuesta por Mladenovic y Hansen (1997).
En cada iteración del recocido simulado se selecciona una solución y se ejecuta una búsqueda local de vecindario variable. Luego de ello se actualiza el archivo de soluciones no dominadas.
En la tercera etapa, se toma el archivo de soluciones no dominadas y se procesa mediante una metaheurística híbrida multi-objetivo. Todas las soluciones del archivo de soluciones se ordenan, de forma que se puedan procesar las soluciones por pares. Para cada par de soluciones "contiguas" se ejecuta la metaheurística híbrida para intentar encontrar nuevas soluciones "intermedias" que rellenen los huecos. La metaheurística híbrida consiste en ejecutar múltiples iteraciones un algoritmo de recocido simulado (Simulated annealing) que en cada iteración mejora la solución obtenida mediante una de varias búsquedas locales.
Existen tres posibles búsquedas locales, cada una está clasificada de acuerdo a su velocidad de ejecución. En cada iteración se ejecuta la búsqueda local más veloz, si esta búsqueda local no es capaz de mejorar la solución seleccionada, se ejecuta la siguiente búsqueda local, y así sucesivamente.

Los autores comparan el algoritmo propuesto contra un algoritmo genético

propuesto por Chang, Chen y Hsieh (2006) que utiliza una estrategia de sub-poblaciones similar a la propuesta en las primeras fases de su propio algoritmo. Lamentablemente, no realizan la comparación utilizando ningún algoritmo multi-objetivo propuesto en la literatura. La principal desventaja del algoritmo propuesto es la velocidad, ya que se trata de una concatenación de varias metaheurísticas. Por otro lado, la generación de soluciones iniciales es también una parte importante de todos los algoritmos que quieran ser competitivos. En este caso los autores han propuesto la generación aleatoria de soluciones iniciales, lo que en principio asegura un arranque lento del algoritmo.

Rashidi, Jahandar y Zandieh (2010) estudian el problema del taller de flujo híbrido bi-objetivo con enfoque Pareto, máquinas paralelas no relacionadas, tiempos de cambios dependientes de la secuencia y bloqueo de procesadores (*processor blocking*). Proponen dos algoritmos genéticos híbridos donde se divide la población inicial en varias sub-poblaciones, estos algoritmos genéticos están planteados de manera similar al propuesto por Behnamian, Ghomi y Zandieh (2009). Los algoritmos inician con una población aleatoria, que se divide en sub-poblaciones, y utilizan random keys como representación de la solución en su primera etapa, cada sub-población se resuelve mediante un algoritmo genético y se le asigna un peso a cada objetivo.
Para el proceso de selección utilizan la selección aleatoria de individuos, que luego se cruzan mediante el método de cruce de un punto (*One Point Crossover u OPX*). Luego del cruce se actualiza el archivo global donde se van guardando y actualizando las mejores soluciones obtenidas por los algoritmos genéticos que procesan cada sub-población. Las soluciones generadas solamente entran en cada sub-población si son mejores que las soluciones padre. Luego de esto se aplica un operador de mutación.

La diferencia entre los dos algoritmos propuestos es que el segundo utiliza una búsqueda local para evitar los óptimos locales y un procedimiento de

re-dirección que actualiza o cambia los pesos de los objetivos en una sub-población cuando la búsqueda local no es capaz de mejorarla.
La búsqueda local se aplica al mejor individuo de una sub-población luego de las fases de cruce y mutación, solamente si no se ha encontrado una solución que mejore a cualquiera de las soluciones elegidas en la fase de selección.
Cuando la búsqueda local es incapaz de mejorar al mejor individuo de la sub-población se utiliza un método de reinicio. En éste se varían los pesos de los objetivos para la sub-población y se crea una nueva sub-población. La mitad de los individuos se obtienen de aplicar el operador de mutación por intercambio (*swap*) a los individuos existentes y la otra mitad se obtiene generando soluciones aleatorias.
En este artículo los autores solamente comparan sus propios algoritmos contra un algoritmo genético para un solo objetivo, en el cual los objetivos a evaluar se combinan linealmente al asignarles un peso y no se comparan contra ningún algoritmo existente en el literatura multi-objetivo.

Los trabajos que tratan el problema del taller de flujo híbrido pueden contarse con los dedos de una mano, además de ello, en todos los casos se estudian problemas que difieren del taller de flujo híbrido "simple", que es el que estudiaremos en este capítulo. Para continuar con el estudio del taller de flujo híbrido utilizaremos el trabajo de Behnamian, Ghomi y Zandieh (2009) como punto de partida para nuestras comparaciones, ya que trata el problema más similar al que nos interesa. Por otro lado, además de proponer nuestro propio algoritmo para resolver el problema, utilizaremos el conocido algoritmo multi-objetivo NSGAII propuesto por Deb (2002) para los experimentos comparativos. Este algoritmo se ha utilizado de manera frecuente en las comparaciones de algoritmos multi-objetivo debido a su flexibilidad, fácil implementación y en general a que obtiene buenos resultados para problemas multi-objetivo.

6.2. Algoritmo propuesto

Para la resolución de este problema proponemos una modificación al antes expuesto algoritmo RIPG. El algoritmo RIPG ha mostrado resultados muy competitivos al utilizarlo con otras variaciones del problema del taller de flujo. Teniendo en cuenta las diferencias entre los problemas a resolver, en este caso hemos tenido que realizar una serie de modificaciones en el funcionamiento del algoritmo. Si bien hemos mantenido la estructura general, el método de selección, el método de evaluación y la representación.

Como pretendemos resolver un taller de flujo híbrido en vez de un taller de permutación, es necesario realizar una serie de decisiones a la hora de asignar las tareas en cada etapa de producción. En el taller de flujo de permutación basta con tener una permutación o secuencia de trabajos como representación de la solución. Como en cada etapa productiva se mantiene la misma permutación, esta representación es suficiente.
En el caso del taller de flujo híbrido, existen varias etapas de producción, cada una de las cuales se encarga de una o varias operaciones diferentes. Existen varias formas de representar un taller de flujo híbrido, una de las más extendidas es la representación de permutación. En esta el orden en la primer etapa viene dado por una permutación, mientras que el orden en las etapas subsiguientes se determina de acuerdo al momento en que terminan las tareas de la etapa anterior. Al existir un conjunto de tareas disponibles que pueden comenzar en un momento dado, es necesario seleccionar cual de esas tareas será la siguiente en comenzar. Para seleccionar la siguiente tarea en iniciar utilizamos un algoritmo o método denominado regla de asignación. Las reglas de asignación son algoritmos muy simples que conforman una serie de condiciones tales que son capaces de seleccionar una tarea de un grupo de tareas que pueden comenzar en un momento dado.
Existen muchas reglas de asignación diferentes, algunas de las más comunes

son: FCFS (*First Come, First Served*) la primera en llegar es la primera en servirse. Básicamente se trata de tener una lista ordenada por orden de llegada de las tareas disponibles y seleccionar siempre la primera de las tareas. Otra regla muy utilizada es la llamada SPT (*Shortest Processing Time First*) o primero la que tiene el tiempo de proceso más corto. En esta regla, se ordenan las tareas susceptibles de comenzar de acuerdo a su tiempo de proceso de menor a mayor. Luego se selecciona la que tiene menor tiempo de proceso. Finalmente, otra de las reglas más utilizadas es la LPT (*Longest Processing Time First*) o primero la que tiene el mayor tiempo de proceso. Es la regla opuesta a SPT, en lugar de seleccionar la tarea con tiempo de proceso más corto primero, seleccionará siempre la tarea que tenga el mayor tiempo de proceso.
Luego de realizar varias pruebas hemos seleccionado la regla LPT para asignar las tareas a las máquinas. Con lo cual, en cada proceso de selección se elige la tarea con mayor tiempo de proceso.

En resumen la modificación al algoritmo RIPG propuesta funciona de la siguiente manera: se generan dos soluciones iniciales para cada objetivo utilizando dos heurísticas. En el caso de utilizar dos objetivos se obtendrán cuatro soluciones iniciales. Las heurísticas utilizadas son la NEH propuesta por Nawaz, Enscore y Ham (1983), que obtiene muy buenas soluciones para $C_{\text{máx}}$ y la heurística propuesta por Rajendran y Ziegler (1997) que obtiene buenas soluciones cuando se utiliza el objetivo de tardanza total.
El conjunto de soluciones iniciales pasa a formar parte del conjunto de soluciones de trabajo. En un primer paso, todas las soluciones son procesadas por la fase voraz del algoritmo. Todas las soluciones obtenidas en este paso previo se añaden al conjunto de soluciones de trabajo. De esta manera se pretende tener un conjunto de soluciones iniciales diverso.
En cada iteración se selecciona una de las soluciones del conjunto de soluciones de trabajo mediante un operador llamado asignación de distancia poblacional modificado o *Modified Crowding Distance Assignment*, que es una

mejora al operador propuesto por Deb (2002). En este caso, el operador es capaz de variar su valor de acuerdo con la cantidad de veces que una solución ha sido seleccionada, de esta manera se garantiza que todas las soluciones serán seleccionadas en algún momento y además que no se perderá demasiado tiempo en soluciones que no se pueden mejorar. Este operador de asignación se describe en profundidad en el Capítulo 4, en la Sección 4.1.2.

La solución seleccionada es procesada en primera instancia por la fase voraz. En esta fase se obtiene un conjunto de soluciones candidatas, se actualiza el conjunto de soluciones de trabajo, y se vuelve a calcular el valor de distancia poblacional para cada solución. Luego se vuelve a seleccionar una solución y esta es mejorada mediante una búsqueda local.

En cada iteración se mide la posibilidad de que el algoritmo se haya estancado sin posibilidad de mejorar las soluciones existentes. Si esto ocurre se ejecuta la fase de reinicio. Esta fase consiste en guardar el conjunto de soluciones de trabajo en un archivo y generar aleatoriamente un nuevo conjunto de soluciones de trabajo. Todo este proceso se itera hasta que el criterio de parada se cumple.
Una vez determinada la secuencia inicial de proceso, se asignan las tareas a las máquinas de la primera etapa. Luego, se aplica la regla LPT para determinar el orden de proceso de las tareas en las etapas posteriores.

6.3. Fase experimental

6.3.1. Descripción del banco de pruebas

Para poder realizar la evaluación de los algoritmos empleamos un conjuntos de instancias generados de manera aleatoria. Este conjunto de instancias, contiene instancias con $n \in \{50, 100\}$, $m \in \{4, 8\}$ y $m_i \in \{2, 4\}$.

Utilizamos una distribución uniforme $p_{ilj} \sim U(1, 99)$ para asignar el tiempo de proceso p_{ilj} para el trabajo j en la máquina l de la etapa i.
El conjunto de instancias fue propuesto y utilizado por primera vez en Ruiz, Şerifoğlu y Urlings (2008), aunque para este trabajo hemos realizado algunos cambios, como añadirles fechas de entrega (*due dates*). También hemos quitado información no necesaria como las relaciones de precedencia. Finalmente hemos reducido la cantidad de instancias quitando el subconjunto de instancias pequeñas debido a que los resultados para estas instancias los obtienen de manera muy rápida todos los algoritmos y esto pervierte los resultados de las pruebas.
El conjunto original de instancias fue preparado teniendo en cuenta un número de parámetros como la cantidad de etapas, las máquinas de cada etapa y la cantidad de trabajos. Luego, de acuerdo a la cantidad de etapas, máquinas y número de trabajos predecesores se clasifica las instancias como grandes o pequeñas, obteniendo así dos conjuntos de instancias. Estas instancias son resultado de una combinación de 10 factores diferentes. De estos factores, los que hemos utilizado en el presente Capítulo y los distintos valores que reciben están representados en la Tabla 6.1. Con esta combinación de factores obtenemos un total de $2 \times 2 \times 2 \times 3 \times 3 \times 4 = 288$ instancias.
Para asegurarnos que los algoritmos probados no están sobre-ajustados a las instancias, hemos realizado las calibraciones de los algoritmos utilizando un conjunto de instancias diferente. El conjunto de instancias de calibración ha sido generado con las mismas condiciones que el conjunto de instancias final, pero con 2 réplicas por combinación de parámetros, lo que da un total de 144 instancias de calibración.

Para obtener fechas de entregas relevantes para los trabajos es necesario contar con los valores óptimos de $C_{\text{máx}}$, o a lo sumo una buena estimación. Estos valores óptimos y estimaciones fueron obtenidos en Urlings (2011) mediante un proceso de varios pasos. El primer paso consiste en ejecutar el modelo matemático del problema utilizando el solver CPLEX. Para esta

Factor	Representación	Niveles	Valores
Número de trabajos	n	2	50, 100
Número de etapas	m	2	4 ,8
Número de máquinas por etapa (i)	m_i	2	2, 4
Factor de tardanza	T	3	0,2, 0,4, 0,6
Rango fechas entrega	R	3	0,2, 0,6, 1
Replicas por tamaño		4	

Tabla 6.1: Factores considerados en el diseño de las instancias.

ejecución se ha utilizado un tiempo límite de una hora. Transcurrida esta hora puede ocurrir que CPLEX haya encontrado la solución óptima, con lo cual ya tendremos el valor de $C_{\text{máx}}$ que estábamos buscando. Si CPLEX no ha sido capaz de encontrar una solución óptima, al menos habrá encontrado una cota superior ($CPLEX_{\text{máx}}$) y una cota inferior ($CPLEX_{\text{mín}}$) para el valor de $C_{\text{máx}}$. Sabemos que el valor de $C_{\text{máx}}$ esta entre estos dos límites. Para ajustar un poco más la estimación de $C_{\text{máx}}$ utilizamos otras dos cotas inferiores, a las que denominamos $LB1$ y $LB2$.
Para ambas cotas inferiores definimos el tiempo de proceso para cada trabajo j en cada etapa i como el tiempo mínimo entre los diferentes tiempos de proceso en las máquinas de la etapa i, como se describe más formalmente en la Ecuación 6.1.

La la Ecuación 6.2 muestra el cálculo de la primera cota ($LB1$). Esta cota representa la suma de los tiempos de proceso mínimos de todos los trabajos. La segunda cota ($LB2$) es un poco más compleja. Primero calculamos el tiempo total de proceso mínimo para cada etapa i, a este valor le restamos la menor carga de trabajo por máquina. Luego le añadimos a este valor, para cada etapa i, el tiempo mínimo que se necesita para que se pueda iniciar la primera tarea de esa etapa. Como en la etapa i hay m_i máquinas y el primer

trabajo de cada máquina necesita procesarse antes en todas las etapas previas. El tiempo mínimo para la etapa i es la m_i ésima menor suma de los tiempos de proceso sobre las etapas previas. De forma análoga, se calcula el tiempo mínimo después de la etapa i como la m_i ésima menor suma de los tiempos de proceso sobre las etapas posteriores a la etapa i. Cabe destacar que $sort_z^+(S)_c$ se utiliza para indicar el elemento en la posición c de un conjunto S ordenado en forma ascendente.

Una vez obtenidas las tres cotas inferiores $LB1$, $LB2$ y $CPLEX_{\text{mín}}$ se toma la menor de las tres. Luego de ello, se obtiene una primera estimación $\widetilde{C_{\text{máx}}}^1$ haciendo la media entre la cota inferior seleccionada y la cota superior $CPLEX_{\text{máx}}$.
Puede ocurrir que la estimación de la cota superior $CPLEX_{\text{máx}}$ sea demasiado alta, si esto ocurre $\widetilde{C_{\text{máx}}}^1$ también será demasiado alta. Para subsanar este problema compararemos esta estimación contra un valor obtenido por la heurística NEH.
El siguiente paso para obtener la estimación de $C_{\text{máx}}$ será ejecutar la heurística NEH para obtener una segunda estimación $\widetilde{C_{\text{máx}}}^{NEH}$. Finalmente se comparan ambas estimaciones y se toma como $\widetilde{C_{\text{máx}}}$ la menor de las dos.

$$p_{ij} = \min_{l \in M_i} p_{ilj} \tag{6.1}$$

$$LB1 = \max_{j \in N} \sum_{i=1}^{m} p_{ij} \tag{6.2}$$

$$LB2 = \max_{i \in M} \Big(\sum_{j=1}^{n} p_{ij} + sort_k^+ (\sum_{a=1}^{i-1} p_{ak})_{m_i} + sort_o^+ (\sum_{b=i+1}^{m} p_{bo})_{m_i} \Big) \tag{6.3}$$

El valor de $C_{\text{máx}}$ utilizado para calcular las fechas de entrega puede ser el óptimo, si CPLEX ha podido encontrarlo o la mejor estimación $\widetilde{C_{\text{máx}}}$ encontrada mediante el proceso descrito en los párrafos anteriores.

Generamos las fechas de entrega considerando la estimación de $C_{\text{máx}}$ y otros dos parámetros: el factor de tardanza (T) y el rango de fecha de entrega (R). Elegimos cada fecha de entrega d_j para cada trabajo j utilizando una probabilidad uniforme en los rangos $C_{\text{máx}}(1-T-R)$ y $C_{\text{máx}}(1-T+R)$. Esta forma de generar las fechas de entrega fue introducida por Potts y Wassenhove (1982). Hemos elegido los valores para T y R de forma similar a como se hizo en Vallada, Ruiz y Minella (2008): $T = \{0,2,0,4,0,6\}$ y $R = \{0{,}2,0,6,1\}$
A partir de todas las posibles combinaciones de los parámetros anteriormente citados hemos obtenido un total de 144 instancias de calibración y 288 instancias para la pruebas finales.
En la Figura 6.2 podemos ver un ejemplo de las instancias utilizadas para la comparativa de algoritmos.

Etapa	1			2			3		
Máquina	1	2	3	4	5	6	7	8	9
Trabajo									
1	18	62	23	61	36	24	90	13	86
2	65	29	98	76	37	18	82	93	17
3	6	53	39	28	30	78	70	58	52
4	56	69	34	81	95	31	47	48	71
5	25	38	57	22	99	68	14	2	15

Tabla 6.2: Ejemplo de instancia, mostrando los tiempos de proceso para cada trabajo en cada máquina.

Como indicadores de calidad en todas las pruebas finales utilizaremos el indicador de hipervolumen y el indicador épsilon, los mismos que utilizamos a través de toda la tesis.

6.3.2. Calibración de los algoritmos

Antes de realizar la comparación final de los tres algoritmos involucrados es necesario hacer una serie de pruebas de calibración para determinar las mejores configuraciones de parámetros para cada algoritmo. Para las pruebas de calibración utilizamos el conjunto de instancias de calibración ya nombrado en la Sección 6.3.1. Este conjunto contiene en total 144 instancias. En las calibraciones solamente utilizaremos el indicador de hipervolumen ya que con ello es suficiente para determinar los mejores parámetros para un algoritmo.
Para muchos algoritmos ocurre que las diferencias obtenidas entre distintos valores de un mismo parámetro no son estadísticamente significativas. De todas maneras siempre elegiremos los valores de los parámetros que han obtenido una media de hipervolumen más alta.

6.3.2.1. Calibración del algoritmo NSGAII

Para la calibración del algoritmo NSGAII hemos utilizado los siguientes parámetros con los siguientes valores:

- probabilidad de cruce $X : 50\,\%, 70\,\%, 90\,\%$
- probabilidad de mutación $M : 30\,\%, 70\,\%, 1/n$
- tamaño de la población $P : 10, 30, 100$

Utilizamos la metodología de diseño de experimentos o DoE. En esta se realizan experimentos con cada combinación posible de parámetros y luego se comparan los valores obtenidos para cada uno de ellos. En nuestro caso hemos realizado experimentos para un total de $X \times M \times P = 27$ algoritmos. Estos algoritmos fueron ejecutados 5 veces (réplicas) para un total de 144 instancias de calibración. Utilizamos los objetivos de $C_{\max}$ y tardanza total y el indicador de hipervolumen.

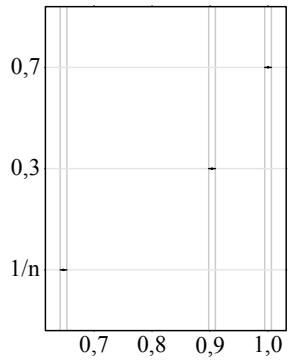
En este tipo de experimento, existen múltiples factores y es importante determinar cual de ellos es más significativo en los resultados. El factor más significativo será la que obtenga el mayor valor de F-Ratio. Se filtran los resultados dejando solamente los obtenidos para el nivel del factor con el mejor F-Ratio. El proceso continúa filtrando los resultados que quedan, siempre identificando el factor más significativo y volviendo a filtrar hasta que se llega al parámetro menos significativo.
La Tabla 6.3 muestra los resultados del análisis ANOVA para el indicador de hipervolumen y todos los parámetros probados. La columna *F-ratio* indica la cantidad de varianza que se debe a los niveles de cada factor por encima de la varianza del residuo (o ruido blanco). Por lo tanto, un F-Ratio muy alto indica un factor muy significativo, es decir un parámetro muy importante en el caso de la calibración.
En este caso el parámetro más significativo para el experimento de calibración del algoritmo NSGAII es la probabilidad de mutación M, el siguiente es el tamaño de la población P. Por último está la probabilidad de cruce X, que es un parámetro no significativo.

Parámetro	Suma de Cuadrados	Grados de libertad	Cuadrado medio	F-Ratio	P-Valor
X (Prob. Cruce)	0,0120486	2	0,0060243	0,06	0,9427
M (Prob. Mutación)	217,4500000	2	108,7250000	1065,31	0,0000
P (Población)	38,7070000	2	19,3535000	189,63	0,0000

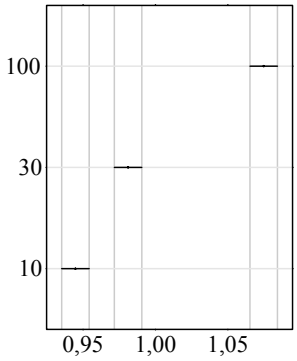
Tabla 6.3: Análisis de varianza para el indicador de hipervolumen y los parámetros calibrados para el algoritmo NSGAII.

El mejor resultado de hipervolumen para el parámetro M se obtiene para el valor $M = 70\,\%$, como muestra la Figura 6.2(a).

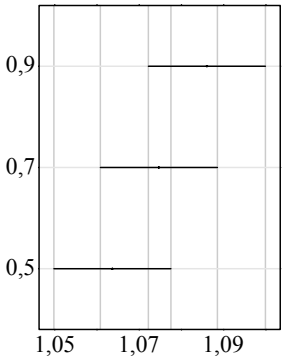
El siguiente parámetro a analizar es el tamaño de la población. Tomamos todos los resultados de la calibración y los filtramos de manera que solamente



(a) Probabilidad de mutación M.



(b) Tamaño de población P.



(c) Probabilidad de cruce X.

Figura 6.2: Resultados del test ANOVA para el experimento de calibración del algoritmo NSGAII con el indicador de hipervolumen. Parámetros de las diferentes partes que componen el algoritmo en orden de relevancia: (a) probabilidad de mutación M, (b) tamaño de la población P y (c) probabilidad de cruce X.

se tienen en cuenta los resultados para $M = 70\,\%$. La Figura 6.2(b) muestra los resultados para el análisis del comportamiento del parámetro P. En este caso, las diferencias entre los distintos valores para P ya no es tan grande

y elegiremos el valor de P que resulte en el mayor valor de media para el indicador de hipervolumen. En este caso, como muestra la gráfica el valor de P con la media más alta es $P = 100$.
El último parámetro bajo estudio es la probabilidad de cruce X. Volvemos a realizar un filtro sobre los resultados de la calibración, en este caso el filtro se hace para los dos parámetros anteriormente estudiados: $M = 70\,\%$ y $P = 100$. La Figura 6.2(c) muestra los resultados para el parámetro X. El valor que mejor media obtiene para el indicador de hipervolumen es $X = 90\,\%$. En conclusión, los mejores parámetros para el algoritmo NSGAII encontrados mediante la calibración son $P = 100$, $X = 90\,\%$ y $M = 70\,\%$.

