UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Tesis Doctoral

Secuenciación de máquinas con necesidad de ajustes y recursos adicionales

---

Doctorado en Estadística y Optimización

Autor

Juan Camilo Yepes Borrero

Directores

Federico Perea Rojas-Marcos

María Fulgencia Villa Juliá

Departamento de Estadística e Investigación Operativa Aplicadas Y Calidad

Valencia, Octubre 2020

Resumen

En esta tesis doctoral se estudia el problema de secuenciación de máquinas paralelas no relacionadas con necesidad de ajustes y recursos adicionales asignados en los ajustes. En este problema, se tiene un grupo de tareas (también llamadas trabajos), donde cada una debe ser procesada en una de las máquinas paralelas disponibles. Para procesar una tarea después de otra en la misma máquina, se debe hacer un ajuste en la máquina. Se asume que estos ajustes deben ser realizados por un recurso adicional limitado (por ejemplo, operarios). En esta tesis doctoral se estudian dos variantes del problema planteado: 1) considerando el problema con el único objetivo de minimizar el tiempo máximo de finalización de todos los trabajos (*makespan*), y 2) considerando el problema multi-objetivo minimizando simultáneamente el *makespan* y el consumo máximo de recursos adicionales.

Inicialmente, se realiza una completa revisión bibliográfica sobre estudios relacionados con el problema planteado. En esta revisión se detecta que, a pesar de existir numerosos estudios de secuenciación de máquinas paralelas, no muchos de estos estudios tienen en cuenta recursos adicionales. Posteriormente, para introducir el problema a estudiar antes de plantear métodos de resolución, se realiza una breve explicación de los principales problemas de secuenciación de máquinas paralelas.

El problema de un solo objetivo está clasificado como $\mathcal{NP}$-Hard. Por ello, para abordar su resolución se han diseñado e implementado heurísticas y metaheurísticas siguiendo dos enfoques diferentes. Para el primer enfoque, que ignora la información sobre el consumo de recursos adicionales en la fase constructiva, se adaptan dos de los mejores algoritmos existentes en la literatura para el problema de máquinas paralelas con ajustes sin necesidad de recursos adicionales. En el segundo enfoque, que sí tiene en cuenta la información sobre el consumo de recursos adicionales en la fase constructiva, se proponen nuevos algoritmos heurísticos y metaheurísticos para resolver el problema. Tras analizar los resultados de los experimentos computacionales realizados, concluimos que hay diferencias entre los dos enfoques, siendo significativamente mejor el enfoque que tiene en cuenta la información sobre los recursos adicionales.

Al igual que en el caso de un solo objetivo, la complejidad del problema multi-objetivo obliga a presentar algoritmos heurísticos o metaheurísticos para resolverlo. En esta tesis se presenta un nuevo algoritmo metaheurístico multi-objetivo eficiente para encontrar buenas aproximaciones a la frontera de Pareto del problema. Además, se adaptaron otros tres algoritmos que han mostrado buenos resultados en diferentes estudios de problemas de secuenciación de máquinas multi-objetivo. Después de realizar experimentos computacionales exhaustivos, concluimos que el nuevo algoritmo propuesto en esta tesis es significativamente mejor que los otros tres algoritmos existentes, y que se han adaptado para resolver este problema.

Abstract

In this thesis we study the unrelated parallel machine scheduling problem with setup times and additional limited resources in the setups. In this problem, we have a group of tasks (also called jobs), where each one must be processed on one of the available parallel machines. To process one job after another on the same machine, a setup must be made on the machine. It is assumed that these setups on machines must be made by a limited additional resource (eg, operators). In this thesis two variants of the problem are studied: 1) considering the problem with the objective of minimizing the maximum completion time of all jobs (makespan), and 2) considering the multi-objective problem, minimizing the makespan and the maximum consumption of additional resources.

Initially, a complete literature review is carried out on studies related to the problem addressed in this thesis. This review finds that despite numerous parallel machine scheduling studies, there are very few that take into account additional resources. Subsequently, to introduce the problem addressed before proposing resolution methods, a brief explanation of the main parallel machines scheduling problems is made.