6.3.2.2. Calibración del algoritmo RIPG

Para el algoritmo RIPG consideramos, más que parámetros, las distintas fases del algoritmo.
El algoritmo voraz iterado se compone de tres fases, alguna de las cuales puede quitarse o desactivarse. La primera fase del algoritmo es la fase voraz G, luego viene la fase de búsqueda local L, que es opcional y finalmente le añadimos la fase de reinicio R, que no se considera en el algoritmo voraz iterado original, pero que ha demostrado mejorar las soluciones en problemas multi-objetivo.
En este experimento de calibración también probamos el efecto de desactivar o quitar las fases de búsqueda local y re-inicio. Los valores utilizados en este experimento de calibración para las distintas fases del algoritmo son:

- $G : 3, 5, 10$
- $L : 0, 3, 5$
- $R : 0, 2 \cdot n, 10$

Donde el valor de G indica cuantos trabajos se excluyen durante la destrucción en cada iteración.
El valor de L indica la amplitud de la búsqueda local. Cuando $L = 0$ indica

que la búsqueda local no se ejecuta y un valor de 3 indica que un elemento seleccionado se intercambiará en todas las posiciones contando desde tres elementos a la derecha y tres elementos a la izquierda de su posición inicial.
El valor de R indica el momento en que se realizará el reinicio, un valor $R = 0$ indica que la fase de reinicio no se ejecutará, mientras que un valor de 10 indica que el reinicio se ejecutará luego que se hayan sucedido 10 iteraciones sin cambios en el tamaño de la población.

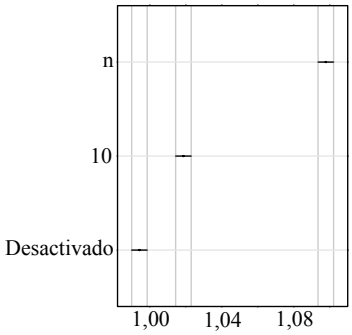
La Tabla 6.4 muestra los resultados para el experimento de calibración del algoritmo RIPG. Según los valores de *F-ratio* se puede ver que la fase más determinante en los resultados del algoritmo es la fase de reinicio R, esto indica que en este tipo de instancias el algoritmo se estanca y es necesario aplicar el reinicio. Luego está la fase voraz G y finalmente la fase de búsqueda local L, que es un parámetro no significativo.

Parámetro	Suma de Cuadrados	Grados de libertad	Cuadrado medio	F-Ratio	P-Valor
R	189,96500	2	94,982700	1044,56	0,0000
G	40,071100	2	20,035600	220,34	0,0000
L	0,353151	2	0,176575	1,94	0,1434

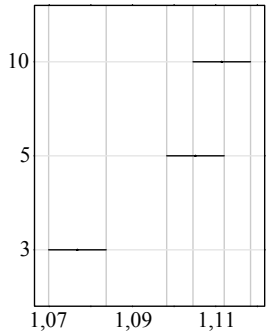
Tabla 6.4: Análisis de varianza para el indicador de hipervolumen y los parámetros calibrados para el algoritmo RIPG.

La Figura 6.3(a) muestra los resultados para la fase más relevante para el algoritmo RIPG, que es la fase de reinicio. La figura indica que el mejor valor para el parámetro R es $2 \cdot n$. Tanto en esta figura como en la Tabla 6.4 se pone en relieve como la inclusión de la fase de reinicio colabora en obtener mejores soluciones.

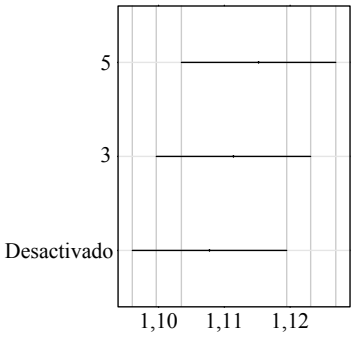
La figura 6.3(b) muestra que los mejores valores de hipervolumen se obtienen con $G = 10$. Esto indica que realizar pocas iteraciones que hacen cambios



(a) Fase de reinicio R.



(b) Fase voraz G.



(c) Fase de búsqueda local LS.

Figura 6.3: Resultados del test ANOVA para el experimento de calibración del algoritmo RIPG con el indicador de hipervolumen. Parámetros de las diferentes fases en orden de relevancia: (a) fase de reinicio R, (b) fase voraz G y (c) fase de búsqueda local L.

profundos en las soluciones es preferible a realizar muchas iteraciones que hacen cambios superficiales en las soluciones.

Finalmente el mejor valor para el parámetro L es 5, como muestra la Figura 6.3(c), aunque este parámetro es no significativo utilizaremos el valor de parámetro que otorgue un valor medio más alto.

Los experimentos de calibración para el algoritmo RIPG han demostrado que para este problema y conjunto de instancias se obtienen mejores resultados realizando el reinicio cada $2 \cdot n$ iteraciones sin cambios en el tamaño de población, configurando la fase voraz para que se destruyan 10 elementos en cada iteración y la búsqueda local de manera que el vecindario de inserción en la solución sea de tamaño 5.

6.3.2.3. Calibración del algoritmo EHCM

El algoritmo EHCM propuesto por Behnamian, Ghomi y Zandieh (2009) ha supuesto un desafío mayor a la hora de calibrarlo. En primer lugar debido a la propia complejidad del algoritmo, y también a la cantidad de posibles parámetros que contiene.
En el artículo original los autores realizan la calibración de su algoritmo modificando los parámetros del algoritmo genético y cambiando los parámetros de los dos recocidos simulados por igual. Nosotros realizamos la misma calibración. Los parámetros a calibrar son entonces la probabilidad de cruce Pc y la probabilidad de mutación Pm para el algoritmo genético y temperatura inicial ST y temperatura final FT para los recocidos simulados. Los valores para los parámetros son:

- Pc : $70\,\%; 80\,\%; 90\,\%$
- Pm : $0; 2\,\%; 5\,\%$
- ST : $-1; 0,7; 0,8$
- FT : $-1; 0,01; 0,02$

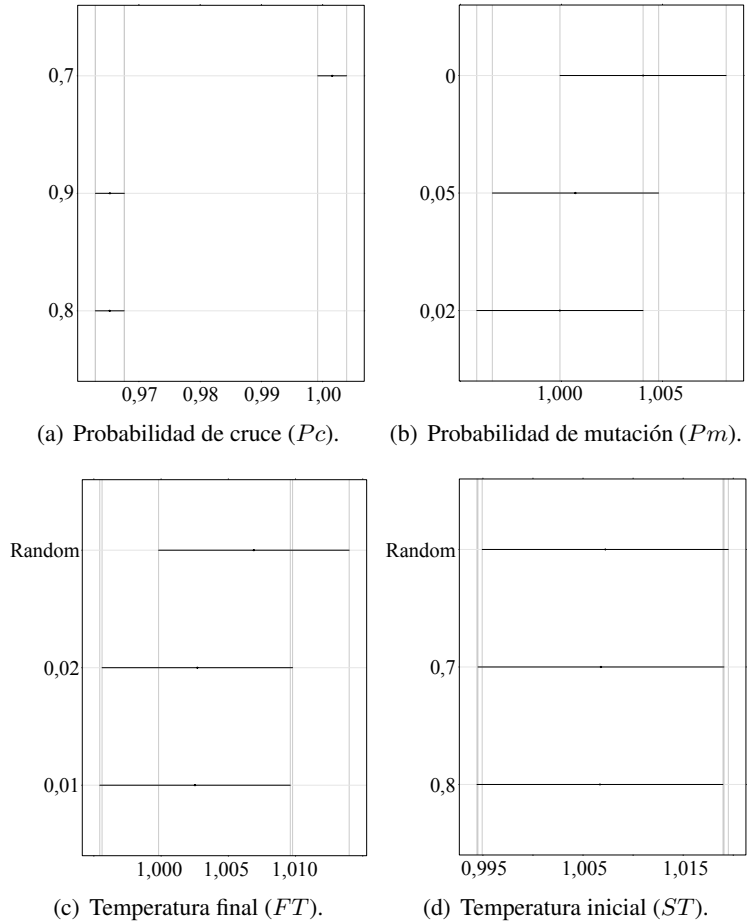
Donde el valor -1 indica que los valores temperatura final o inicial serán generados como un valor aleatorio entre el 10 y 100 en cada iteración.

La Tabla 6.5 muestra los resultados obtenidos para la calibración del algoritmo EHCM. En este caso, el valor más relevante es la probabilidad de cruce Pc. El segundo es la probabilidad de mutación Pm. Cabe destacar que los dos parámetros más importantes en la calibración resultaron ser parte de la primera fase del algoritmo, el método genético. Los siguientes parámetros en orden de importancia son, en tercer lugar la temperatura final FT del método de recocido simulado y el cuarto parámetro es la temperatura inicial ST del mismo método.

Parámetro	Suma de Cuadrados	Grados de libertad	Cuadrado medio	F-Ratio	P-Valor
Pm	7,05815000	2	3,52908000	86,07	0,0000
Pc	42,68640000	2	21,34320000	520,54	0,0000
ST	0,00201963	2	0,00100981	0,02	0,9757
FT	0,21259300	2	0,10629700	2,59	0,0748

Tabla 6.5: Análisis de varianza para el indicador de hipervolumen y los parámetros calibrados para el algoritmo EHCM.

Las Figura 6.4 muestra los resultados para los parámetros Pc, Pm, ST y FT, respectivamente en orden de relevancia.
El mejor valor encontrado para el parámetro más relevante Pc en este experimento de calibración es $70\,\%$ y para el parámetro Pm el mejor valor es 0. Estos parámetros sirven para ajustar el algoritmo genético que compone la primer fase de la metaheurística EHCM. En la misma figura pueden verse los resultados de la calibración de los parámetros de la fase de recocido simulado. Ambos parámetros son no significativos y, por lo tanto, los distintos valores de cada parámetros no varían demasiado el resultado final. De todas formas siempre se eligen los valores de cada parámetro que obtengan mejor media para el indicador de hipervolumen. En este caso los valores para los parámetros del recocido simulado son $Random$ para FT y $Random$ para ST.
Una vez que hemos encontrado los mejores valores para todos los parámetros de los distintos algoritmos podemos realizar la comparación final. En la sección siguiente ahondaremos en la descripción del banco de pruebas y veremos los resultados finales obtenidos con los tres algoritmos comparados.



(a) Probabilidad de cruce (Pc). (b) Probabilidad de mutación (Pm).

(c) Temperatura final (FT). (d) Temperatura inicial (ST).

Figura 6.4: Resultados del test ANOVA para el experimento de calibración del algoritmo EHCM con el indicador de hipervolumen. Parámetros de la fase genética en orden de relevancia probabilidad de cruce (a) y probabilidad de mutación (b). Parámetros la fase de recocido simulado en orden de relevancia temperatura final (c) y temperatura inicial (d).

6.3.3. Comparativa

Realizamos la comparación de los tres algoritmos calibrados utilizando las instancias que nombramos al comienzo de esta sección, quitando las instancias que utilizamos para la calibración. Utilizamos los dos indicadores de calidad ya utilizados en capítulos anteriores: el indicador de hipervolumen y el indicador épsilon.

Para esta comparación utilizaremos tres criterios de parada diferentes, igual que hicimos en el Capítulo 5. Al utilizar diferentes criterios de parada se puede comprobar si la diferencia entre los algoritmos se mantiene, o cambia, al variar la cantidad de tiempo de ejecución. Así se comprueba que las diferencias en los experimentos no se deben a falta de tiempo de ejecución. Cuando las diferencias se mantienen o se incrementan al aumentar el tiempo de ejecución queda claro que un algoritmo tiene mejor rendimiento que otro. Los tres criterios de parada utilizados en los experimentos de esta sección son: $t = 50$, $t = 200$ y $t = 300$.

La Tabla 6.6 muestra las medias para el indicador de hipervolumen y para el indicador épsilon de los experimentos separadas por los distintos valores de criterio de parada t. En cada columna, el mejor valor se ha resaltado. Como puede verse el algoritmo RIPG obtiene, con diferencia, los mejores resultados en todos los experimentos, para todos los distintos criterios de parada y para los dos indicadores probados. El algoritmo EHCM obtiene unos resultados tan pobres que no permiten la correcta comparación gráfica de los otros algoritmos. Por esta razón, en las siguientes figuras no se mostrarán los resultados obtenidos por el algoritmo EHCM en los experimentos debido a que solamente añade ruido a la comparación.

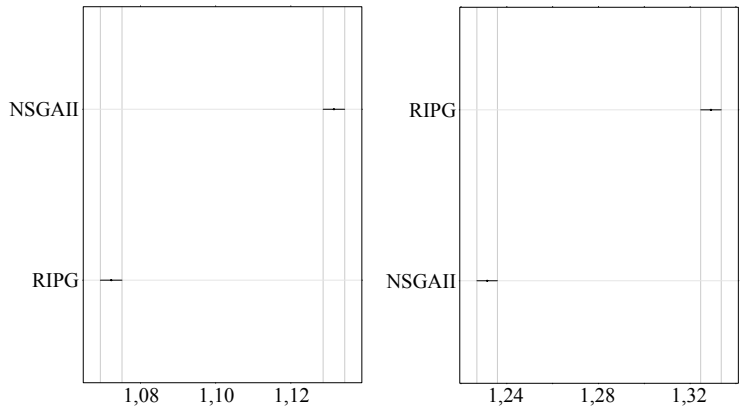
Las Figuras 6.5(a) y 6.5(b) muestran los resultados para los indicadores epsilon y de hipervolumen respectivamente, para el experimento con criterio de parada $t = 50$. Los resultados de los experimentos para $t = 200$ se muestran en las Figuras 6.6(a) y 6.6(b) para los indicadores épsilon y de hipervolumen respectivamente. Finalmente las Figuras 6.7(a) y 6.7(b) muestran los resultados

t	50		200		300	
Método	I_H	I_ε	I_H	I_ε	I_H	I_ε
RIPG	**1,3297**	**1,0716**	**1,3723**	**1,0470**	**1,3832**	**1,0408**
NSGAII	1,2354	1,1294	1,2633	1,1130	1,2782	1,1041
EHCM	0,3464	1,6999	0,3476	1,6996	0,3472	1,6983

Tabla 6.6: Tabla de medias para los indicadores de indicadores de hipervolumen y épsilon, para todas las instancias. Valores clasificados según los diferentes criterios de parada $t = 50$, $t = 200$ y $t = 300$.

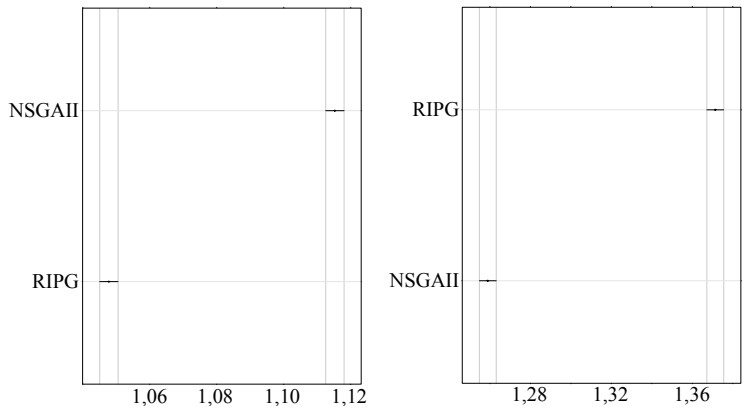
obtenidos para los indicadores épsilon y de hipervolumen para el criterio de parada $t = 300$.

Como ya se puede ver en la Tabla 6.6 el algoritmo RIPG propuesto en este capítulo para resolver el problema del taller de flujo híbrido multi-objetivo obtiene de manera consistente los mejores resultados para todos los criterios de parada y para los dos indicadores probados en las instancias propuestas.



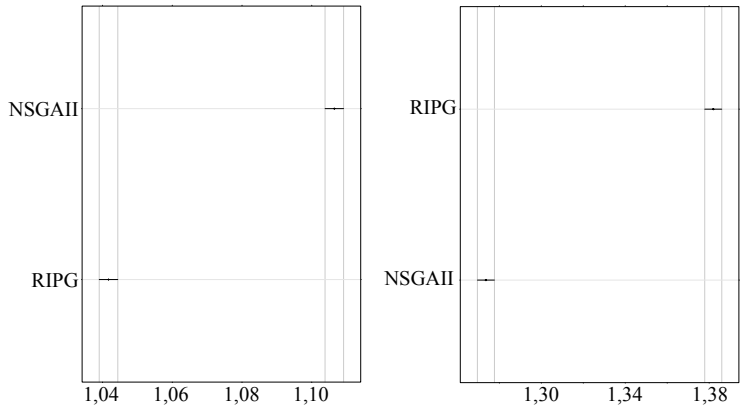
(a) Indicador épsilon. (b) Indicador de hipervolumen.

Figura 6.5: Gráfica de medias e intervalos de confianza de Tuckey al 99 % para el indicador épsilon (a) e indicador de hipervolumen (b). Resultados del test ANOVA del experimento para la comparación los algoritmos NSGAII y RIPG. Objetivos $C_{\text{máx}}$ y tardanza total, criterio de parada $t = 50$.



(a) Indicador épsilon. (b) Indicador de hipervolumen.

Figura 6.6: Gráfica de medias e intervalos de confianza de Tuckey al 99 % para el indicador épsilon (a) e indicador de hipervolumen (b). Resultados del test ANOVA del experimento para la comparación los algoritmos NSGAII y RIPG. Objetivos $C_{\max}$ y tardanza total, criterio de parada $t = 200$.



(a) Indicador épsilon. (b) Indicador de hipervolumen.

Figura 6.7: Gráfica de medias e intervalos de confianza de Tuckey al 99 % para el indicador épsilon (a) e indicador de hipervolumen (b). Resultados del test ANOVA del experimento para la comparación los algoritmos NSGAII y RIPG. Objetivos $C_{\max}$ y tardanza total, criterio de parada $t = 300$.

6.4. Conclusiones del capítulo

En este capítulo hemos presentado el problema de taller de flujo híbrido multi-objetivo. Este problema representa un paso más de acercamiento hacia problemas realistas multi-objetivo. Se trata de cambiar la representación que se hace de la realidad mediante la asignación de diferentes secuencias de trabajos en las distintas etapas que componen el problema. En este caso, hemos elegido el problema de taller de flujo híbrido sin ninguna restricción extra, como podrían ser los tiempos de cambio, para permitirnos realizar estudios graduales de problemas cada vez más cercanos a la realidad.

En nuestra revisión bibliográfica para este problema hemos notado que este tipo de problemas prácticamente no ha sido estudiado aún, habiendo encontrado apenas dos artículos que estudian algunas variaciones sobre el tema. Cabe destacar que en ninguno de los artículos encontrados se realiza una comparación de los algoritmos propuestos contra otros algoritmos genéricos multi-objetivo presentes en la literatura y ampliamente conocidos como puede ser el algoritmo NSGAII. En este capítulo se demuestra claramente que incluir este tipo de algoritmos cuando se propone un nuevo método para un problema que no se ha resuelto antes es casi una obligación. Prueba de ello es el buen desempeño del algoritmo NSGAII con las pruebas propuestas.

Proponemos aquí la comparación de uno de los algoritmos propuestos en la literatura, el que resuelve el problema más cercano al que estamos estudiando, llamado por sus autores EHCM, contra un algoritmo genérico ampliamente conocido y utilizado en problemas multi-objetivo, como es el algoritmo NSGAII. Además proponemos también una modificación al algoritmo RIPG, que ya presentamos en capítulos anteriores. Con estos tres algoritmos desarrollamos un extenso estudio para, en primer lugar, parametrizar cada algoritmo de la manera más competitiva que se pueda para el problema propuesto. Luego

de ejecutar un experimento de parametrización para cada algoritmo, también llamado por su sigla DoE (Diseño de experimento) hemos realizado un experimento comparativo. Como resultado a la comparación final de los algoritmos hemos visto como el algoritmo RIPG obtiene de manera consistente y para todos los indicadores y diferentes criterios de parada utilizados los mejores resultados. También es destacable el hecho de que el algoritmo NSGAII obtiene también de manera consistente mejores resultados que el algoritmo EHCM que ha sido propuesto para resolver problemas de taller de flujo híbridos multiobjetivo.

Este es el último capítulo de este trabajo de tesis, en el siguiente capítulo presentaremos las conclusiones del trabajo, plantearemos un hilo de posibles investigaciones que se desprenden de todo el trabajo realizado y ampliaremos acerca del aporte realizado durante este trabajo a la comunidad científica.

CAPÍTULO 7

CONCLUSIONES Y DESARROLLO FUTURO

La historia relativa a la investigación del taller de flujo tiene casi 60 años. En este tiempo se han estudiado muchas variaciones del taller de flujo para un solo objetivo. Sin embargo los problemas de taller de flujo para varios objetivos han recibido relativamente poca atención, posiblemente debido a su complejidad. En todo caso los avances en cuanto a algoritmos (heurísticas y metaheurísticas) han comenzado a notarse en las últimas dos décadas y han recibido un empuje especialmente notorio en los últimos años.
A través de este trabajo de tesis hemos realizado un recorrido de los problemas de taller de flujo multi-objetivo, desde los más estudiados y simples hasta problemas muy poco estudiados y más complejos. También hemos desarrollado, probado y ajustado nuestro propio algoritmo, basado en el algoritmo *Iterated Greedy* propuesto por Ruiz y Stützle (2007).

El primer problema que hemos estudiado ha sido el taller de flujo de permutación multi-objetivo simple (sin restricciones), pasando luego al taller de flujo de permutación con tiempos de cambio dependientes de la secuencia,

para finalmente estudiar un problema más realista, como es el taller de flujo híbrido multi-objetivo.

Hemos realizado una extensa revisión de la literatura multi-objetivo para el problema del taller de flujo, uno de los problemas más estudiados en el campo de la programación de la producción. Esta investigación amplía y completa otras existentes, como Nagar, Heragu y Haddock (1995b), T'Kindt y Billaut (2001) o Hoogeveen (2005) que no se refieren al taller de flujo de varias máquinas. Los artículos revisados incluyen también técnicas heurísticas para diferentes tipos de enfoques multi-objetivo.
Además de la revisión bibliográfica hemos realizado la implementación de 23 algoritmos, algunos de ellos propuestos para el taller de flujo multi-objetivo y otros generales (como el NSGAII). Con este profundo estudio hemos sido capaces de realizar un trabajo nunca antes llevado a cabo hasta la actualidad: la comparación experimental de los mejores algoritmos propuestos hasta el momento.
Los resultados de este trabajo inicial han sido presentado en varios congresos nacionales e internacionales:

- Ruiz, R. y Minella, G. (2007). Nuevos algoritmos avanzados para secuenciacion multiobjetivo. En *XXX Congreso Nacional de Estadística e Investigación Operativa*.
- Minella, G., Ruiz, R., y Ciavotta, M. (2007a). A review and evaluation of multi-objective algorithms for the flowshop scheduling problem. En *EURO XXII 2007 22nd European Conference on Operational Research*.
- Minella, G., Ruiz, R., y Ciavotta, M. (2007b). A review and evaluation of multi-objective algorithms for the flowshop scheduling problem. En *ORP3 Conference*.

Finalmente hemos publicado los resultados obtenidos en un artículo impreso en la revista *Informs*:

- Minella, G., Ruiz, R., y Ciavotta, M. (2008a). A review and evaluation of multi-objective algorithms for the flowshop scheduling problem. *Informs Journal on Computing*, 20(3):451–471.

Cabe destacar que durante el año de publicación de nuestro artículo, la revista *Informs* tuvo un índice de impacto de $1,041$ y ocupó la posición 28 de 64, quedando en el percentil 52.
Este artículo ha recibido, hasta el 2013, 52 citas según *google scholar* y 25 citas según *Web of Knowledge*.
Otra contribución de este trabajo es un nuevo conjunto de instancias de pruebas, basadas en las bien conocidas pruebas de Taillard (1993), que está disponible en línea en `http://soa.iti.es`, junto con los mejores resultados conocidos para los objetivos probados.

En el Capítulo 4 hemos propuesto un algoritmo basado en un algoritmo de búsqueda voraz iterativa o IG. También hemos propuesto una mejora al operador de asignación *Crowding Distance Operator* o CDO propuesto en Deb (2002) al que denominamos *Modified Crowding Distance Assignment* o MCDA. Este método de asignación de distancia poblacional es muy eficaz y permite obtener soluciones bien distribuidas en la frontera.
El algoritmo propuesto ha obtenido excelentes resultados, siendo estado del arte para el problema del taller de flujo de permutación "simple".
Un aspecto importante del algoritmo desarrollado es que se trata de un algoritmo versátil y genérico que se adapta fácilmente a la resolución de varios problemas.
Hemos presentado los resultados de éste capítulo en los siguientes congresos nacionales e internacionales:

- Minella, G., Ruiz, R., y Ciavotta, M. (2008b). New iterated pareto greedy algorithms for the multi-objective flowshop problem. En *Eleventh international workshop on project management and scheduling - PMS 2008*.

- Minella, G., Ruiz, R., y Ciavotta, M. (2009). Un algoritmo voraz iterativo para el taller de flujo de permutación con múltiples objetivos. En *XXXI Congreso Nacional de Estadística e Investigación Operativa*.

Además de ello, hemos aportado este conocimiento a la comunidad científica publicando un artículo en la revista *Computers and Operations Research*:

- Minella, G., Ruiz, R., y Ciavotta, M. (2011). Restarted iterated pareto greedy algorithm for multi-objective flowshop scheduling problems. *Computers and Operations Research*, 38(11):1521–1533.

Este artículo fue publicado en el año 2011. Durante ese año la revista tuvo un índice de impacto de $1,720$ y se ubicó en el percentil 16, estando en la posición 7 de 43.
Este artículo ha recibido hasta el 2013 7 citas según *google scholar* y 4 citas según *scopus*.

Uno de los objetivos de esta tesis es el estudio de problemas de taller de flujo más complejos que el taller de permutación "simple". Orientados hacia ese objetivo hemos aplicado el al algoritmo RIPG al problema de taller de flujo de permutación con tiempos de cambio dependientes de la secuencia. Hemos dedicado el Capítulo 5 al estudio de este problema. Al no encontrar ningún artículo que propusiera algoritmos para este problema multi-objetivo decidimos adaptar los 10 mejores algoritmos multi-objetivo que encontramos en capítulos anteriores. A estos algoritmos añadimos la adaptación del ya nombrado algoritmo RIPG.
Hemos presentado los resultados obtenidos en el Capítulo 5 en varios congresos, tanto nacionales como internacionales:

- Ciavotta, M., Ruiz, R., y Minella, G. (2009). Multi-objective sequence dependent setup times flowshop scheduling: A new algorithm and a comprehensive study. En *Learning and intelligent optimization Third International Conference (LION3)*.

- Ciavotta, M., Minella, G., y Ruiz, R. (2009). Un algoritmo eficaz para el problema del taller de flujo multiobjetivo con tiempos de cambio dependientes de la secuencia. En *XXXI Congreso Nacional de Estadística e Investigación Operativa*.

También hemos publicado estos resultados un artículo en la revista *European Journal of Operational Research* en el año 2013:

- Ciavotta, M., Minella, G., y Ruiz, R. (2013). Multi-objective sequence dependent setup times permutation flowshop: A new algorithm and a comprehensive study. *European Journal of Operational Research*. En prensa.

Durante el año 2012 (todavía no hay datos para el 2014) esta revista ocupó la posición 9 de 78 quedando en el percentil 12, con un índice de impacto de $2,038$.

Con la finalidad de seguir nuestro camino hacia problemas más complejos y más realistas estudiamos en el Capítulo 6 el problema del taller de flujo híbrido multi-objetivo.
En la revisión bibliográfica hemos visto que el problema del taller de flujo híbrido multi-objetivo no ha sido estudiado hasta hace poco tiempo, donde recién aparecen tres artículos a partir del año 2009. Hemos adaptado el algoritmo RIPG a este problema obteniendo excelentes resultados.
El trabajo realizado en el Capítulo 6 ha sido realizado en estrecha colaboración con Thijs Urlings y los resultado preliminares obtenidos también han sido incluidos en su trabajo de tesis Urlings (2011). Hemos presentado también estos resultados en dos congresos:

- Urlings, T., Minella, G., y Ruiz, R. (2010b). Optimización muilti-objetivo para problemas de flowshop híbrido. En *XXXII Congreso Nacional de Estadística e Investigación Operativa y de las VI Jornadas de Estadística Pública*.

- Urlings, T., Minella, G., y Ruiz, R. (2010a). Bi-objective pareto optimization for the hybrid flowshop problem. En *Twelfth International Workshop on Project management and Scheduling*.

En la actualidad continuamos el trabajo necesario para convertir estos resultados preliminares en un artículo completo, que esperamos tener publicado en 2014.

Nuestra visión respecto al futuro de nuestra investigación es muy prometedor. Partiendo de todo el trabajo realizado en estos años avanzamos hacia la resolución de problemas complejos de una manera natural, escalando en cada largo paso un peldaño más hacia la resolución de problemas complejos y realistas. Nuestra intención, como es de esperar, es continuar este camino, incluyendo en cada paso algún condicionante que lleve nuestra investigación un paso más cerca de los problemas industriales y logísticos que tienen que acometer las empresas de nuestra comunidad día a día.

APÉNDICE A

RESULTADOS DE TESTS DE RANGOS PARA LOS EXPERIMENTOS DE RIPG PARA EL PROBLEMA PFSP SIN TIEMPOS DE CAMBIO

En este anexo se incluyen los resultados de tests de rangos que completan los resultados presentados en el Capítulo 4, donde contrastamos el rendimiento de los tres mejores algoritmos del Capítulo 3, contra otros algoritmos surgidos en los últimos tiempos y contra nuestro algoritmo RIPG propuesto en el Capítulo 4. Los resultados se han separado, al igual que en el Capítulo 4, en tres partes, donde en cada parte se presentan los resultados para una de las combinaciones de objetivos probadas en los experimentos.

A.1. Resultados de tests de rangos para los experimentos con $C_{\max}$ y Tiempo total de flujo

En esta sección mostramos los resultados de los tests de rangos para los experimentos del Capítulo 4. La siguiente tabla presenta los rangos medios

para cada algoritmo y cada indicador en los distintos valores de t utilizados en los experimentos. Un valor menor en un test de rangos indica que el algoritmo ha obtenido una mejor posición respecto a los demás algoritmos en los experimentos, esto es válido para ambos indicadores. La Tabla A.1 muestra los valores obtenidos para los tests de rangos ordenados de menor a mayor para los indicadores de hipervolumen y épsilon. Los valores señalados en negrita indican el mejor valor para cada indicador y cada valor de t.

	Tiempo	100			200	
#	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
1	RIPG	**13,8655**	**10,9991**	RIPG	**13,2791**	**10,7018**
2	MOSAII$_M$	17,9591	21,9409	MOSAII$_M$	17,7627	22,6091
3	MOIGS	33,0127	26,6464	MOIGS	29,6018	23,2409
4	MOSAII	35,5673	36,8073	MOGALS	38,0036	36,4709
5	MOGALS	37,8764	37,4518	MOSAII	39,8936	41,4245
6	MOTS	47,2536	46,7000	MOTS	47,2455	45,5891
7	hMGA	66,1582	65,7909	hMGA	66,0473	65,7000

Tabla A.1: Resultados de tests de rangos para $C_{\text{máx}}$ y tiempo total de flujo con los criterios de parada $t = 100$ y $t = 200$. Los métodos están ordenados de acuerdo a I_H.

A continuación veremos las gráficas representando los valores mostrados en la tabla precedente. Cada gráfica representa los valores medios de rangos para un valor de t y uno de los indicadores utilizados, indicador de hipervolumen o indicador épsilon. Al igual que en la Tabla A.1 los valores más bajos indican un mejor rendimiento.

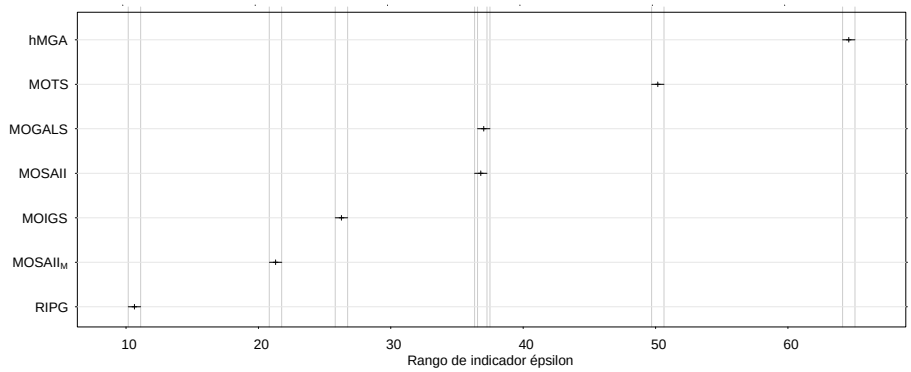


Figura A.1: Resultados de tests de rango para el indicador épsilon y $t = 100$ para los objetivos de $C_{\max}$ y tardanza total con intervalos de confianza de Tukey al 99 % (niveles de confianza ajustados al 95 %).

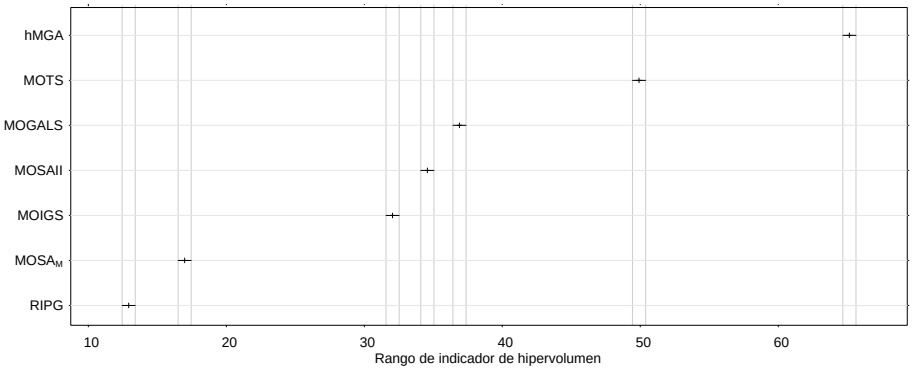


Figura A.2: Resultados de test de rango para el indicador de hipervolumen y $t = 100$ para los objetivos de $C_{\max}$ y tardanza total con intervalos de confianza al Tukey de 99 % (niveles de confianza ajustados al 95 %).

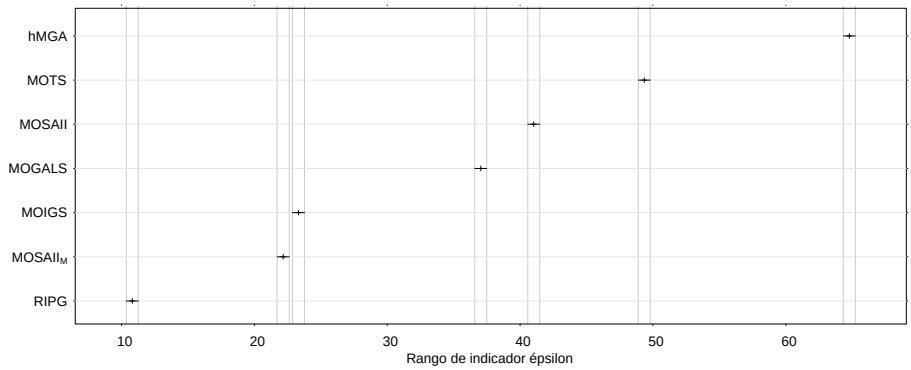


Figura A.3: Resultados de tests de rango para el indicador épsilon y $t = 200$ para los objetivos de $C_{\max}$ y tardanza total con intervalos de confianza al Tukey de 99 % (niveles de confianza ajustados al 95 %).

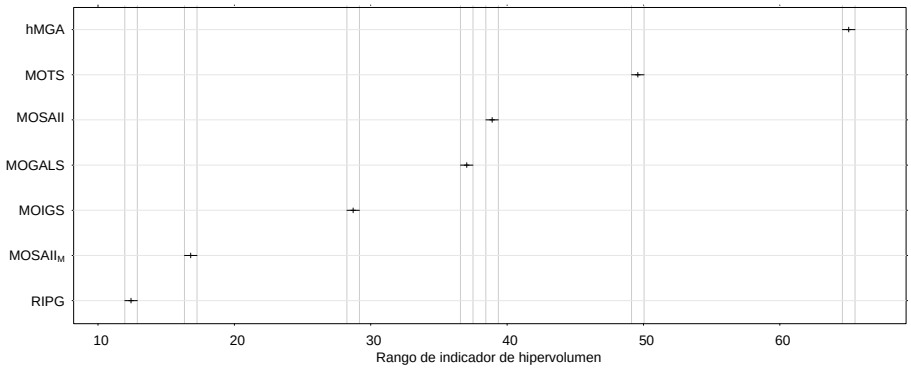


Figura A.4: Resultados de test de rango para el indicador de hipervolumen y $t = 200$ para los objetivos de $C_{\max}$ y tardanza total con intervalos de confianza al Tukey de 99 % (niveles de confianza ajustados al 95 %).

El comportamiento de los algoritmos es consistente para esta combinación de objetivos. El algoritmo RIPG se mantiene en la primera posición en todos los casos, tanto para el indicador I_H como para I_ε. La posición relativa del

resto de los algoritmos se mantiene constante también para los distintos valores de t. El algoritmo MOSAII$_M$ ocupa la segunda posición y el MOIGS la tercera. Mientras que, el algoritmo MOSAII ocupa la cuarta posición, el MOGALS la quinta, el MOTS la sexta y en la última posición se ubica el algoritmo hmga. Se puede observar como el algoritmo MOSAII obtiene resultados ligeramente peores para $t = 200$. Esto se debe a que este algoritmo no para por tiempo y, a diferencia de lo que ocurre con los demás algoritmos, obtiene los mismos resultados para cualquier valor de t. Otro algoritmo que empeora un poco (aunque se ubica en la misma posición relativa respecto de los demás) con $t = 200$ es el algoritmo MOGALS.

A.2. Resultados de tests de rangos para los experimentos con $C_{\max}$ y tardanza total

La siguiente tabla muestra los valores medios de rangos obtenidos del experimento realizado en el Capítulo 4 para $t = 100$ y $t = 200$ y para los dos indicadores utilizados. En este caso, la tabla presenta los resultados obtenidos para la combinación de objetivos $C_{\max}$ y tardanza total. Los mejores valores para cada indicador están señalados en negrita.

	Tiempo	100			200	
#	Algoritmo	I_H	I_ε	Algoritmo	I_H	I_ε
1	RIPG	**11,1600**	**10,1682**	RIPG	**10,6573**	**9,5091**
2	MOSAII$_M$	17,3164	19,5255	MOSAII$_M$	17,2182	19,8918
3	MOIGS	27,1364	25,3055	MOIGS	25,2300	22,5473
4	MOSAII	33,9155	33,6055	MOSAII	36,7764	36,6355
5	MOGALS	46,3091	44,8609	MOGALS	46,3373	45,0255
6	MOTS	50,8582	49,5527	MOTS	51,0373	48,8455
7	hMGA	64,8655	64,3255	hMGA	64,6591	64,2091

Tabla A.2: Resultados de rangos medios para los objetivos $C_{\max}$ y tardanza total y criterios de parada $t = 100$ y $t = 200$. Los métodos están ordenados de acuerdo a I_H.

A continuación exponemos las gráficas representando los resultados de tests de rangos para uno de los indicadores, un valor de t y los objetivos de $C_{\max}$ y tardanza total.

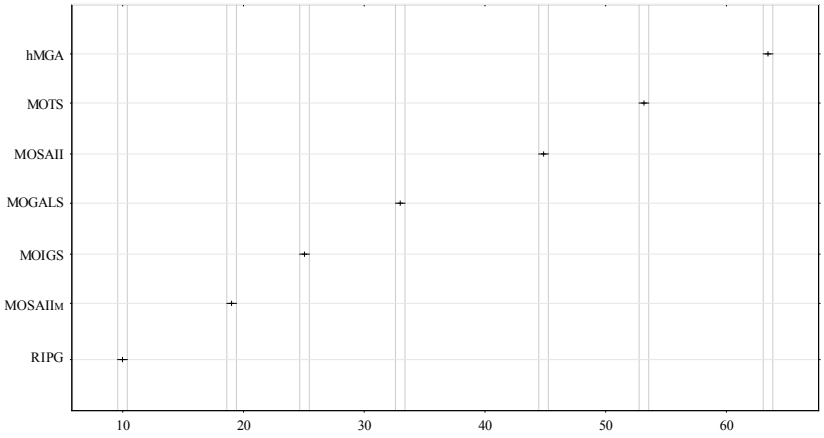


Figura A.5: Resultados de tests de rango para el indicador épsilon y $t = 100$ para los objetivos de $C_{\max}$ y tardanza total con intervalos de confianza de Tukey al 99 %(niveles de confianza ajustados al 95 %).

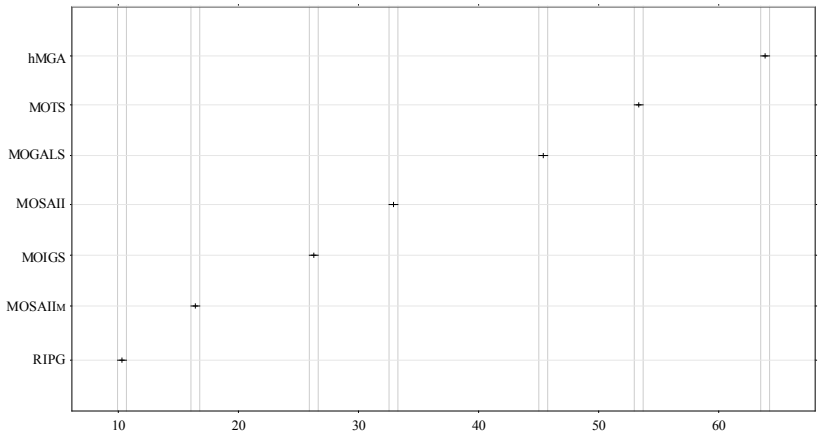


Figura A.6: Resultados de test de rango para el indicador de hipervolumen y $t = 100$ para los objetivos de $C_{\max}$ y tardanza total con intervalos de confianza de Tukey al 99 %(niveles de confianza ajustados al 95 %).

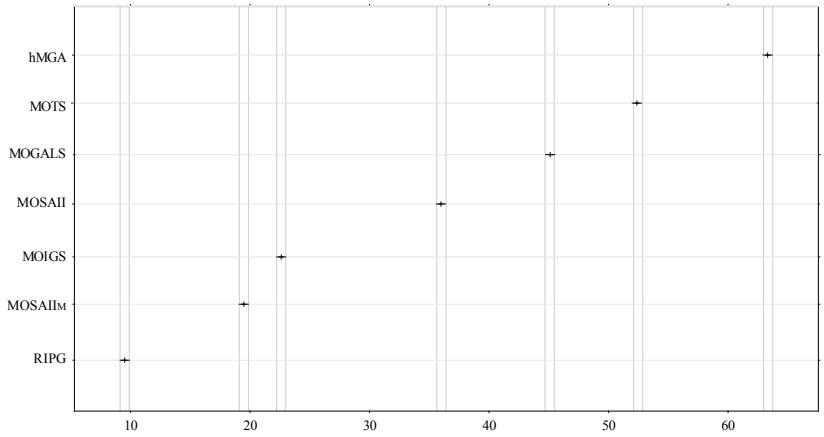


Figura A.7: Resultados de tests de rango para el indicador épsilon y $t = 200$ para los objetivos de $C_{\max}$ y tardanza total con intervalos de confianza de Tukey al 99 %(niveles de confianza ajustados al 95 %).

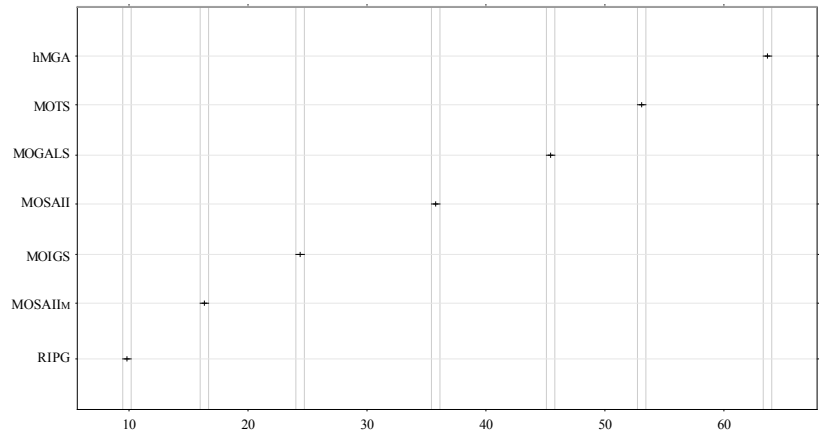


Figura A.8: Resultados de test de rango para el indicador de hipervolumen y $t = 200$ para los objetivos de $C_{\max}$ y tardanza total con intervalos de confianza de Tukey al 99 %(niveles de confianza ajustados al 95 %).

Para los experimentos con los objetivos de $C_{\max}$ y tardanza total los resultados son similares a los experimentos con $C_{\max}$ y tiempo total de flujo. El algoritmo RIPG obtiene la primera posición para todos los tests, quedando en segundo lugar el MOSAII$_M$, en tercer lugar el MOIGS, en cuarto lugar el MOSAII, en quinta posición el MOGALS, y finalmente ocupando la sexta y séptima posición están el algoritmo MOTS y el hMGA. Existen diferencias pequeñas en los resultados para $t = 100$ y $t = 200$, aunque estas diferencias no implican cambio en la posición relativa de ningún algoritmo.

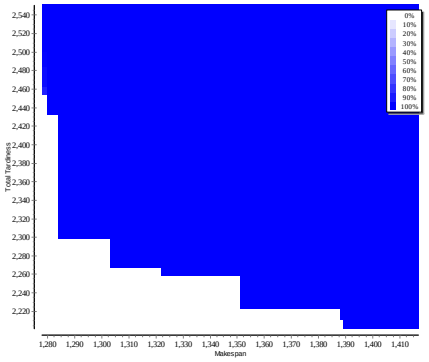
APÉNDICE B

GRÁFICAS DE ATTAINMENT FUNCTIONS PARA LOS EXPERIMENTOS DE RIPG PARA EL PROBLEMA PFSP SIN TIEMPOS DE CAMBIO

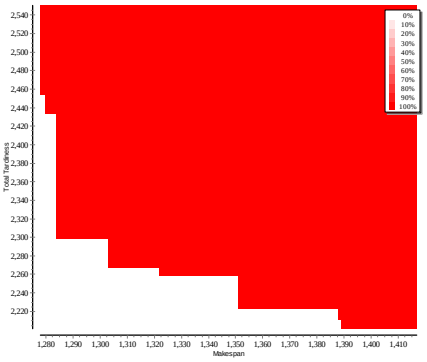
En este anexo presentamos las gráficas de Attainment function que completan los resultados de los experimentos del Capítulo 4. Para generar cada gráfica de Attainment function se ha efectuado un experimento en el cual, para una única instancia, algoritmo y combinación de objetivo se han generado 100 réplicas. En todos los casos el criterio de parada ha sido de $t = 200$. Debido al enorme esfuerzo que representa llegar a generar cada uno de las gráficas no hemos generado una gráfica de Attainment function para cada instancia. En su lugar, hemos seleccionado la primera de las instancias de cada grupo de instancias, con lo cual hemos obtenido un total de 11 gráficas de attainment function.