The problem with a single objective is classified as $\mathcal{NP}$-Hard. Therefore, to solve it, heuristics and metaheuristics have been designed and implemented following two different approaches. For the first approach, which ignores the information on the consumption of resources in the construction phase, two of the best algorithms existing in the literature for the problem of parallel machines with setups without additional resources are adapted. For the second approach, which does take into account information on the consumption of resources in the construction phase, new heuristic and metaheuristic algorithms are proposed to solve the problem. Following the results of the computational experiments, we conclude that there are differences between the two approaches, the approach that takes into account the information on resources being significantly better.

As in the case of a single objective, the complexity of the multi-objective problem requires the formulation of heuristic or metaheuristic algorithms to solve it. In this thesis, a new efficient multi-objective metaheuristic algorithm is presented to find good approximations to the Pareto front of the problem. In addition, three other algorithms that have shown good results in different studies of multi-objective machine scheduling problems were adapted. After carrying out exhaustive computational experiments, we concluded that the new algorithm proposed in this thesis is significantly better than the other three adapted algorithms.

Resum

En aquesta tesi doctoral s'estudia el problema de seqüenciació de màquines paral·leles no relacionades amb necessitat d'ajustos i recursos addicionals assignats en els ajustos. En aquest problema, es tenen un grup de tasques (també anomenades treballs), on cadascuna ha de ser processada en una de les màquines paral·leles disponibles. Per processar una tasca després d'una altra en la mateixa màquina, s'ha de fer un ajustament en la màquina. S'assumeix que aquests ajustos en les màquines per a processar una tasca després del processament d'una altra, han de ser realitzats per un recurs addicional limitat (per exemple, operaris). En aquesta tesi doctoral s'estudien dos variants al problema plantejat: 1) considerant el problema com l'únic objectiu de minimitzar el temps màxim de finalització de tots els treballs (*makespan*), i 2) considerant el problema multi-objectiu minimitzant simultàniament el *makespan* i el consum màxim de recursos addicionals.

Inicialment, es realitza una completa revisió bibliogràfica sobre estudis relacionats amb el problema plantejat. En esta revisió es detecta que, tot i existir nombrosos estudis de seqüenciació de màquines paral·leles, hi ha molts pocs que tenen en compte recursos addicionals. Posteriorment, per introduir el problema a estudiar abans de plantejar mètodes de resolució, es realitza una breu explicació dels principals problemes de seqüenciació de màquines paral·leles.

El problema d'un sol objectiu està classificat com $\mathcal{NP}$-Hard. Per això, per abordar la seua resolució s'han dissenyat i implementat heurístiques y metaheurístiques seguint dos enfocs diferents. El primer enfoc ignora la informació sobre el consum de recursos en la fase constructiva, adaptant dos dels millors algoritmes existents en la literatura per al problema de seqüenciació de màquines paral·leles amb ajustaments sense necessitat de recursos. Per al segon enfoc si es té en compte la informació sobre el consum de recursos en la fase constructiva. Després d'analitzar els resultats dels experiments computacionals realitzats, concloem que hi ha diferencies entre els dos enfocs, sent significativament millor l'enfoc que té en compte la informació sobre el recursos.

De la mateixa manera que en el cas d'un sol objectiu, la complexitat del problema multi-objectiu obliga a presentar algoritmes heurístics o metaheurístics per a resoldre-ho. En aquesta tesi es presenta un nou algoritme metaheurístic multi-objectiu eficient per trobar bones aproximacions a la frontera de Pareto del problema. A més, es van adaptar altres tres algoritmes que han mostrat bons resultats en diferents estudis de problemes de seqüenciació de màquines multi-objectiu. Després de realitzar experiments computacionals exhaustius, concloem que el nou algoritme proposat en aquesta tesi és significativament millor que els altres tres algoritmes existents i que s'han adaptat per resoldre aquest problema.

Índice general

Agradecimientos

Esta tesis doctoral fue financiada por el Instituto Colombiano de Crédito Educativo y Estudios Técnicos en el Exterior ICETEX, bajo el programa Pasaporte a la Ciencia Doctorado, enmarcado en el Foco-reto país 4.2.3 (Foco 4-Sociedad, reto 2-lnnovación social para el desarrollo económico y la inclusión productiva, número 3-Productividad). Este apoyo es inmensamente agradecido.

Soy consciente que a pesar del esfuerzo personal que he hecho durante muchos años, no habría podido llegar hasta donde estoy sin el apoyo de muchas personas que he tenido la suerte de cruzarme. En estas líneas intentaré agradecerle a todos y todas.