Cada gráfica de Attainment function representa el resultado de un algoritmo para una instancia. La principal desventaja que presentan las EAFs es que no permiten la comparación de resultados. Por este motivo hemos utilizado las gráficas diferenciales de Attainment función (*Differential Empirical At-*

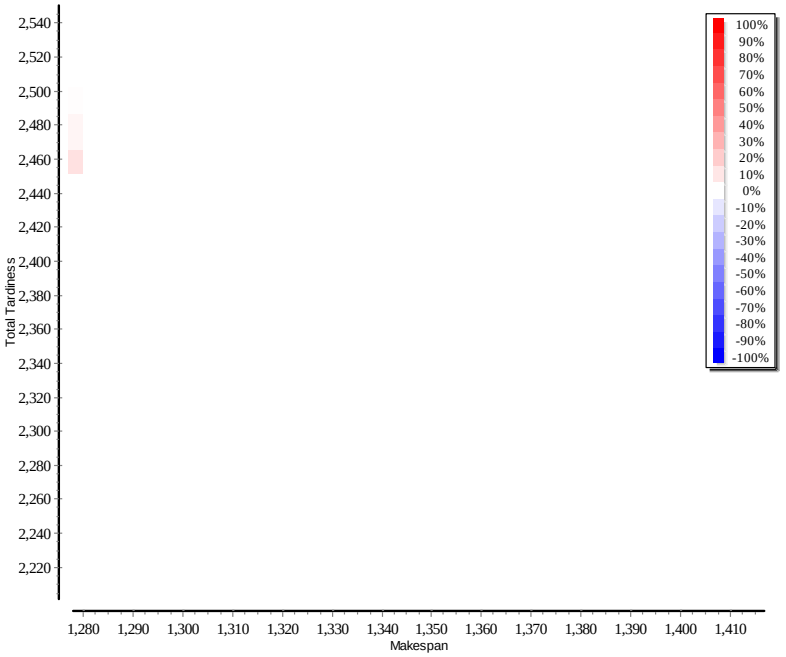
tainment Functions ó DEAF). Estas gráficas representan una diferencia entre dos EAF, con lo cual permiten observar en que partes del espacio de objetivo un algoritmo es capaz de superar consistentemente a otro algoritmo. Para interpretar correctamente estas gráficas es necesario tener presente que representan una resta de dos gráficas, con lo cual, en principio el color blanco (o la falta de color) indica que no existe diferencia entre los dos algoritmos que se está representando. Por otro lado, las zonas coloreadas con un color indican que el algoritmo representado con dicho color ha superado (en esa zona) consistentemente al otro algoritmo en los experimentos.



(a) EAF para MOSAII$_M$

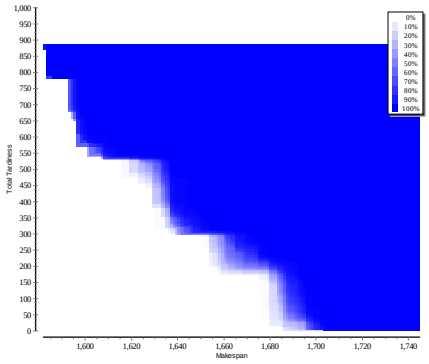


(b) EAF para RIPG

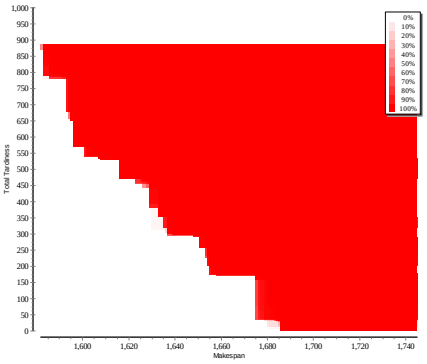


(c) Diff-EAF

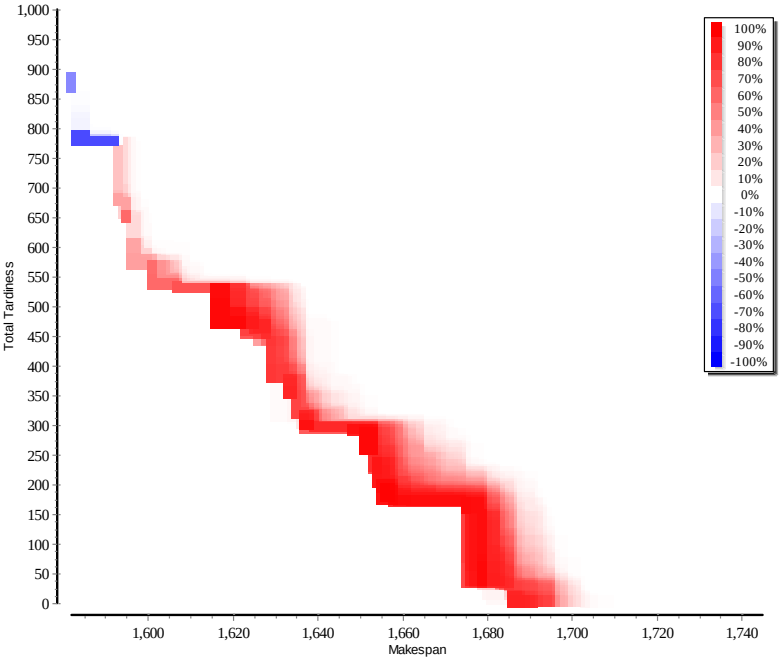
Figura B.1: Gráficas de *attainment function* y *differential attainment function* para la instancia Ta001 con 20 trabajos y 5 máquinas. $t = 100$, $C_{\text{máx}}$ y tardanza total. Algoritmos MOSAII$_M$ y RIPG.



(a) EAF para MOSAII$_M$

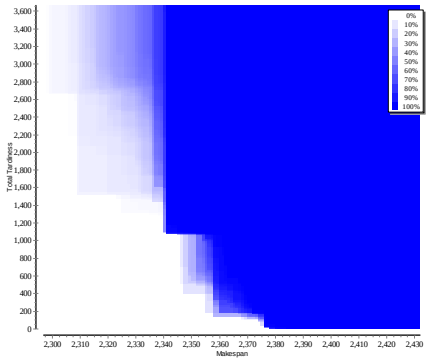


(b) EAF para RIPG

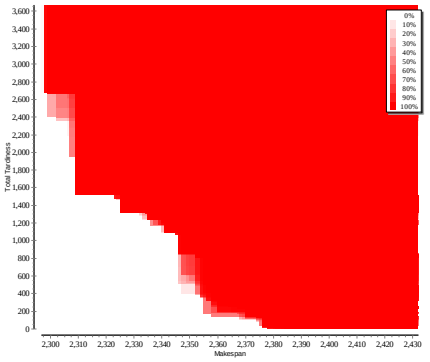


(c) Diff-EAF

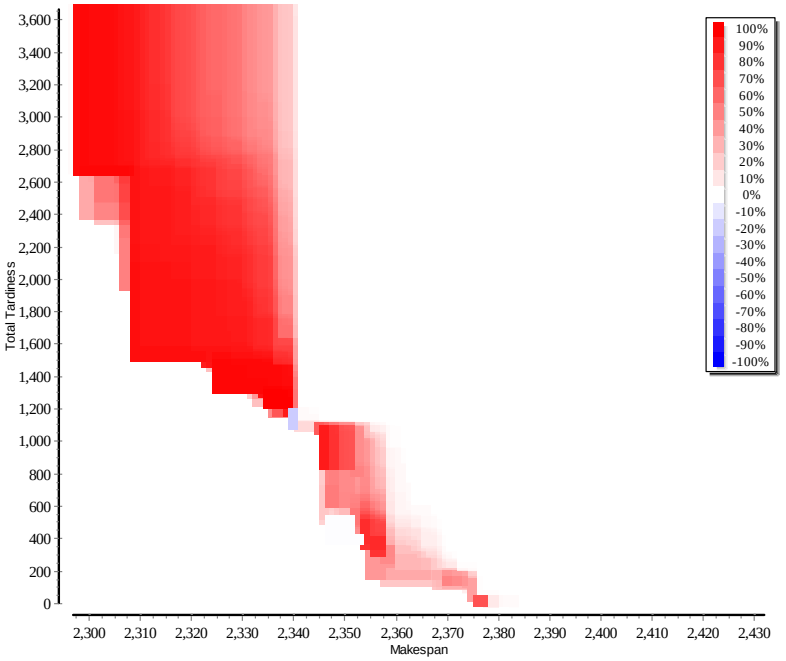
Figura B.2: Gráficas de *attainment function* y *differential attainment function* para la instancia Ta011 con 20 trabajos y 10 máquinas. $t = 100$, $C_{\max}$ y tardanza total. Algoritmo MOSAII$_M$ y RIPG.



(a) EAF para MOSAII$_M$

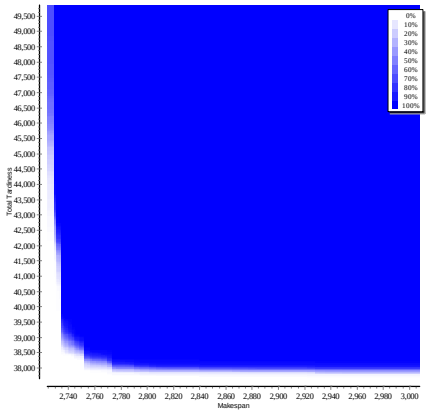


(b) EAF para RIPG

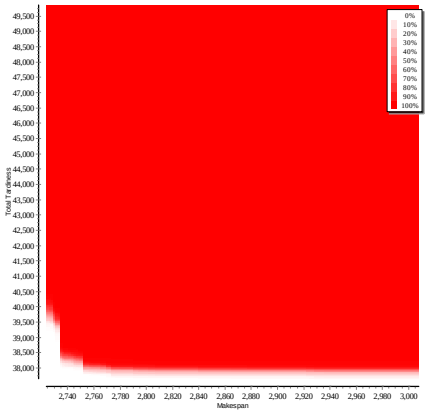


(c) Diff-EAF

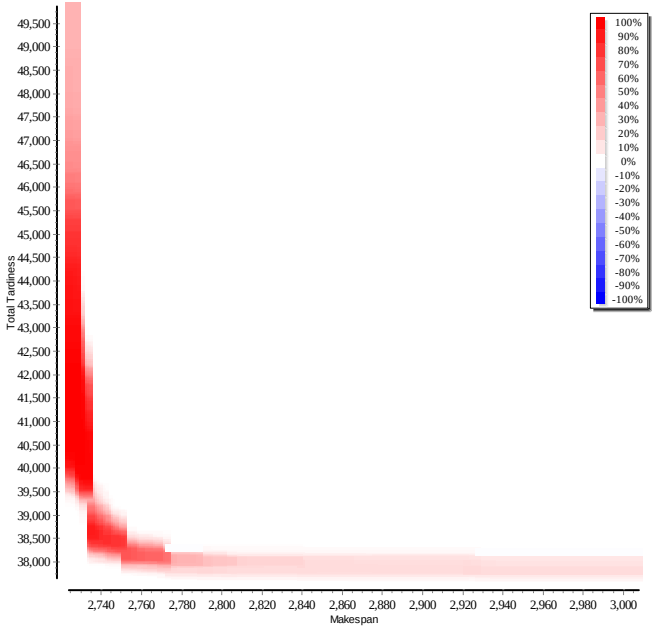
Figura B.3: Gráficas de *attainment function* y *differential attainment function* para la instancia Ta021 con 20 trabajos y 20 máquinas. $t = 100$, $C_{\text{máx}}$ y tardanza total. Algoritmos MOSAII$_M$ y RIPG.



(a) EAF para MOSAII$_M$

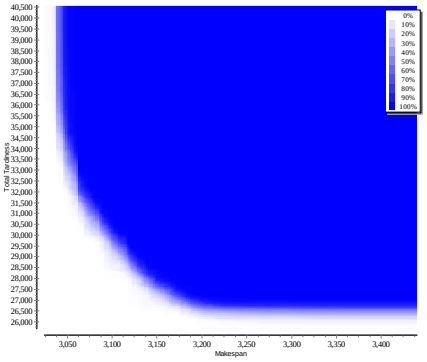


(b) EAF para RIPG

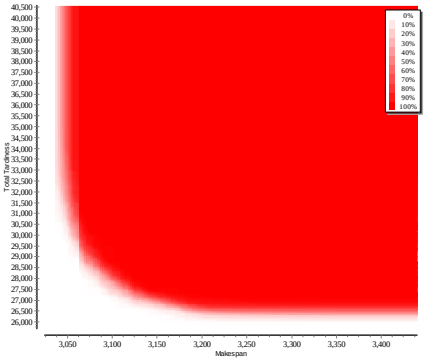


(c) Diff-EAF

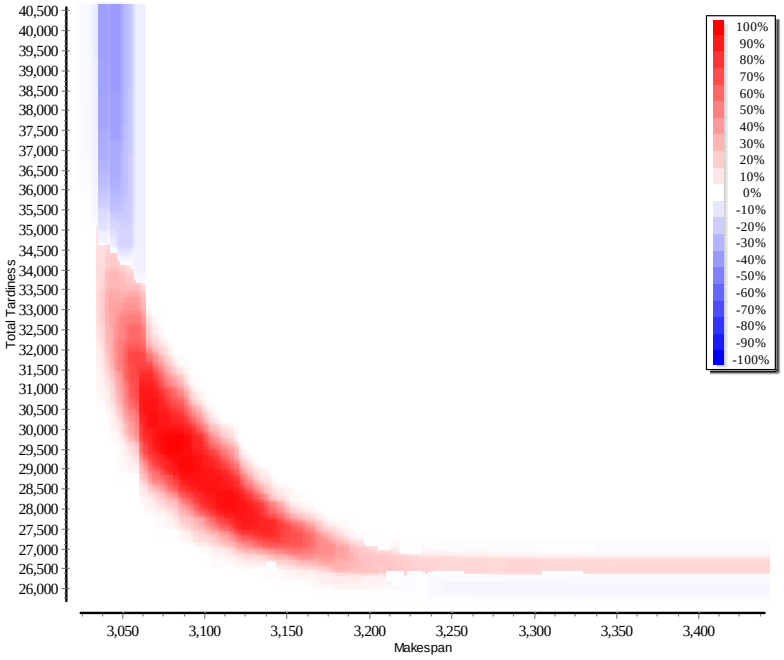
Figura B.4: Gráficas de *attainment function* y *differential attainment function* para la instancia Ta031 con 50 trabajos y 5 máquinas. $t = 100$, $C_{\text{máx}}$ y tardanza total. Algoritmos MOSAII$_M$ y RIPG.



(a) EAF para MOSAII$_M$

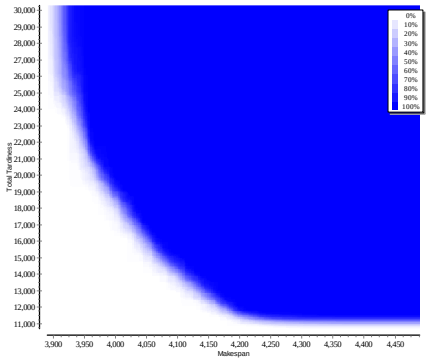


(b) EAF para RIPG

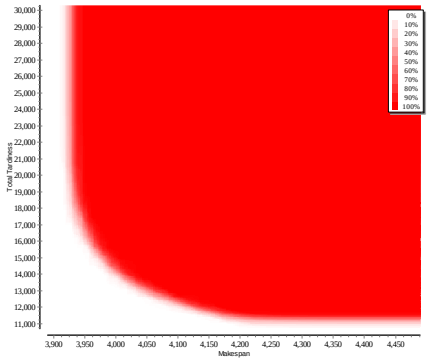


(c) Diff-EAF

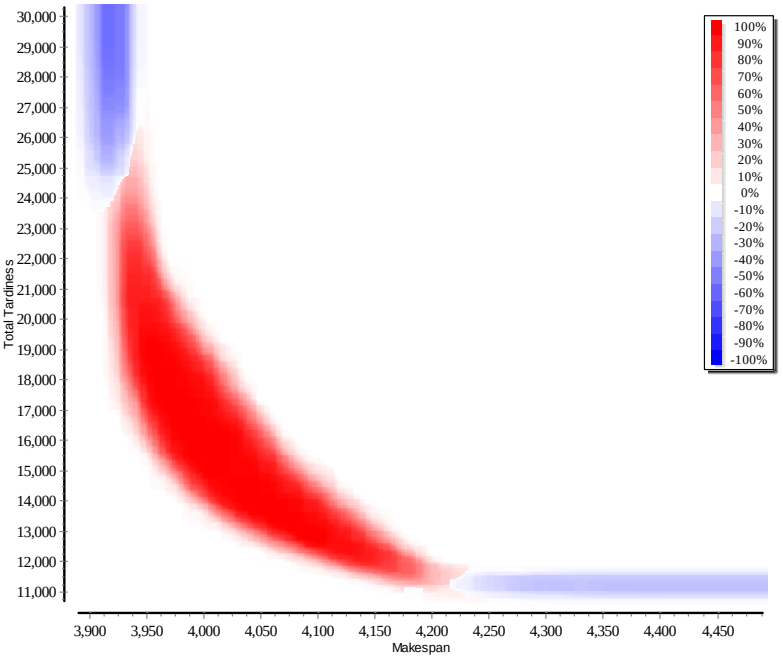
Figura B.5: Gráficas de *attainment function* y *differential attainment function* para la instancia Ta041 con 50 trabajos y 10 máquinas. $t = 100$, $C_{\max}$ y tardanza total. Algoritmo MOSAII$_M$ y RIPG.



(a) EAF para MOSAII$_M$

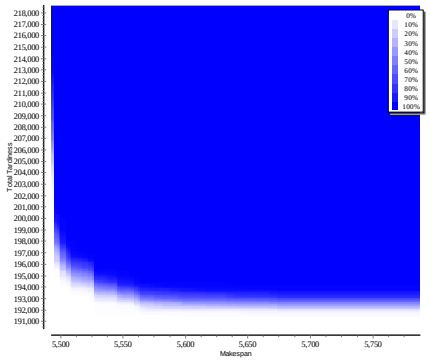


(b) EAF para RIPG

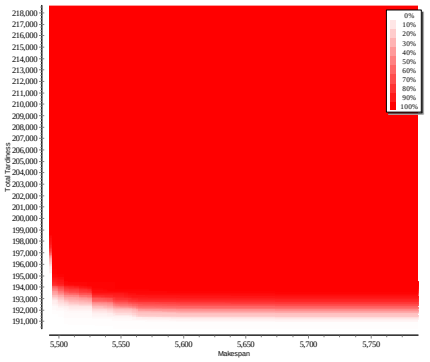


(c) Diff-EAF

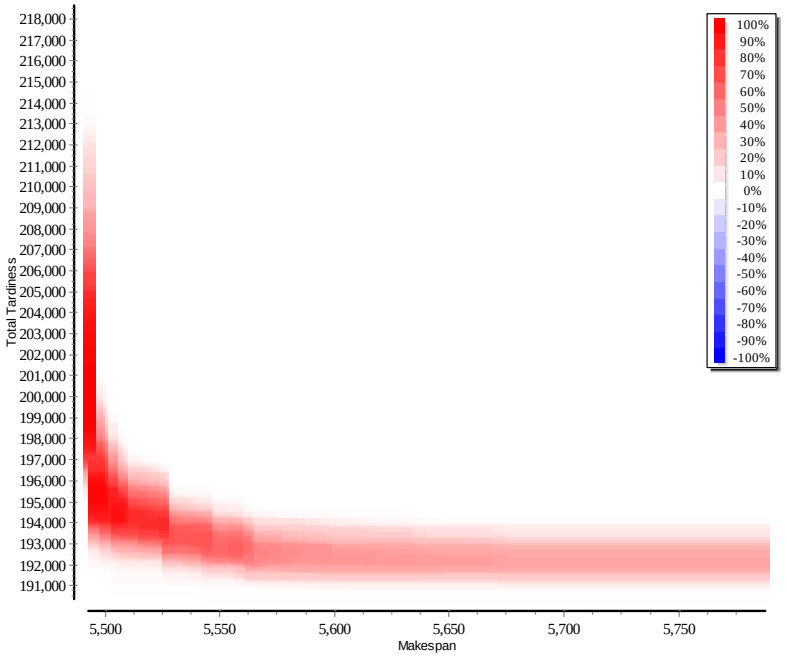
Figura B.6: Gráficas de *attainment function* y *differential attainment function* para la instancia Ta051 con 50 trabajos y 20 máquinas. $t = 100$, $C_{\max}$ y tardanza total. Algoritmo MOSAII$_M$ y RIPG.



(a) EAF para MOSAII$_M$

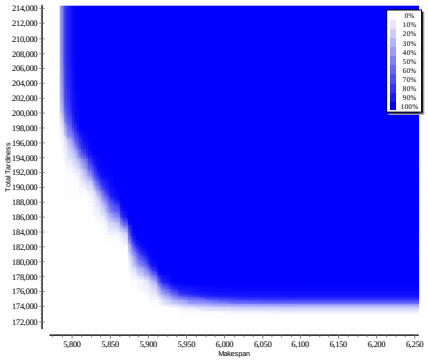


(b) EAF para RIPG

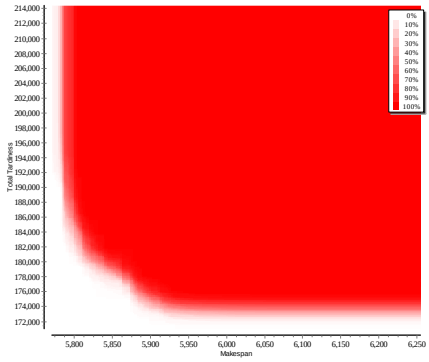


(c) Diff-EAF

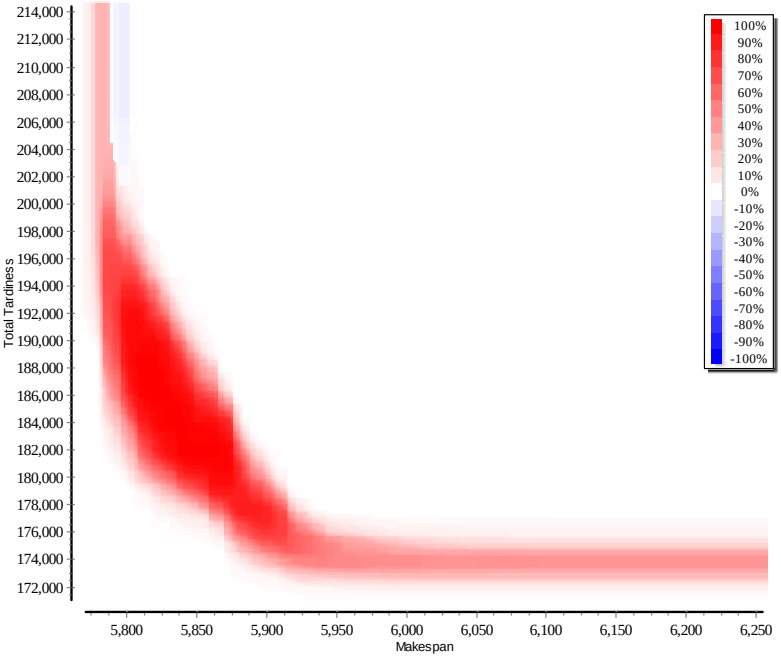
Figura B.7: Gráficas de *attainment function* y *differential attainment function* para la instancia Ta061 con 100 trabajos y 5 máquinas. $t = 100$, $C_{\text{máx}}$ y tardanza total. Algoritmo MOSAII$_M$ y RIPG.



(a) EAF para MOSAII$_M$

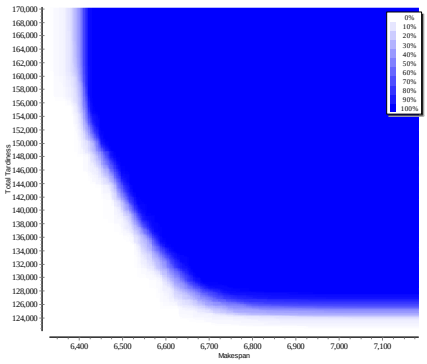


(b) EAF para RIPG

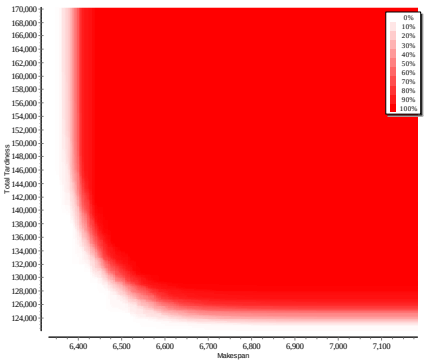


(c) Diff-EAF

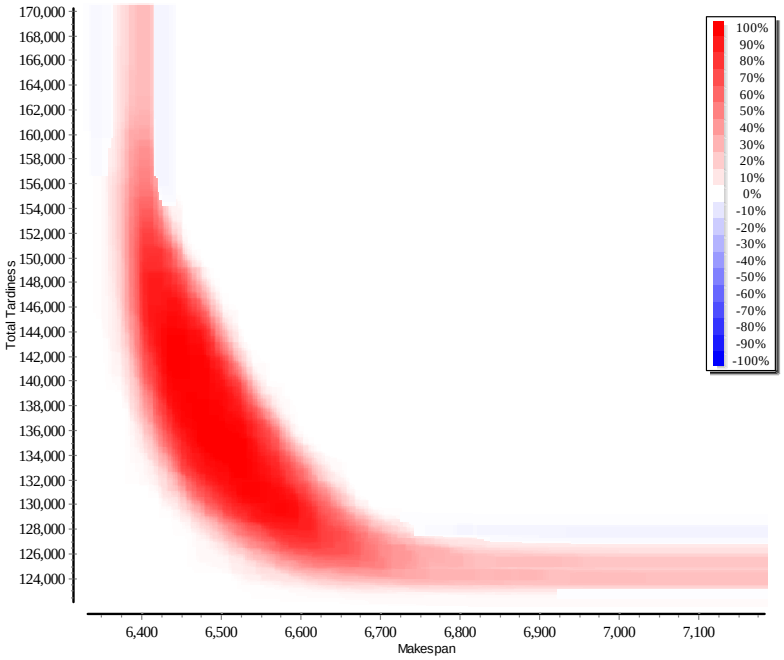
Figura B.8: Gráficas de *attainment function* y *differential attainment function* para la instancia Ta071 con 100 trabajos y 10 máquinas. $t = 100$, $C_{\max}$ y tardanza total. Algoritmo MOSAII$_M$ y RIPG.



(a) EAF para MOSAII$_M$

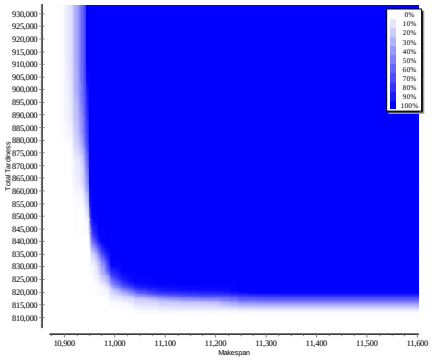


(b) EAF para RIPG

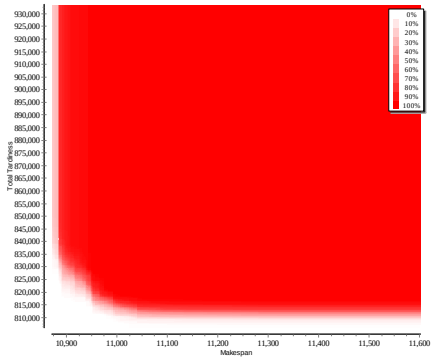


(c) Diff-EAF

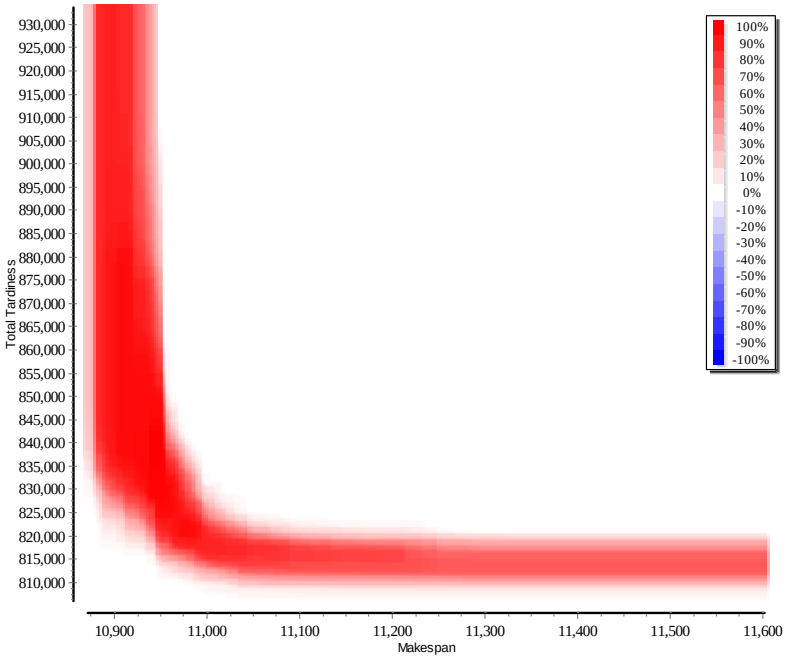
Figura B.9: Gráficas de *attainment function* y *differential attainment function* para la instancia Ta081 con 100 trabajos y 20 máquinas. $t = 100$, $C_{\max}$ y tardanza total. Algoritmo MOSAII$_M$ y RIPG.



(a) EAF para MOSAII$_M$

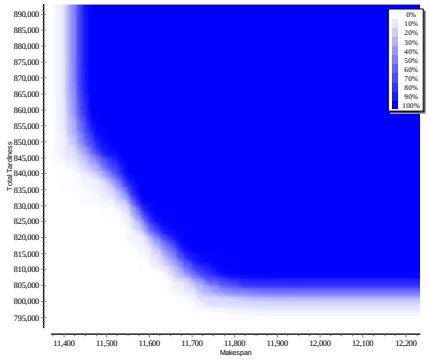


(b) EAF para RIPG

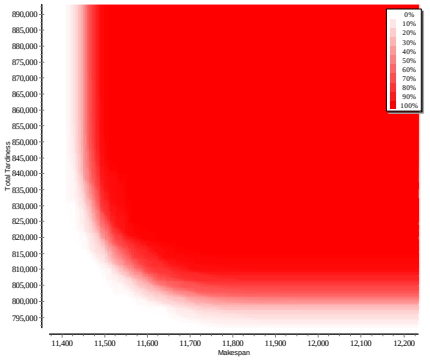


(c) Diff-EAF

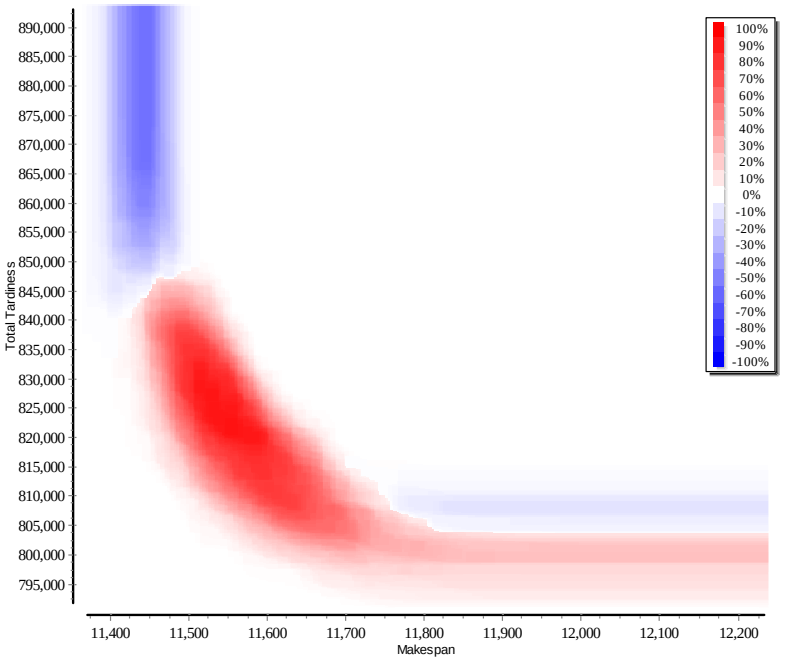
Figura B.10: Gráficas de *attainment function* y *differential attainment function* para la instancia Ta091 con 200 trabajos y 10 máquinas. $t = 100$, $C_{\max}$ y tardanza total. Algoritmo MOSAII$_M$ y RIPG.



(a) EAF para MOSAII$_M$



(b) EAF para RIPG



(c) Diff-EAF

Figura B.11: Gráficas de *attainment function* y *differential attainment function* para la instancia Ta101 con 200 trabajos y 20 máquinas. $t = 100$, $C_{\max}$ y tardanza total. Algoritmo MOSAII$_M$ y RIPG.

APÉNDICE C

RESULTADOS DE TESTS ANOVA Y TEST DE RANGOS PARA EL PROBLEMA DEL TALLER DE FLUJO DE PERMUTACIÓN MULTI-OBJETIVO CON TIEMPOS DE CAMBIO DEPENDIENTES DE LA SECUENCIA

En este anexo presentamos los resultados de los tests ANOVA y los tests de rangos para los experimentos del Capítulo 5. En éste contrastamos el rendimiento de los 10 mejores algoritmos del Capítulo 3, contra otros algoritmos surgidos en los últimos tiempos y contra la adaptación nuestro algoritmo RIPG propuesto en el Capítulo 4 al problema del taller de flujo con tiempos de cambio.

Hemos separado los resultados de acuerdo a la combinación de objetivos, en primer lugar presentaremos los resultados para $C_{\max}$ y tiempo total de flujo y luego los resultados para $C_{\max}$ y tardanza total ponderada.

C.1. Resultados de tests de rangos para la combinación de objetivos $C_{\max}$ y tiempo total de flujo

En esta sección mostramos y analizamos los resultados de tests de rangos o Friedman para los experimentos realizados para el problema del taller de flujo de permutación multi-objetivo con tiempos de cambio dependientes de la secuencia.

Es de destacar que los tests de ranking no tienen en cuenta las diferencias reales en los indicadores, y en consecuencia los resultados varían, a veces de manera significativa. Por ejemplo, en los tests de rangos los algoritmos PESA y PESAII se muestran entre los mejores, mientras que los algoritmos MOSAII y MOSAII$_M$ no llegan a estar entre los cinco mejores métodos. Esto sólo indica que el ranking medio para PESA y PESAII es mejor al de MOSAII Y MOSAII$_M$. Sin embargo no implica que los indicadores de hipervolumen o épsilon en promedio sean peores.

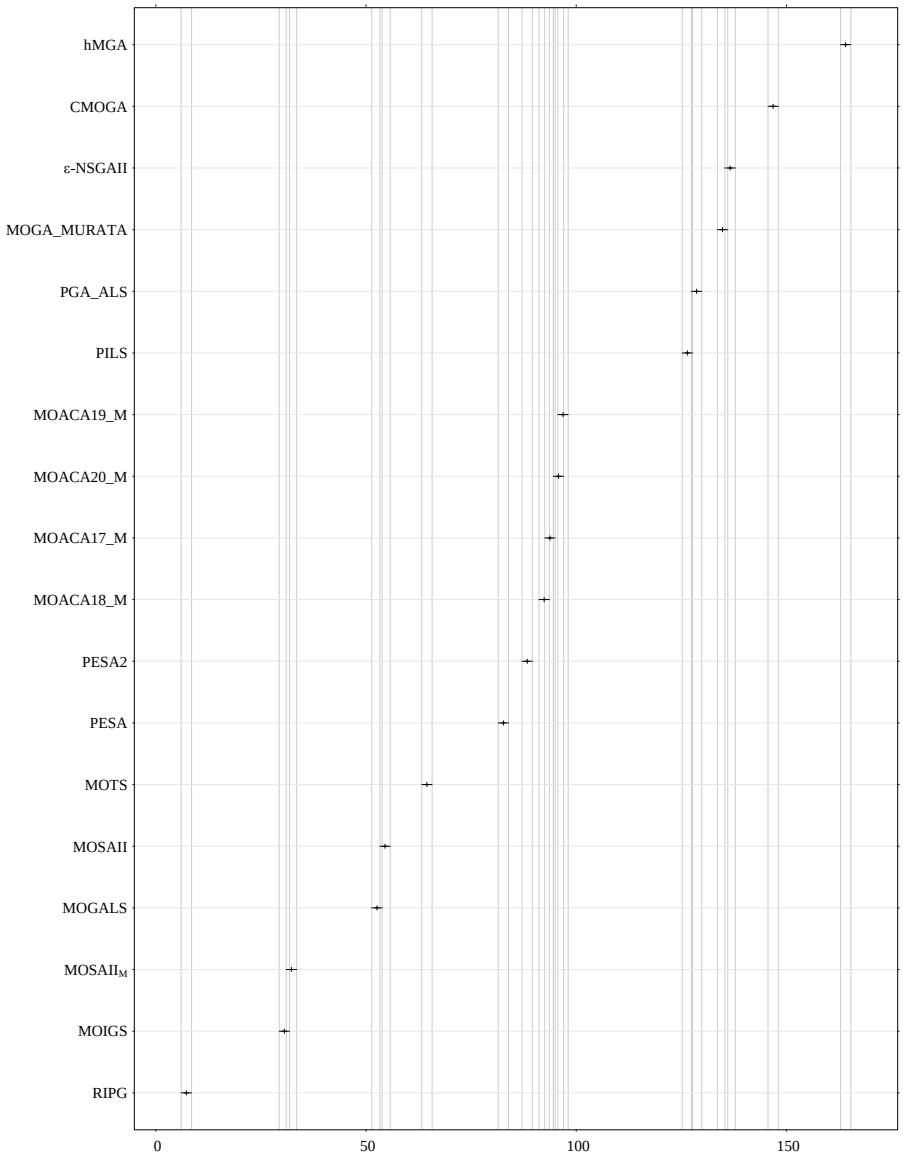


Figura C.1: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD50, variable respuesta indicador épsilon y $t = 150$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

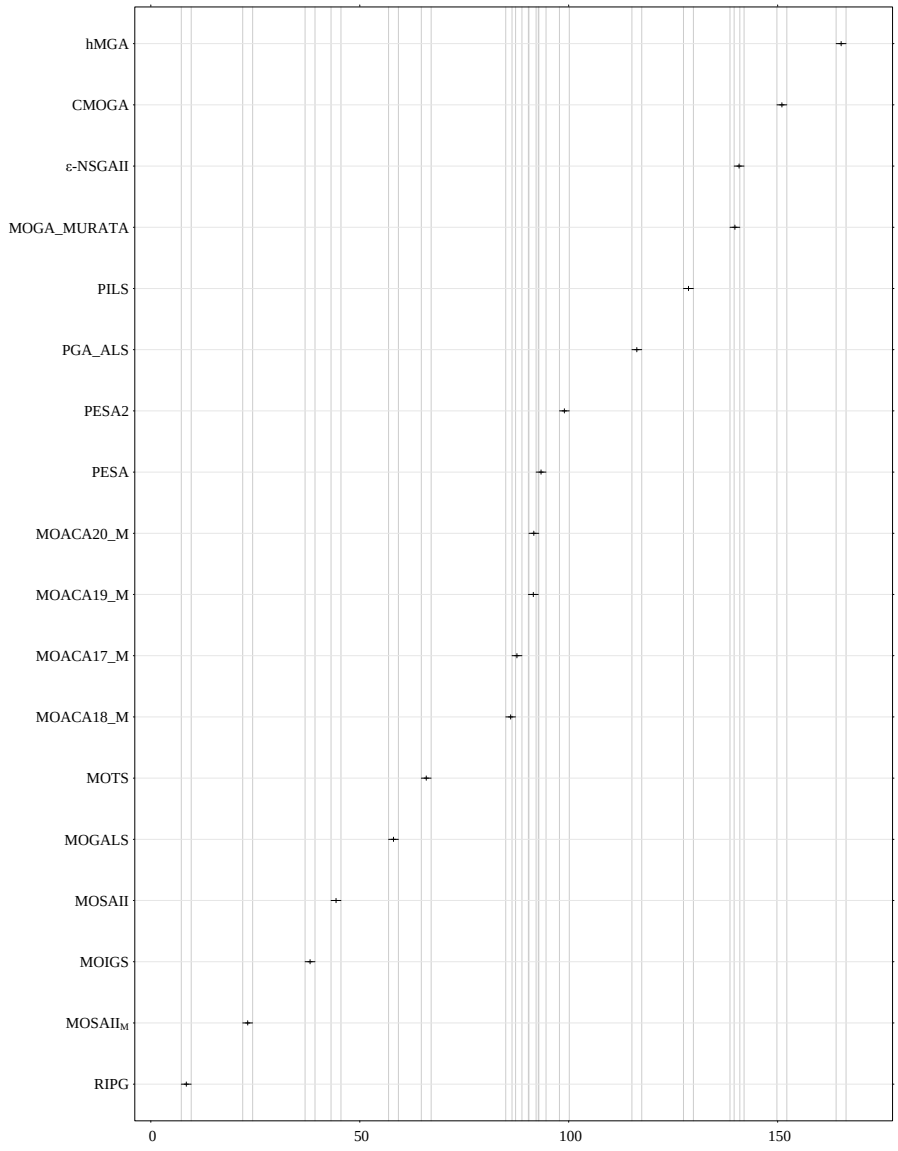


Figura C.2: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD50, variable respuesta indicador de hipervolumen y $t = 150$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

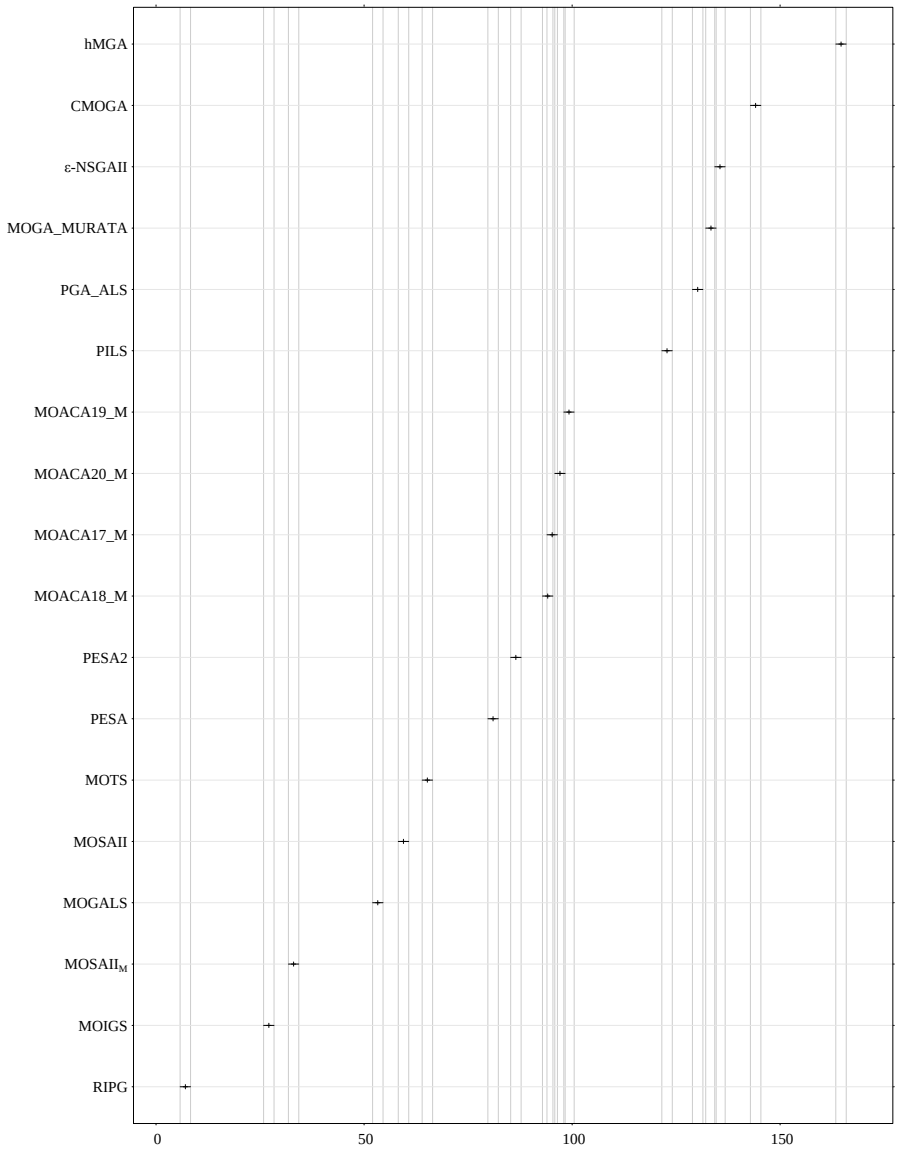


Figura C.3: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD50, variable respuesta indicador épsilon y $t = 200$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

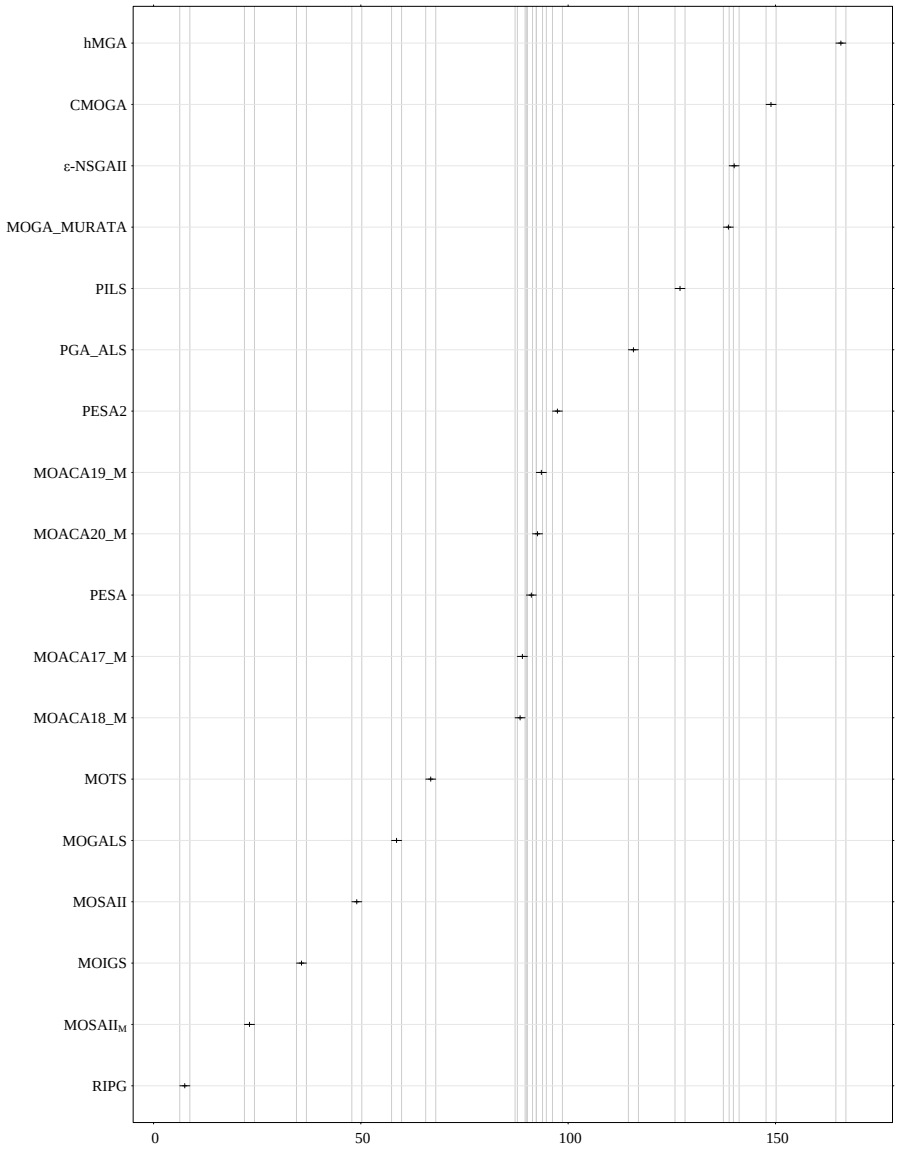


Figura C.4: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD50, variable respuesta indicador de hipervolumen y $t = 200$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