Quiero agradecer principalmente a mis directores de tesis, Fede y Ful, por todo el apoyo que me han dado durante estos años. Por todo el tiempo que dedicaron con toda la paciencia del mundo para guiarme en este trabajo. Gracias por ser mucho más que mis directores de tesis, porque siempre han estado disponibles para lo que he necesitado. Ha sido un placer trabajar con ustedes durante estos años y espero poder seguir haciéndolo durante muchos más. Gracias porque han hecho que esta experiencia fuera una de las mejores de mi vida, y aunque Ful me diga que no les debo dar las gracias todo el tiempo, lo seguiré haciendo porque no tengo más que agradecimiento hacia ustedes.

También quiero agradecer a todos los miembros del SOA, especialmente a Eva por las charlas diarias camino a la universidad, y porque siempre que tuve alguna duda, estuvo dispuesta a ayudarme. Obviamente también a Rubén, que siempre sacaba tiempo de su muy apretada agenda para ayudarme en lo que necesitara. A Gerardo, por toda la paciencia que tuvo para explicarme la utilización del clúster para mis experimentos.

A Joan, Alba, Dani, Pedro y Giorgia, por hacer que el día a día en el despacho fuera agradable y hacer que prefiriera trabajar en el despacho y no en casa. Gracias a eso terminé a tiempo esta tesis.

Gracias a Pili y a mi abue, que a pesar de las dificultades, me adoptaron durante mis

años de estudio en Bogotá. Gracias porque lo único que puedo decir, es que recibí siempre el apoyo necesario para completar mis estudios.

Quiero agradecer especialmente a Mauricio, María Angélica y Geña, por recibirme en su casa cuando más lo necesitaba. Sin ustedes y su ayuda, nada de lo que he hecho después de eso hubiera sido posible.

A Rafael que siempre me animó a viajar y a tomar la decisión de venir a Europa a estudiar. Gracias porque sé que te alegras tanto como yo por esta tesis.

Estar lejos de mi familia y amigos durante tanto tiempo no ha sido fácil, por eso quiero agradecer a Maria, que ha sido mi compañera, familia y amiga estos años. Gracias por tanto.

Finalmente, quiero agradecer a mis padres, porque gracias a su trabajo y esfuerzo durante muchos años, he tenido una vida privilegiada, donde nunca me ha faltado nada para poder trabajar por lo que he querido.

Capítulo 1

Introducción

Durante las últimas décadas, la industria se ha venido enfrentando a entornos cada vez más competitivos. Debido a esto, la programación de la producción ha tomado gran relevancia en los últimos años. En el entorno actual, producir eficientemente es la prioridad número uno para que una compañía pueda sobrevivir y crecer en el mercado, por lo que es importante para las empresas contar con herramientas que les ayuden a la toma de decisiones encaminadas a ser más competitivas.

Dentro de la programación de la producción, encontramos los problemas de secuenciación de máquinas que, en pocas palabras, consisten en decidir en qué recurso (llamado máquina en la literatura) se procesa una tarea y cuándo debe ser procesada en ese recurso.

Existen muchas variaciones de problemas de secuenciación de máquinas estudiados en la literatura, buscando adaptarse a los diferentes contextos que se pueden encontrar en entornos industriales o de servicios. Dentro de estos problemas, uno de los más importantes es el de secuenciación de máquinas paralelas, donde cada una de las tareas a procesar debe ser procesada solo en una máquina. El gran interés que ha presentado este problema a lo largo de los años, puede deberse a que muchos entornos de manufactura se modelan con máquinas paralelas.

Debido a la constante evolución del mercado, sumado a la gran exigencia de la demanda, las empresas actualmente deben enfrentarse a constantes cambios en sus sistemas productivos. Hoy en día, la demanda está cambiando hacia un mercado más personalizado, donde

el cliente decide qué producto o servicio quiere y cuándo lo quiere, provocando que una empresa deba cambiar su programación de la producción diariamente.

Lo anterior obliga a las empresas actuales a contar con herramientas que les ayuden a planificar su producción ajustándose a los cambios de la demanda. Estas herramientas pueden ser modelos de planificación de la producción, donde el objetivo principal sea tener sistemas eficientes optimizando uno o más objetivos. Los objetivos a tener en cuenta en estos problemas pueden ser muy variados, como la minimización de tiempos de producción, la minimización de costes de almacenaje, o la maximización de beneficios.