C.2. Resultados de tests de rangos para la combinación de objetivos $C_{\max}$ y tardanza total ponderada

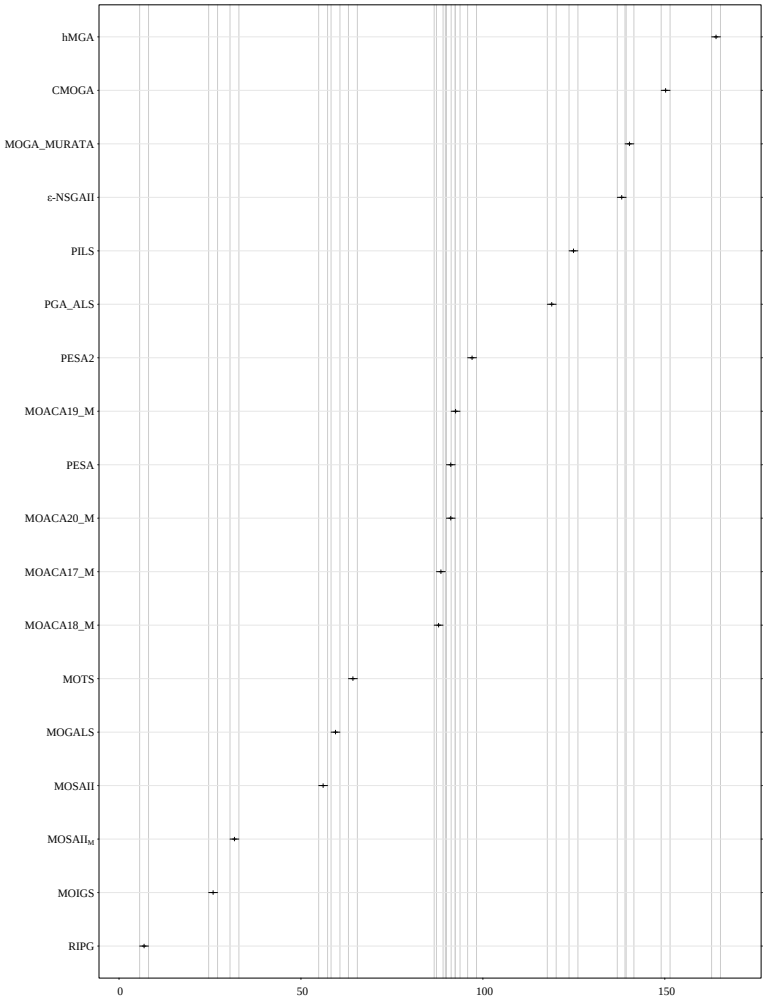


Figura C.9: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD50, variable respuesta indicador épsilon y $t = 150$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

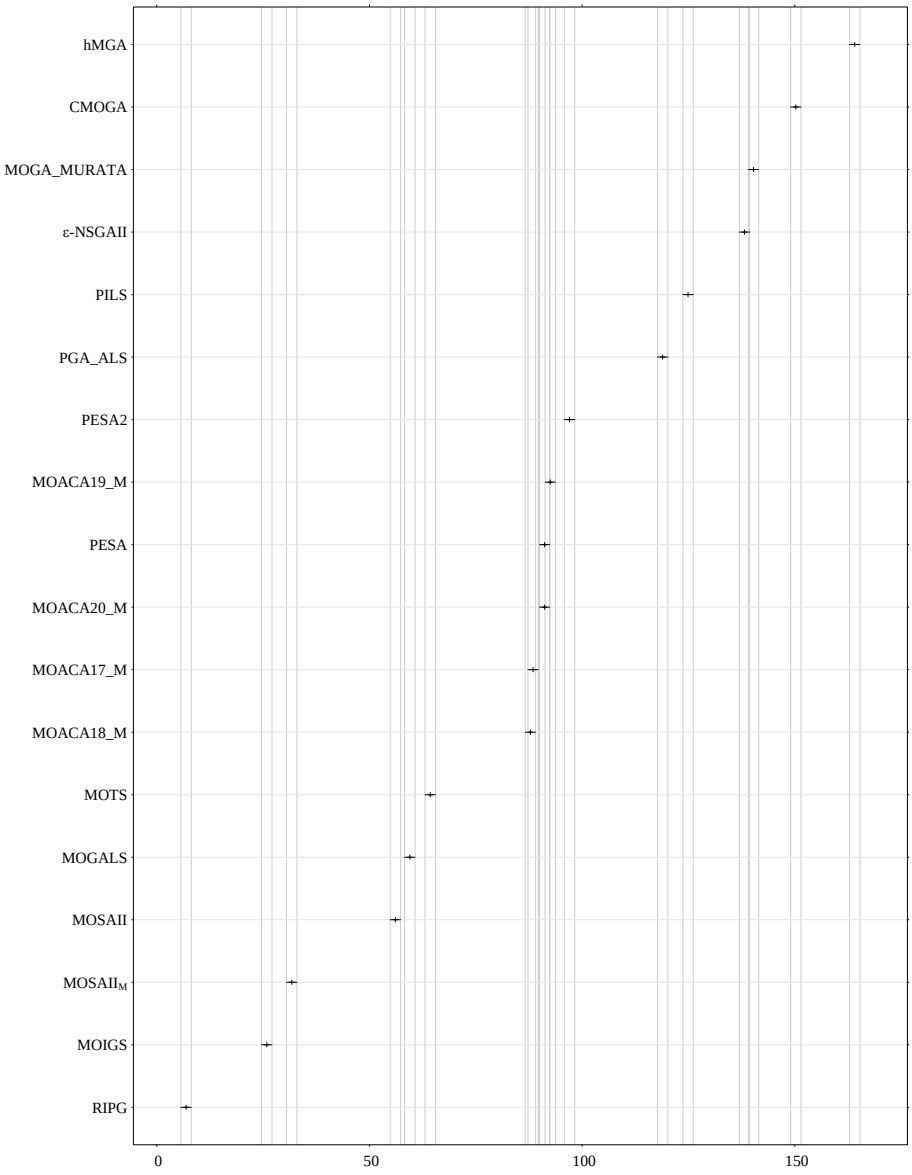


Figura C.10: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD50, variable respuesta indicador de hipervolumen y $t = 150$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

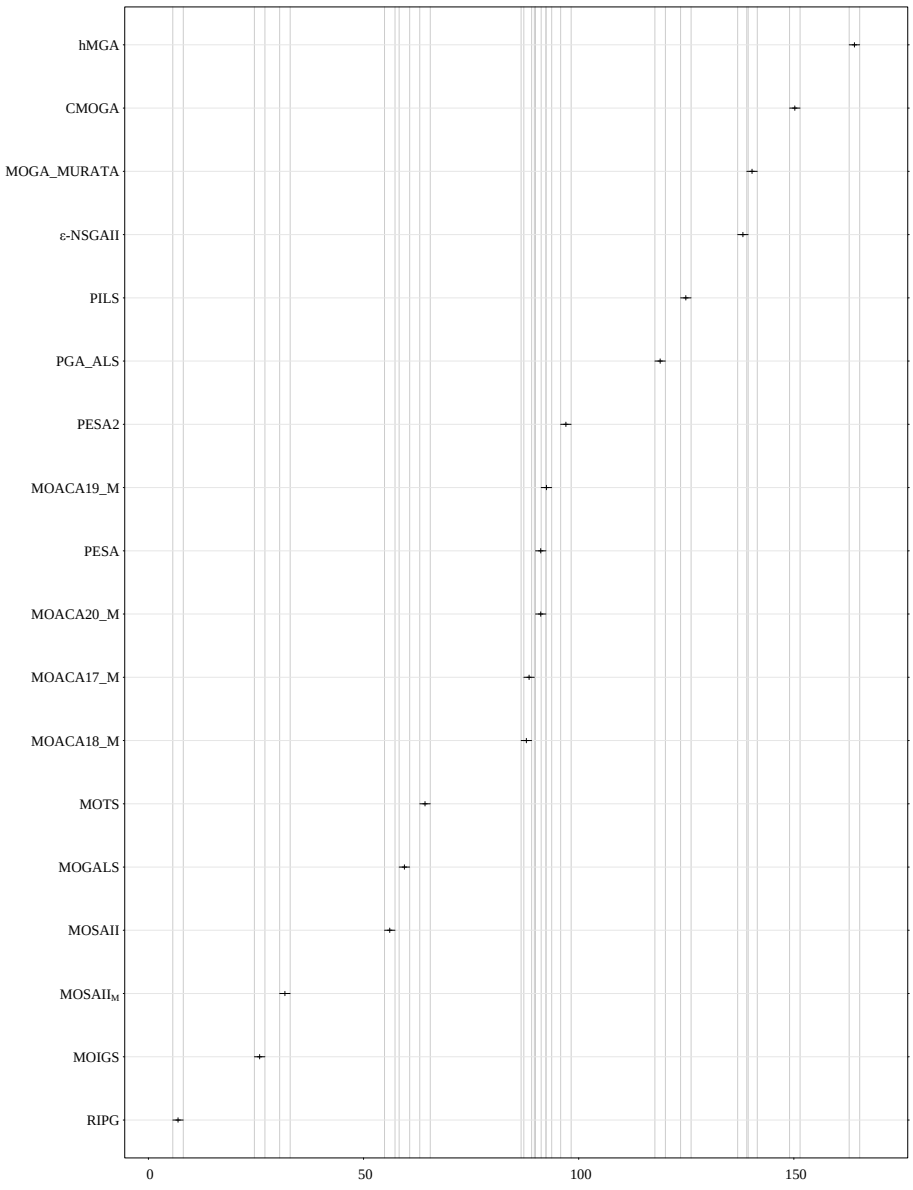


Figura C.11: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD125, variable respuesta indicador épsilon y $t = 150$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

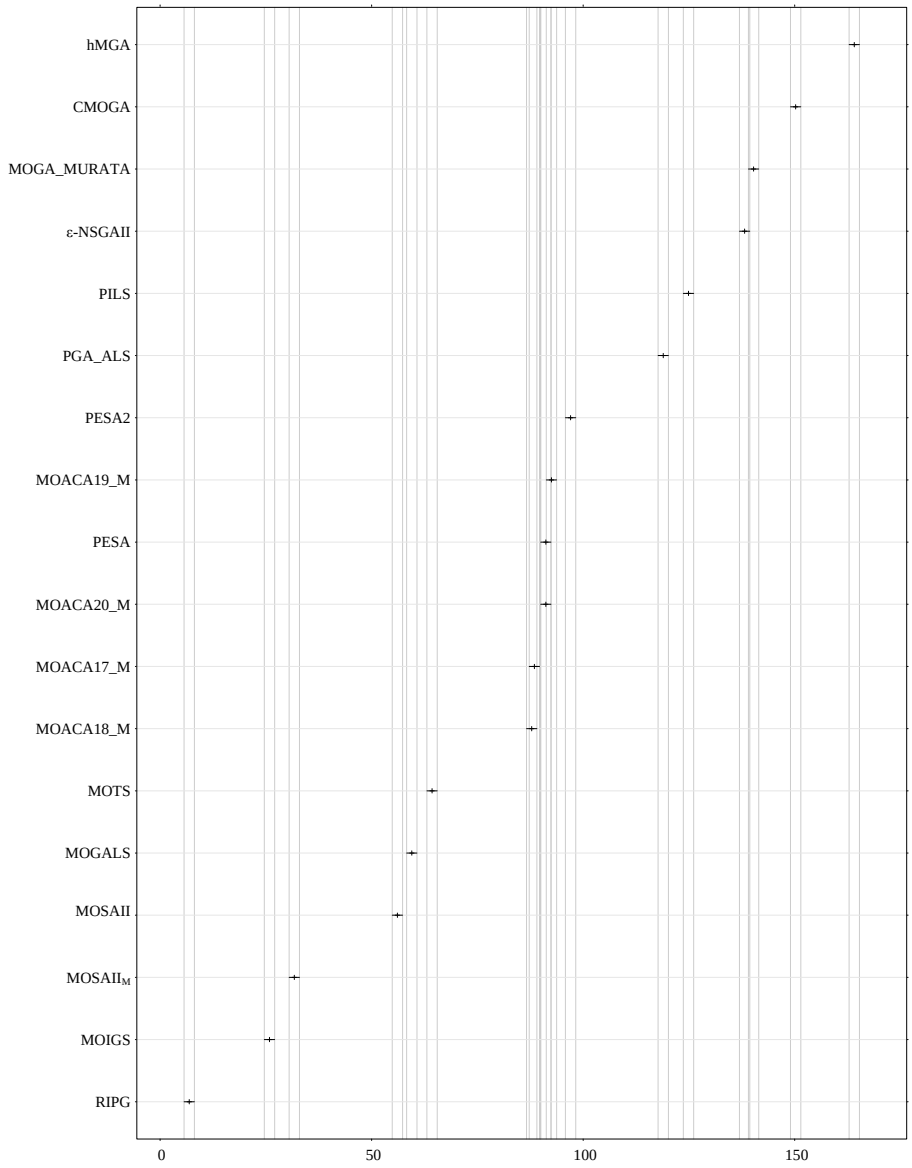


Figura C.12: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD125, variable respuesta indicador de hipervolumen y $t = 150$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

En el caso de los tests de rangos, para el grupo de instancias SSD50 y el criterio de parada $t = 150$, el orden de los algoritmos cambia un poco respecto al de los tests ANOVA. En primer lugar se ubica el algoritmo RIPG, en la segunda posición está el algoritmo MOIGS, mientras que el MOSAII$_M$ ocupa el tercer lugar (aunque no se muestran diferencias estadísticamente significativas con el MOIGS). Luego, en cuarto lugar está el algoritmo MOGALS y en quinto lugar queda el algoritmo MOSAII.
Si bien los cinco primeros algoritmos siguen siendo los mismos, el único que mantiene su posición respecto a los tests ANOVA es el RIPG.
Para el caso del indicador I_H, con el mismo grupo de instancias y valor de t el orden de los algoritmos vuelve a cambiar un poco. El primer lugar se encuentra el algoritmo RIPG, mientras que el segundo lugar es ocupado por MOSAII$_M$, en el tercer lugar está MOIGS, luego viene MOSAII, quedando MOGALS en la quinta posición.

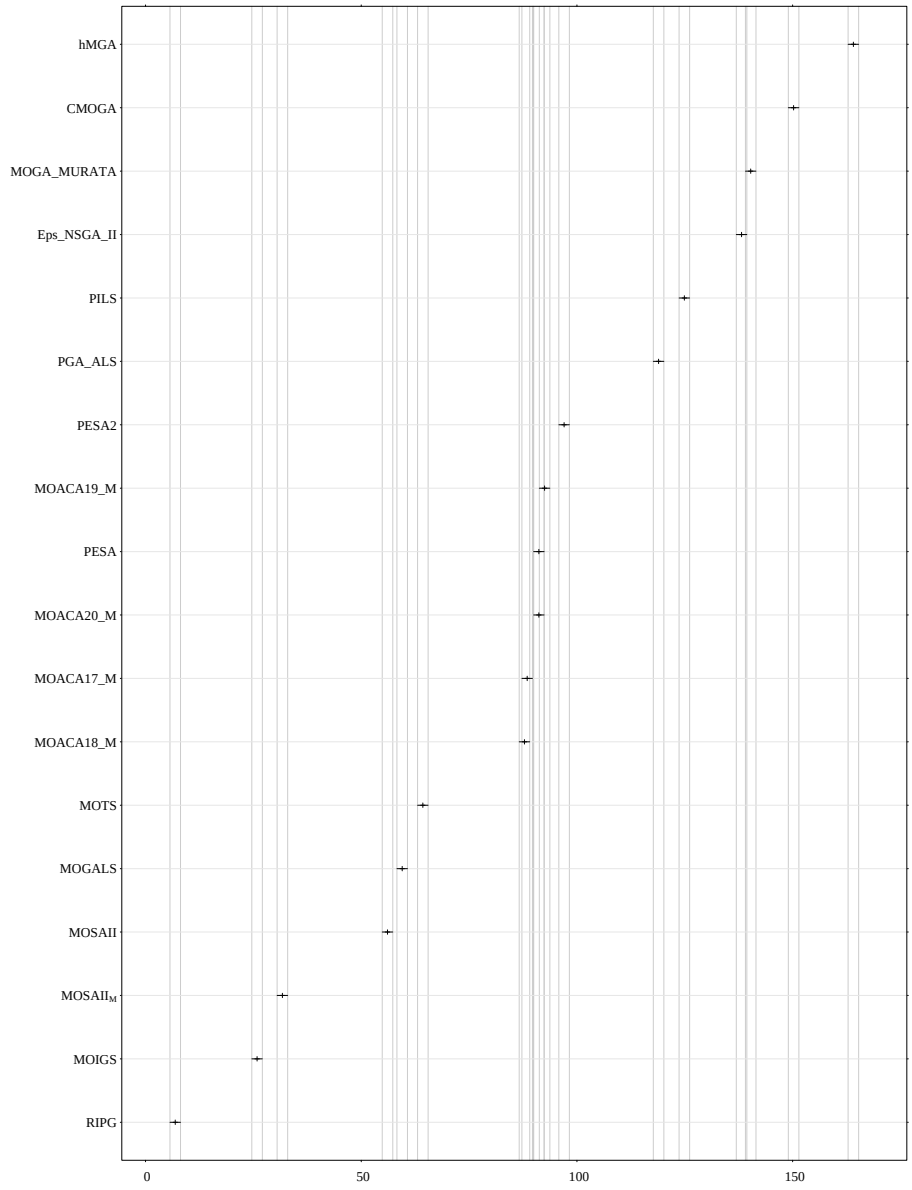


Figura C.13: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD50, variable respuesta indicador épsilon y $t = 200$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

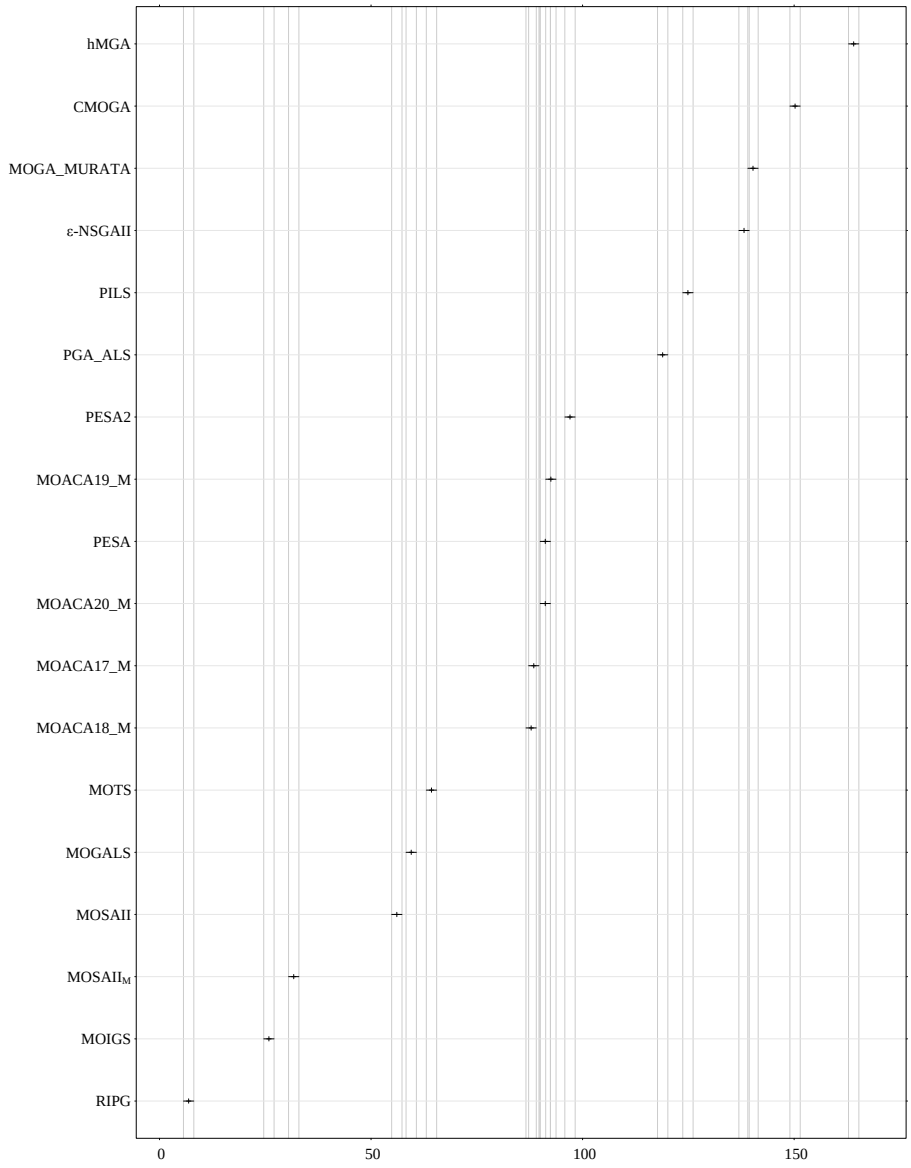


Figura C.14: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD50, variable respuesta indicador de hipervolumen y $t = 200$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

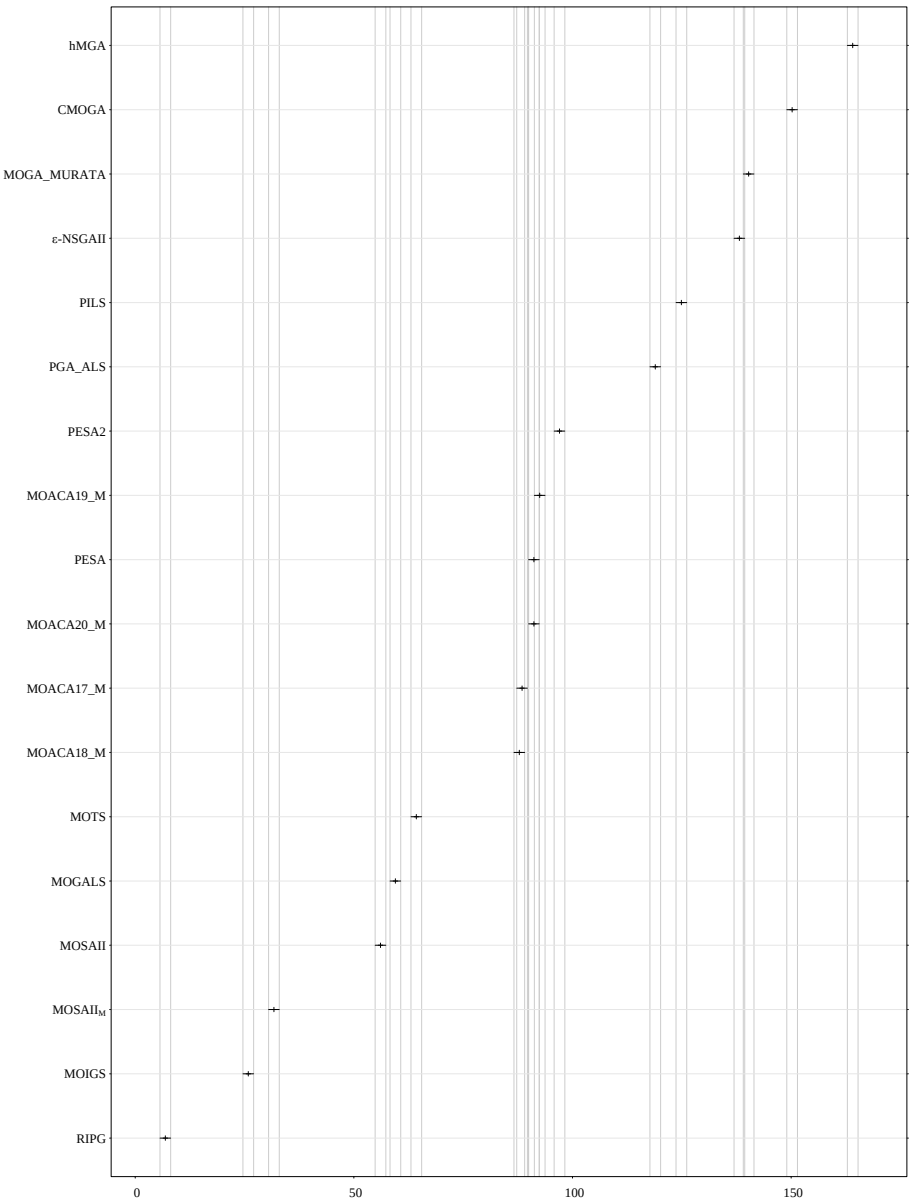


Figura C.15: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD125, variable respuesta indicador épsilon y $t = 200$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

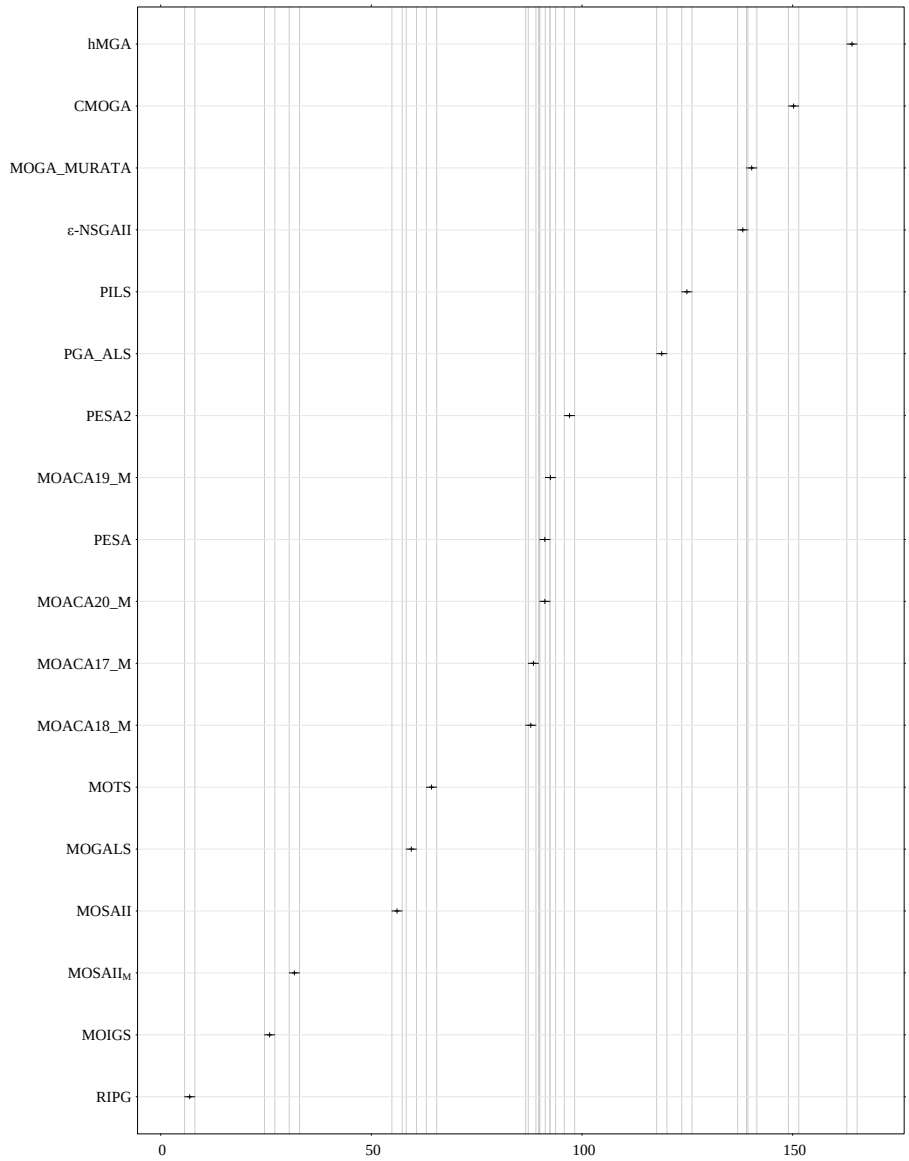


Figura C.16: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD125, variable respuesta indicador de hipervolumen y $t = 200$ para los objetivos de $C_{\max}$ y tardanza total ponderada con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

Finalmente, para terminar este anexo, presentamos los resultados de tests de rangos para el experimento para el taller de flujo de permutación multi-objetivo con tiempos de cambio para el criterio de parada $t = 200$.
En esta caso en particular, los tests de rango devuelven un orden consistente en la clasificación de los algoritmos, que no cambia para los dos indicadores utilizados, ni por grupo de instancias. En prime lugar se ubica el algoritmo RIPG, en segundo lugar el MOIGS, luego el algoritmo MOSAII$_M$, seguido por el MOSAII y en la quinta posición se ubica el algoritmo MOGALS.
El hecho que el algoritmo MOGALS tienda a ubicarse más abajo en la clasificación cuando t es mayor puede indicar que el algoritmo queda estancado a pesar que aumenta el tiempo de ejecución, mientras que los demás algoritmos son capaces de seguir avanzando en la búsqueda. Esto se explicaría debido a que el algoritmo MOGALS posee una búsqueda local muy exhaustiva que ocupa casi todo el tiempo disponible.

En conclusión, se puede ver como de manera consistente y para todos los tests realizados el algoritmo RIPG ocupa la primera posición, para todos los grupos de instancias, indicadores, combinaciones de objetivos y criterios de parada.

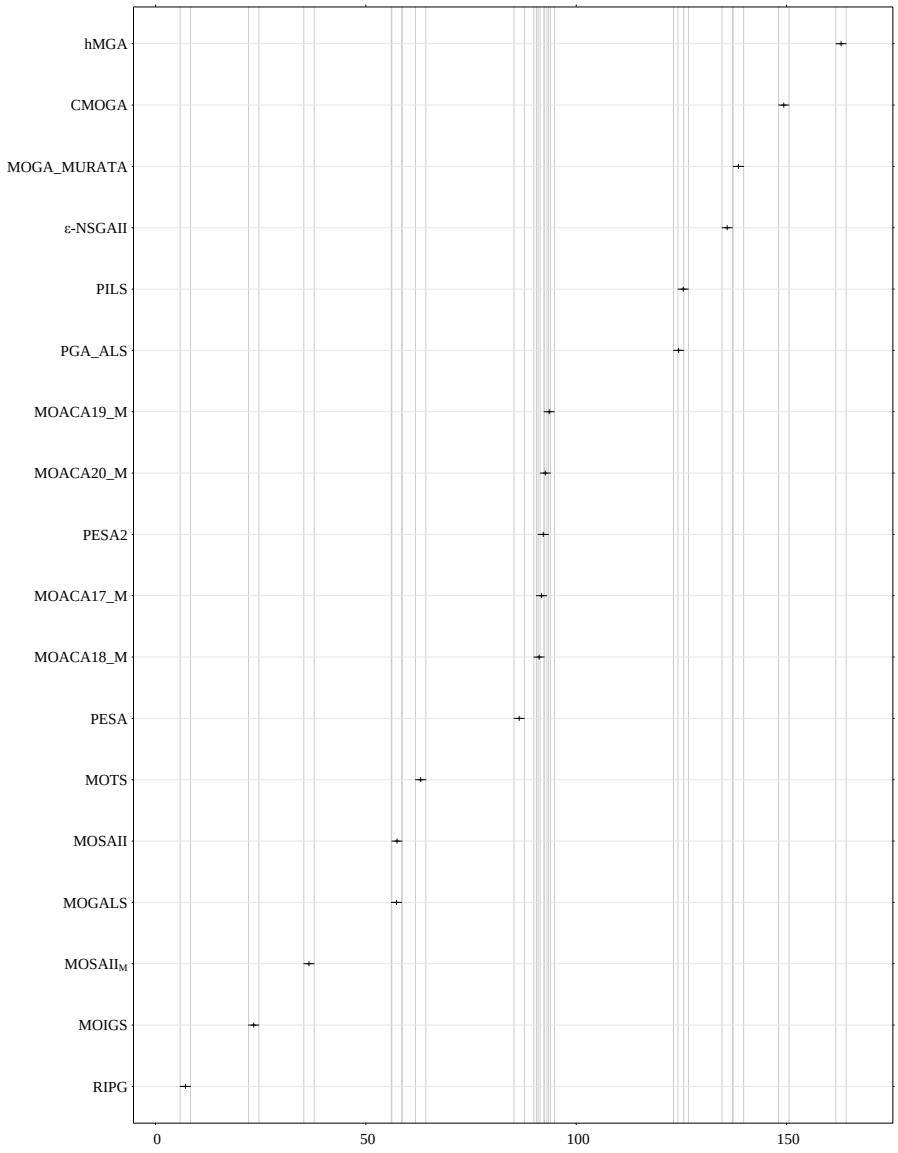


Figura C.5: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD125, variable respuesta indicador épsilon y $t = 150$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

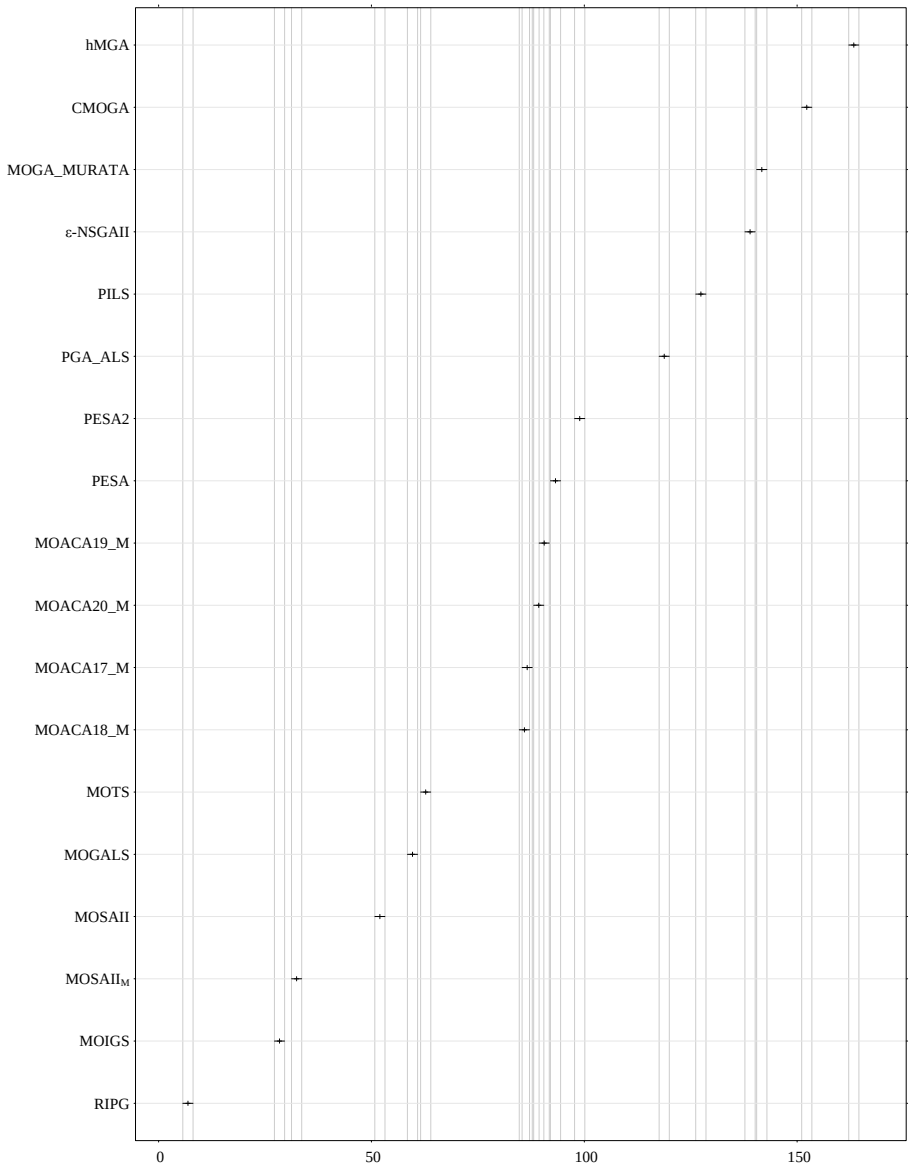


Figura C.6: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD125, variable respuesta indicador de hipervolumen y $t = 150$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

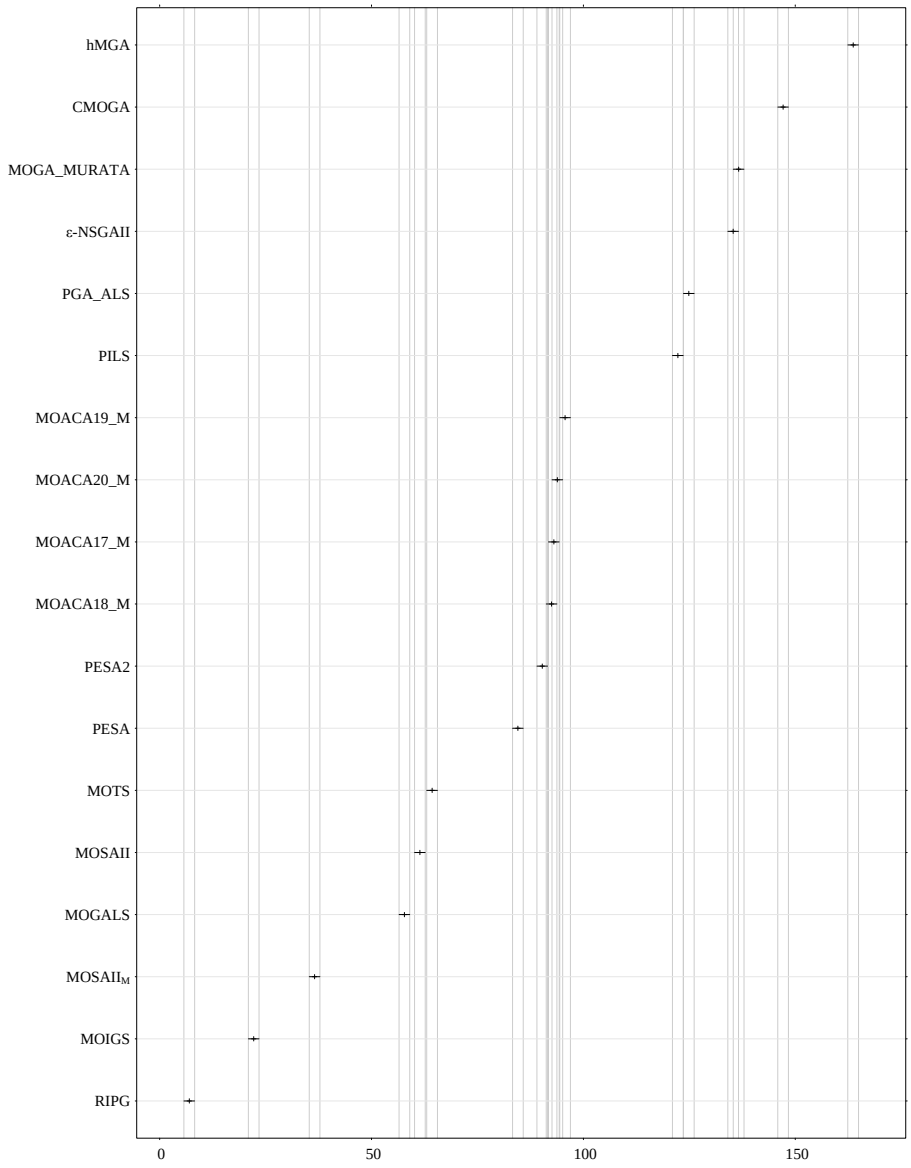


Figura C.7: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD125, variable respuesta indicador épsilon y $t = 200$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

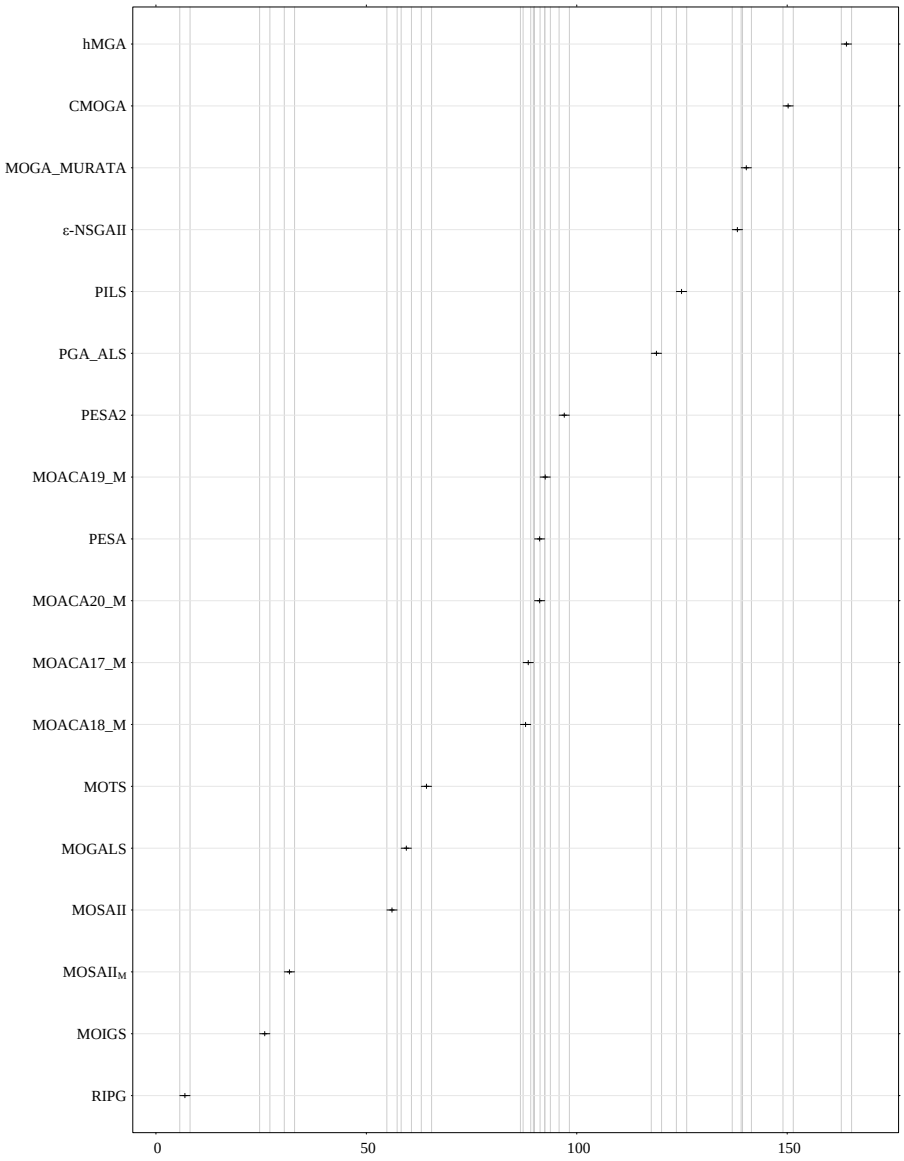


Figura C.8: Resultados de tests de Friedman (rangos) para el grupo de instancias SSD125, variable respuesta indicador de hipervolumen y $t = 200$ para los objetivos de $C_{\max}$ y tiempo total de flujo con intervalos de confianza de Tukey de 99 % (niveles de confianza ajustados al 95 %).

REFERENCIAS

Allahverdi, A. (2003). The two- and m-machine flowshop scheduling problems with bicriteria of makespan and mean flowtime. *European Journal of Operational Research*, 147(2):373–396.

Allahverdi, A. (2004). A new heuristic for m-machine flowshop scheduling problem with bicriteria of makespan and maximum tardiness. *Computers & Operations Research*, 31(2):157–180.

Allahverdi, A., Ng, C. T., Cheng, T. C. E., y Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032.

Armentano, V. A. y Arroyo, J. E. C. (2004). An application of a multi-objective tabu search algorithm to a bicriteria flowshop problem. *Journal of Heuristics*, 10(5):463–481.

Arroyo, J. E. C. y Armentano, V. A. (2004). A partial enumeration heuristic for multi-objective flowshop scheduling problems. *Journal of the Operational Research Society*, 55(9):1000–1007.

Arroyo, J. E. C. y Armentano, V. A. (2005). Genetic local search for multi-objective flowshop scheduling problems. *European Journal of Operational Research*, 167(3):717–738.

Arroyo, J. E. C. y Souza Pereira, A. A. (2011). A grasp heuristic for the multi-objective permutation flowshop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 55(5-8):741–753.

Bagchi, T. P. (2001). Pareto-optimal solutions for multi-objective production scheduling problems. En Zitzler, E., Deb, K., Thiele, L., Coello Coello, C. A., y Corne, D., editores, *Evolutionary Multi-Criterion Optimization, First International Conference, EMO 2001, Zurich, Switzerland, March 7-9, 2001, Proceedings*, volumen 1993 de *Lecture Notes in Computer Science*, páginas 458–471. Springer.

Basseur, M. (2005). *Conception d'algorithmes coopératifs pour l'optimisation multi-objectif: Application aux problèmes d'ordonnancement de type Flow-shop.* Tesis doctoral, Laboratoire d'Informatique Fondamentale de Lille. Lille, France. En francés.

Behnamian, J., Ghomi, S. F., y Zandieh, M. (2009). A multi-phase covering pareto-optimal front method to multi-objective scheduling in a realistic hybrid flowshop using a hybrid metaheuristic. *Expert Systems with Applications*, 36(8):11057 – 11069.

Bradstreet, L., While, L., y Barone, L. (2008). A fast incremental hypervolume algorithm. *Evolutionary Computation, IEEE Transactions on*, 12(6):714–723.

Cavalieri, S. y Gaiardelli, P. (1998). Hybrid genetic algorithms for a multiple-objective scheduling problem. *Journal of Intelligent Manufacturing*, 9(4):361–367.

Chakravarthy, K. y Rajendran, C. (1999). A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization. *Production Planning and Control*, 10(7):707–714.

Chang, P.-C., Chen, S.-H., y Hsieh, J.-C. (2006). A global archive sub-population genetic algorithm with adaptive strategy in multi-objective parallel-machine scheduling problem. En Jiao, L., Wang, L., Gao, X.-b., Liu, J., y Wu, F., editores, *Advances in Natural Computation*, volumen 4221 de *Lecture Notes in Computer Science*, páginas 730–739. Springer Berlin Heidelberg.

Chang, P. C., Hsieh, J. C., y Lin, S. G. (2002). The development of gradual-priority weighting approach for the multi-objective flowshop scheduling problem. *International Journal of Production Economics*, 79(3):171–183.

Cheng, H.-C., Chiang, T.-C., y Fu, L.-C. (2008). Multiobjective permutation flowshop scheduling by an adaptive genetic local search algorithm. En *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, páginas 1596 –1602.