El estudio realizado en esta tesis doctoral tiene la intención de ayudar con el desarrollo y la mejora de entornos y procesos productivos. Se busca proveer a la industria de herramientas de optimización de procesos y de buenas prácticas de manufactura. Todo esto está enmarcado en el Foco-reto país 3.2.3 del programa Pasaporte a la Ciencia del Instituto Colombiano de Crédito Educativo y Estudios Técnicos en el Exterior ICETEX (Foco 3-Sociedad, reto 2-innovación social para el desarrollo económico y la inclusión productiva, número 3-Productividad).

1.1. Motivación

A pesar del interés que ha despertado el estudio de la secuenciación de máquinas paralelas en los últimos años, la gran mayoría de los artículos que se encuentran en la literatura, no consideran algunas características que, a nuestro juicio, son importantes para modelar entornos productivos reales. Los primeros estudios sobre este tema se centran únicamente en la secuenciación de tareas en las diferentes máquinas, solo teniendo en cuenta si los tiempos de proceso de las tareas son iguales en las diferentes máquinas (problema de máquinas paralelas idénticas), o si el tiempo de proceso de una tarea puede variar dependiendo de la máquina (problemas de máquinas paralelas no relacionadas o uniformemente relacionadas). Con el paso de los años, se empezó a estudiar el problema asumiendo que las máquinas deben ser ajustadas entre el procesamiento de dos tareas diferentes.

Sin embargo, en la mayoría de entornos productivos, las máquinas no son el único recurso necesario, habiendo también operarios, moldes, plantillas, etc. Estos recursos adicionales suelen ser también limitados, generando una nueva restricción al problema estudiado. Además, estos recursos adicionales pueden ser necesarios para el procesamiento de una tarea o para realizar ajustes en las máquinas. Como se verá en el Capítulo 2, existen pocos trabajos que tienen en cuenta estos recursos adicionales limitados, estando la mayoría de ellos enfocados a asignar los recursos adicionales al procesamiento de las tareas.

Desde nuestro punto de vista, viendo que en muchos entornos productivos el procesamiento de tareas por parte de las máquinas suele ser automático, pero no así los ajustes de las máquinas que sí suelen necesitar de recursos adicionales, es necesario estudiar también el problema asignando recursos adicionales al ajuste de las máquinas. Por ejemplo, una máquina puede requerir limpieza después de procesar una tarea y antes de procesar la siguiente. Esta limpieza debe ser realizada por uno o más recursos adicionales y limitados (operarios, robots, etc.). En este contexto, introducimos el problema a estudiar en esta tesis doctoral, llamado problema de secuenciación de máquinas paralelas no relacionadas con necesidad de ajustes y recursos adicionales asignados a los ajustes.

Viendo la importancia que tienen los problemas de secuenciación de máquinas paralelas, y al no encontrar muchos trabajos relacionados que tengan en cuenta recursos adicionales, en esta tesis doctoral nos propusimos estudiar el problema, y diseñar e implementar métodos exactos y aproximados para abordar su resolución.

Finalmente, también se verá en el Capítulo 2 que la mayoría de estudios de secuenciación de máquinas paralelas solo consideran un objetivo a optimizar, mientras que en en la mayoría de problemas que se pueden encontrar en entornos reales, se requiere optimizar más de un objetivo a la vez. Por esto, también consideramos interesante analizar dos versiones del problema:

1. El problema con el único objetivo de minimizar el tiempo máximo de finalización de todos los trabajos, también llamado *makespan*.

2. El problema multi-objetivo buscando minimizar simultáneamente el *makespan* y el consumo máximo de recursos adicionales.

1.2. Objetivos

A continuación, expondremos tanto el objetivo general como los objetivos específicos que se pretenden alcanzar en esta tesis doctoral.

1.2.1. Objetivo general

- Introducir el problema de secuenciación de máquinas paralelas no relacionadas con tiempos de ajuste y necesidad de recursos adicionales en los ajustes, buscando acercarse un poco más a entornos productivos realistas.

1.2.2. Objetivos específicos

- Desarrollar métodos de resolución mono-objetivo eficientes para resolver el problema de secuenciación de máquinas no relacionadas con necesidad de ajustes y recursos adicionales en los ajustes, con el objetivo de minimizar el *makespan*.