Chiang, T.-C., Cheng, H.-C., y Fu, L.-C. (2009). Multiobjective permutation flow shop scheduling using a memetic algorithm with an neh-based local search. En Huang, D.-S., Jo, K.-H., Lee, H.-H., Kang, H.-J., y Bevilacqua, V., editores, *Emerging Intelligent Computing Technology and Applications*, volumen 5754 de *Lecture Notes in Computer Science*, páginas 813–825. Springer Berlin / Heidelberg.

Chiang, T.-C. y Fu, L.-C. (2010). An improved multiobjective memetic algorithm for permutation flow shop scheduling. En *Evolutionary Computation (CEC), 2010 IEEE Congress on*, páginas 1 –8.

Chou, F. D. y Lee, C. E. (1999). Two-machine flowshop scheduling with bicriteria problem. *Computers & Industrial Engineering*, 36(3):549–564.

Ciavotta, M., Minella, G., y Ruiz, R. (2009). Un algoritmo eficaz para el problema del taller de flujo multiobjetivo con tiempos de cambio dependientes de la secuencia. En *XXXI Congreso Nacional de Estadística e Investigación Operativa*.

Ciavotta, M., Minella, G., y Ruiz, R. (2013). Multi-objective sequence dependent setup times permutation flowshop: A new algorithm and a comprehensive study. *European Journal of Operational Research*. En prensa.

Ciavotta, M., Ruiz, R., y Minella, G. (2009). Multi-objective sequence dependent setup times flowshop scheduling: A new algorithm and a comprehensive study. En *Learning and intelligent optimization Third International Conference (LION3)*.

Conover, W. (1999). *Practical Nonparametric Statistics*. John Wiley & Sons, New York, third edición.

Corne, D. W., Jerram, N. R., Knowles, J. D., y Oates, M. J. (2001). PESA-II: Region-based selection in evolutionary multiobjective optimization. En Spector, L., Goodman, E. D., Wu, A., Langdon, W. B., Voigt, H. M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M. H., y Burke, E., editores, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, páginas 283–290, San Francisco, California, USA. Morgan Kaufmann.

Corne, D. W., Knowles, J. D., y Oates, M. J. (2000). The pareto envelope-based selection algorithm for multiobjective optimization. En Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo Guervós, J. J., y Schwefel, H. P., editores, *Parallel Problem Solving from Nature - PPSN VI, 6th International Conference, Paris, France, September 18-20, 2000, Proceedings*, volumen 1917 de *Lecture Notes in Computer Science*, páginas 839–848. Springer.

Daniels, R. L. y Chambers, R. J. (1990). Multiobjective flow-shop scheduling. *Naval research logistics*, 37(6):981–995.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, West Sussex, England.

Deb, K. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Deb, K. (2003). Towards a quick computation of well-spread pareto-optimal solutions. *EVOLUTIONARY MULTI-CRITERION OPTIMIZATION, PROCEEDINGS*, 2632:222–236.

Du, J. y Leung, J. Y.-T. (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15(3):483–495.

Dubois-Lacoste, J., López-Ibáñez, M., y Stützle, T. (2009). Effective hybrid stochastic local search algorithms for biobjective permutation flowshop scheduling. En Blesa, M., Blum, C., Di Gaspero, L., Roli, A., Sampels, M., y Schaerf, A., editores, *Hybrid Metaheuristics*, volumen 5818 de *Lecture Notes in Computer Science*, páginas 100–114. Springer Berlin // Heidelberg.

Dubois-Lacoste, J., López-Ibáñez, M., y Stützle, T. (2011). A hybrid tp+pls algorithm for bi-objective flow-shop scheduling problems. *Computers & Operations Research*, 38(8):1219–1236.

Fanjul-Peyro, L. y Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55 – 69.

Framiñan, J. M., Gupta, J. N. D., y Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(12):1243–1255.

Framiñan, J. M. y Leisten, R. (2006). A heuristic for scheduling a permutation flowshop with makespan objective subject to maximum tardiness. *International Journal of Production Economics*, 99(1-2):28–40.

Framiñan, J. M. y Leisten, R. (2008). A multi-objective iterated greedy search for flowshop scheduling with makespan and flowtime criteria. *OR Spectrum*, 30:787–804.

Framiñan, J. M., Leisten, R., y Ruiz-Usano, R. (2002). Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. *European Journal of Operational Research*, 141(3):559–569.

Gangadharan, R. y Rajendran, C. (1994). A simulated annealing heuristic for scheduling in a flowshop with bicriteria. *Computers & Industrial Engineering*, 27(1-4):473–476.

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

Geiger, M. J. (2007). On operators and search space topology in multi-objective flow shop scheduling. *European Journal of Operational Research*, 181(1):195 – 206.

Gonzalez, T. y Sahni, S. (1978). Flowshop and jobshop schedules: Complexity and approximation. *Operations Research*, 26(1):36–52.

Graham, R. L., Lawler, E. L., Lenstra, J. K., y Kan, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326.

Grunert da Fonseca, V., Fonseca, C. M., y Hall, A. O. (2001). Inferential performance assessment of stochastic optimisers and the attainment function. *In Evolutionary Multi-Criterion Optimization, First International Conference, vol. 1993 of Lecture Notes in Computer Science. Spring-Verlang*, 1993:213–225.

Gupta, J. N. D., Hennig, K., y Werner, F. (2002). Local search heuristics for two-stage flow shop problems with secondary criterion. *Computers & Operations Research*, 29(2):123–149.

Gupta, J. N. D., Neppalli, V. R., y Werner, F. (2001). Minimizing total flow time in a two-machine flowshop problem with minimum makespan. *International Journal of Production Economics*, 69(3):323–338.

Gupta, J. N. D., Palanimuthu, N., y Chen, C. L. (1999). Designing a tabu search algorithm for the two-stage flow shop problem with secondary criterion. *Production Planning & Control*, 10(3):251–265.

Hasija, S. y Rajendran, C. (2004). Scheduling in flowshops to minimize total tardiness of jobs. *International Journal of Production Research*, 42(11):2289–2301.

Ho, J. C. y Chang, Y.-L. (1991). A new heuristic for the n-job, M-machine flow-shop problem. *European Journal of Operational Research*, 52:194–202.

Hoogeveen, H. (2005). Multicriteria scheduling. *European Journal of Operational Research*, 167(3):592–623.

Ishibuchi, H. y Murata, T. (1998). A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems Man and Cybernetics*, 28(3):392–403.

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

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

Jones, D. F., Mirrazavi, S. K., y Tamiz, M. (2002). Multi-objective meta-heuristics: An overview of the current state-of-the-art. *European Journal of Operational Research*, 137(1):1–9.

Knowles, J. y Corne, D. W. (2000). Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172.

Knowles, J., Thiele, L., y Zitzler, E. (2006). A tutorial on the performance assessment of stochastic multiobjective optimizers. Technical Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland. revised version.

Kollat, J. B. y Reed, P. M. (2005). The value of online adaptive search: A performance comparison of nsgaii, $\varepsilon-$nsgaii and $\varepsilon-$moea. En Coello Coello, C. A., Hernández Aguirre, A., y Zitzler, E., editores, *Evolutionary Multi-Criterion Optimization, Third International Conference, EMO 2005, Guanajuato, Mexico, March 9-11, 2005, Proceedings*, volumen 3410 de *Lecture Notes in Computer Science*, páginas 386–398. Springer.

Kurz, M. E. y Askin, R. G. (2004). Scheduling flexible flow lines with sequence-dependent setup times. *European Journal of Operational Research*, 159(1):66 – 82.

Lee, C. E. y Chou, F. D. (1998). A two-machine flowshop scheduling heuristic with bicriteria objective. *International Journal of Industrial Engineering*, 5(2):128–139.

Lee, W. C. y Wu, C. C. (2001). Minimizing the total flow time and the tardiness in a two-machine flow shop. *International Journal of Systems Science*, 32(3):365–373.

Lemesre, J., Dhaenens, C., y Talbi, E. G. (2007). An exact parallel method for a bi-objective permutation flowshop problem. *European Journal of Operational Research*, 177(3):1641–1655.

Liao, C.-J., Yu, W.-C., y Joe, C.-B. (1997). Bicriterion scheduling in the two-

machine flowshop. *Journal of the Operational Research Society*, 48(9):929–935.

Lin, B. M. T. y Wu, J. M. (2006). Bicriteria scheduling in a two-machine permutation flowshop. *International Journal of Production Research*, 44(12):2299–2312.

López-Ibáñez, M., Paquete, L., y Stützle, T. (2006). Hybrid population-based algorithms for the bi-objective quadratic assignment problem. *Journal of Mathematical Modelling and Algorithms*, 5(1):111–137.

Loukil, T., Teghem, J., y Fortemps, P. (2000). Solving multi-objective production scheduling problems with tabu search. *Control and Cybernetics*, 29(3):819–828.

Loukil, T., Teghem, J., y Tuyttens, D. (2005). Solving multi-objective production scheduling problems using metaheuristics. *European Journal of Operational Research*, 161(1):42–61.

McGeoch, C. (1992). Analyzing algorithms by simulation: variance reduction techniques and simulation speedups. *ACM Comput. Surv.*, 24(2):195–212.

Melab, N., Mezmaz, M., y Talbi, E. G. (2006). Parallel cooperative metaheuristics on the computational grid. a case study: the bi-objective flow-shop problem. *Parallel Computing*, 32(9):643–659.

Minella, G., Ruiz, R., y Ciavotta, M. (2007a). A review and evaluation of multi-objective algorithms for the flowshop scheduling problem. En *EURO XXII 2007 22nd European Conference on Operational Research*.

Minella, G., Ruiz, R., y Ciavotta, M. (2007b). A review and evaluation of multi-objective algorithms for the flowshop scheduling problem. En *ORP3 Conference*.

Minella, G., Ruiz, R., y Ciavotta, M. (2008a). A review and evaluation of multi-objective algorithms for the flowshop scheduling problem. *Informs Journal on Computing*, 20(3):451–471.

Minella, G., Ruiz, R., y Ciavotta, M. (2008b). New iterated pareto greedy algorithms for the multi-objective flowshop problem. En *Eleventh international workshop on project management and scheduling - PMS 2008*.

Minella, G., Ruiz, R., y Ciavotta, M. (2009). Un algoritmo voraz iterativo para el taller de flujo de permutación con múltiples objetivos. En *XXXI Congreso Nacional de Estadística e Investigación Operativa*.

Minella, G., Ruiz, R., y Ciavotta, M. (2011). Restarted iterated pareto greedy algorithm for multi-objective flowshop scheduling problems. *Computers and Operations Research*, 38(11):1521–1533.

Mladenovic, N. y Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100.

Montgomery, D. C. (2009). *Design and Analysis of Experiments*. Wiley, New York, seventh edición.

Murata, T., Ishibuchi, H., y Gen, M. (2001). Specification of genetic search directions in cellular multi-objective genetic algorithms. En Zitzler, E., Deb, K., Thiele, L., Coello Coello, C. A., y Corne, D., editores, *Evolutionary Multi-Criterion Optimization, First International Conference, EMO 2001, Zurich, Switzerland, March 7-9, 2001, Proceedings*, volumen 1993 de *Lecture Notes in Computer Science*, páginas 82–95. Springer.

Murata, T., Ishibuchi, H., y Tanaka, H. (1996). Multi-objective genetic algorithm and its applications to flowshop scheduling. *Computers & Industrial Engineering*, 30(4):957–968.

Mutlu Yenisey, M. y Yagmahan, B. (2013). Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega*, (0):–.

Nagar, A., Heragu, S. S., y Haddock, J. (1995a). A branch-and-bound approach for a two-machine flowshop scheduling problem. *Journal of the Operational Research Society*, 46(6):721–734.

Nagar, A., Heragu, S. S., y Haddock, J. (1995b). Mutiple and bicriteria scheduling: A lierature survey. *European Journal of Operational Research*, 81(1):88–104.

Nagar, A., Heragu, S. S., y Haddock, J. (1996). A combined branch-and-bound and genetic algorithm based approach for a flowshop scheduling problem. *Annals of Operations Research*, 63(3):397–414.

Nawaz, M., Enscore, E., y Ham, I. (1983). A Heuristic Algorithm for the m-Machine, n-Job Flow-shop Sequencing Problem. *OMEGA, The International Journal of Management Science*, 11(1):91–95.

Neppalli, V. R., Chen, C.-L., y Gupta, J. N. D. (1996). Genetic algorithms for the two-stage bicriteria flowshop problem. *European Journal of Operational Research*, 95(2):356–373.

Pan, Q.-K. y Ruiz, R. (2013). A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research*, 40(1):117 – 128.

Pan, Q.-K., Wang, L., y Zhao, B.-H. (2008). An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. *The International Journal of Advanced Manufacturing Technology*, 38(7-8):778–786.

Paquete, F. L. (2005). *Stochastic Local Search Algorithms for Multiobjective Combinatorial Optimization: Method and Analysis*. Tesis doctoral, Com-

puter Science Department. Darmstadt University of Technology. Darmstadt, Germany.

Parthasarathy, S. y Rajendran, C. (1997a). A simulated annealing heuristic for scheduling to minimize mean weighted tardiness in a flowshop with sequence-dependent setup times of jobs - a case study. *Production Planning & Control*, 8(5):475–483.

Parthasarathy, S. y Rajendran, C. (1997b). An experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence-dependent setup times of jobs. *International Journal of Production Economics*, 49(3):255–263.

Pasupathy, T., Rajendran, C., y Suresh, R. K. (2006). A multi-objective genetic algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs. *The International Journal of Advanced Manufacturing Technology*, 27(7-8):804–815.

Ponnambalam, S. G., Jagannathan, H., Kataria, M., y Gadicherla, A. (2004). A TSP-GA multi-objective algorithm for flow-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 23(11-12):909–915.

Potts, C. y Wassenhove, L. V. (1982). A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters*, 1(5):177 – 181.

Rahimi-Vahed, A. R. y Mirghorbani, S. M. (2007). A multi-objective particle swarm for a flow shop scheduling problem. *Journal of Combinatorial Optimization*, 13(1):79–102.

Rajendran, C. (1992). Two-stage flowshop scheduling problem with bicriteria. *Journal of the Operational Research Society*, 43(9):871–884.

Rajendran, C. (1994). A heuristic for scheduling in flowshop and flowline-based manufacturing cell with multicriteria. *International Journal of Production Research*, 32(11):2541–2558.

Rajendran, C. (1995). Heuristics for scheduling in flowshop with multiple objectives. *European Journal of Operational Research*, 82(3):540–555.

Rajendran, C. y Ziegler, H. (1997). A heuristic for scheduling to minimize the sum of weighted flowtime of jobs in a flowshop with sequence-dependent setup times of jobs. *Computers & Industrial Engineering*, 33(1-2):281–284.

Rajendran, C. y Ziegler, H. (2003). Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. *European Journal of Operational Research*, 149(3):513–522.

Rajendran, C. y Ziegler, H. (2009). A multi-objective ant-colony algorithm for permutation flowshop scheduling to minimize the makespan and total flowtime of jobs. En Chakraborty, U., editor, *Computational Intelligence in Flow Shop and Job Shop Scheduling*, volumen 230 de *Studies in Computational Intelligence*, páginas 53–99. Springer Berlin / Heidelberg, segunda edición.

Rashidi, E., Jahandar, M., y Zandieh, M. (2010). An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines. *The International Journal of Advanced Manufacturing Technology*, 49(9-12):1129–1139.

Ravindran, D., Haq, A. N., Selvakuar, S. J., y Sivaraman, R. (2005). Flow shop scheduling with multiple objective of minimizing makespan and total flow time. *The International Journal of Advanced Manufacturing Technology*, 25(9-10):1007–1012.

Ribas, I., Companys, R., y Tort-Martorell, X. (2011). An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega*, 39(3):293 – 301.

Ribas, I., Leisten, R., y Framiñan, J. M. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, 37(8):1439 – 1454.

Ríos-Mercado, R. Z. y Bard, J. F. (1998). Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. *Computers & Operations Research*, 25(5):351–366.

Ríos-Mercado, R. Z. y Bard, J. F. (1999). An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup times. *Journal of Heuristics*, 5(1):53–70.

Ríos-Mercado, R. Z. y Bard, J. F. (2003). The flow shop scheduling polyhedron with setup times. *Journal of Combinatorial Optimization*, 7(3):291–318.

Ruiz, R. y Allahverdi, A. (2009). New heuristics for no-wait flow shops with a linear combination of makespan and maximum lateness. *International Journal of Production Research*, 47(20):5717–5738.

Ruiz, R., Şerifoğlu, F. S., y Urlings, T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, 35(4):1151 – 1175.

Ruiz, R. y Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494.

Ruiz, R., Maroto, C., y Alcaraz, J. (2005). Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *European Journal of Operational Research*, 165(1):34–54.

Ruiz, R. y Minella, G. (2007). Nuevos algoritmos avanzados para secuenciacion multiobjetivo. En *XXX Congreso Nacional de Estadística e Investigación Operativa*.

Ruiz, R. y Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033 – 2049.

Ruiz, R. y Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143 – 1159.

Ruiz, R. y Vázquez-Rodríguez, J. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1):1–18.

Sayın, S. y Karabatı, S. (1999). A bicriteria approach to the two-machine flow shop scheduling problem. *European Journal of Operational Research*, 113(2):435–449.

Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. En *Proceedings of the 1st International Conference on Genetic Algorithms*, páginas 93–100, Mahwah, NJ, USA. Lawrence Erlbaum Associates, Inc.

Selen, W. J. y Hott, D. D. (1986). A mixed-integer goal-programming formulation of the standard flowshop scheduling problem. *Journal of the Operational Research Society*, 37(12):1121–1128.

Sha, D. y Hung Lin, H. (2009). A particle swarm optimization for multiobjective flowshop scheduling. *The International Journal of Advanced Manufacturing Technology*, 45:749–758.

Simons, Jr, J. V. (1992). Heuristics in flow shop scheduling with sequence dependent setup times. *Omega-International Journal of Management Science*, 20(2):215–225.

Sivrikaya-Şerifoğlu, F. y Ulusoy, G. (1998). A bicriteria two-machine permutation flowshop problem. *European Journal of Operational Research*, 107(2):414–430.

Sridhar, J. y Rajendran, C. (1996). Scheduling in flowshop and cellular manufacturing systems with multiple objectives: A genetic algorithmic approach. *Production Planning & Control*, 7(4):374–382.

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

Suresh, R. K. y Mohanasundaram, K. M. (2004). Pareto archived simulated annealing for permutation flow shop scheduling with multiple objectives. En *IEEE Conference on Cybernetics and Intelligent Systems (CIS), Singapore, December 1-3, 2004, Proceedings*, volumen 2, páginas 712–717.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

Tasgetiren, M. F., Pan, Q.-K., Suganthan, P., y Chen, A. H.-L. (2011). A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. *Information Sciences*, 181(16):3459 – 3475.

T'Kindt, V. y Billaut, J.-C. (2001). Multicriteria schduling problems: A survey. *RAIRO Recherche Opérationnelle - Operations Research*, 35(2):143–163.

T'Kindt, V. y Billaut, J.-C. (2002). *Multicriteria scheduling: Theory, models and algorithms*. Springer, Berlin.

T'Kindt, V., Gupta, J. N. D., y Billaut, J.-C. (2003). Two-machine flowshop scheduling with a secondary criterion. *Computers & Operations Research*, 30(4):505–526.

T'Kindt, V., Monmarché, N., Tercinet, F., y Laügt, D. (2002). An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. *European Journal of Operational Research*, 142(2):250–257.

Toktaş, B., Azizoğlu, M., y Köksalan, S. K. (2004). Two-machine flow shop scheduling with two criteria: Maximum earliness and makespan. *European Journal of Operational Research*, 157(2):286–295.

Toyama, F., Shoji, K., y Miyamichi, J. (2008). An iterated greedy algorithm for the node placement problem in bidirectional manhattan street networks. En *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, páginas 579–584, New York, NY, USA. ACM.

Tseng, L.-Y. y Lin, Y.-T. (2009). A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 198(1):84 – 92.

Urlings, T. (2011). *Heuristics and Metaheuristics for heavily constrained hybrid Flowshop Problems*. Tesis doctoral, Universitat Politècnica de València.

Urlings, T., Minella, G., y Ruiz, R. (2010a). Bi-objective pareto optimization for the hybrid flowshop problem. En *Twelfth International Workshop on Project management and Scheduling*.

Urlings, T., Minella, G., y Ruiz, R. (2010b). Optimización muilti-objetivo para problemas de flowshop híbrido. En *XXXII Congreso Nacional de Estadística e Investigación Operativa y de las VI Jornadas de Estadística Pública*.

Vallada, E., Ruiz, R., y Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research*, 35(4):1350 – 1373.

Varadharajan, T. 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.

Wenzhong, G., Guolong, C., Min, H., y Shuili, C. (2007). A discrete particle swarm optimization algorithm for the multiobjective permutation flowshop sequencing problem. En Cao, B.-Y., editor, *Fuzzy Information and Engineering*, volumen 40 de *Advances in Soft Computing*, páginas 323–331. Springer Berlin / Heidelberg.

While, L., Bradstreet, L., Barone, L., y Hingston, P. (2005). Heuristics for optimizing the calculation of hypervolume for multi-objective optimization problems. En *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volumen 3, páginas 2225–2232 Vol. 3.

While, L., Hingston, P., Barone, L., y Huband, S. (2006). A faster algorithm for calculating hypervolume. *Evolutionary Computation, IEEE Transactions on*, 10(1):29–38.

Wilson (1989). Alternative formulations of a flowshop scheduling problem. *Journal of the Operational Research Society*, 40(4):395–399.

Yandra y Tamura, H. (2007). A new multiobjective genetic algorithm with heterogeneous population for solving flowshop scheduling problems. *International journal of computer integrated manufacturing*, 20(5):465–477.

Yeh, W.-C. (1999). A new branch-and-bound approach for the $n/2/flowshop/\alpha F + \beta C_{max}$. *Computers & Operations Research*, 26(13):1293–1310.

Yeh, W.-C. (2001). An efficient branch-and-bound algorithm for the two-machine bicriteria flowshop scheduling problem. *Journal of Manufacturing Systems*, 20(2):113–123.

Yeh, W.-C. (2002). A memetic algorithm for the $n/2/flowshop/\alpha F+\beta C_{max}$ scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 20(6):464–473.

Ying, K.-C. (2008). An iterated greedy heuristic for multistage hybrid flowshop scheduling problems with multiprocessor tasks. *IEEE, Transactions on Evolutionary Computation*, 60(6):810–817.

Zhi, Y., Armin, F., Henning, H., Prasanna, B., Thomas, S., y Michael, S. (2008). Iterated greedy algorithms for a real-world cyclic train scheduling problem. En *Hybrid Metaheuristics*, páginas 102–116. Springer Berlin.

Zitzler, E., Brockhoff, D., y Thiele, L. (2007). The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. En Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., y Murata, T., editores, *Evolutionary Multi-Criterion Optimization*, volumen 4403 de *Lecture Notes in Computer Science*, páginas 862–876. Springer Berlin / Heidelberg.

Zitzler, E., Knowles, J., y Thiele, L. (2008). Quality assessment of pareto set approximations. En Branke, J., Deb, K., Miettinen, K., y Slowinski, R., editores, *Multiobjective Optimization*, volumen 5252 de *Lecture Notes in Computer Science*, páginas 373–404. Springer Berlin / Heidelberg.

Zitzler, E. y Künzli, S. (2004). Indicator-based selection in multiobjective search. En *8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, páginas 832–842, Birmingham, UK. Springer-Verlag, Berlin, Germany.

Zitzler, E., Laumanns, M., y Thiele, L. (2001). SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland.

Zitzler, E. y Thiele, L. (1999). Multiobjective evolutionary algorithms: A

comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., y da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: an analysis and review. *IEEE, Transactions on Evolutionary Computation*, 7(2):117–132.