- Desarrollar métodos de resolución multi-objetivo eficientes para resolver el problema de secuenciación de máquinas no relacionadas con necesidad de ajustes y recursos adicionales en los ajustes, buscando minimizar simultáneamente el *makespan* y el consumo de recursos adicionales.

- Realizar un análisis basado en experimentos computacionales para mostrar la efectividad de los métodos de resolución propuestos para los problemas mono-objetivo y multi-objetivo.

1.3. Estructura de la tesis doctoral

En el Capítulo 2 realizamos una completa revisión bibliográfica de problemas de secuenciación de máquinas, dividiendo los estudios en dos partes: 1) estudios de problemas de secuenciación de máquinas con un único objetivo, y 2) estudios de problemas de secuenciación de máquinas multi-objetivo.

En el Capítulo 3 introducimos brevemente los problemas de secuenciación de máquinas paralelas más relevantes, empezando por el problema más "simple", que es el de máquinas paralelas idénticas, hasta llegar al problema estudiado en esta tesis doctoral.

En el Capítulo 4 estudiamos el problema con el objetivo de minimizar el tiempo máximo de finalización de todos los trabajos (*makespan*). En este capítulo, definimos formalmente el problema, proponemos un modelo de programación lineal entera mixta, así como algoritmos heurísticos y metaheurísticos para resolverlo. Adicionalmente, presentamos los resultados computacionales tanto del modelo matemático, como de todos los algoritmos propuestos.

En el Capítulo 5 analizamos el problema multi-objetivo, buscando minimizar el *makespan* y el consumo de recursos adicionales simultáneamente. En este capítulo hacemos una breve explicación de problemas multi-objetivo y la manera de evaluar sus soluciones. Presentamos un algoritmo nuevo para resolver este problema, el cual comparamos con otros tres algoritmos adaptados de la literatura.

En el Capítulo 6 presentamos las conclusiones de esta tesis doctoral y las futuras líneas de investigación a desarrollar.

Por último, se muestran cuatro apéndices con información detallada de algunos estudios y partes de la tesis. En el Apéndice A se muestran los resultados completos de los algoritmos metaheurísticos propuestos en el Capítulo 4. El Apéndice B muestra los resultados completos de la calibración del nuevo algoritmo propuesto en el Capítulo 5. El Apéndice C muestra la calibración completa con sus resultados de los algoritmos adaptados en el Capítulo 5. Finalmente, el Apéndice D muestra los primeros avances de una variación del problema estudiado en esta tesis doctoral, donde considera que los tiempos de ajuste de las máquinas

son estocásticos.

Capítulo 2

Revisión bibliográfica

Los problemas de máquinas paralelas han sido ampliamente estudiados durante las últimas décadas. Sin embargo, variaciones que tienen en cuenta ajustes dependientes de la secuencia y/o recursos adicionales han sido objeto de muchos menos estudios. Igualmente, la gran mayoría de los estudios que se han hecho sobre máquinas paralelas, consideran solo un objetivo a optimizar. A pesar de que la optimización multi-objetivo ha recibido mayor interés últimamente, existe poca literatura de problemas de secuenciación de máquinas multi-objetivo. En esté capítulo dividiremos la revisión bibliográfica en dos secciones; una dedicada a problemas de secuenciación de máquinas paralelas con un solo objetivo, y otra centrada en problemas multi-objetivo.

2.1. Problemas de secuenciación de máquinas paralelas mono-objetivo

Tal y como ya se ha comentado previamente, se pueden encontrar muchos estudios sobre máquinas paralelas en la literatura. En Cheng y Sin (1990) encontramos una revisión bibliográfica sobre problemas de máquinas paralelas, en el que la gran mayoría de los estudios son de máquinas paralelas idénticas o uniformemente relacionadas. Años después, en Mokotoff (2001) se hace una nueva revisión en la que aún no se encuentran muchos estudios con máquinas paralelas no relacionadas. Posteriormente, en la Tesis de Fanjul-Peyró (2010) se hace una revisión muy completa de problemas de máquinas paralelas idénticas, uniformemente

relacionadas y no relacionadas. En esta tesis doctoral no se tratan en detalle problemas con necesidad de ajustes ni de recursos adicionales. Ya que para nosotros es fundamental tener en cuenta ajustes y recursos adicionales, a continuación, realizaremos una revisión sobre los trabajos más relevantes sobre máquinas paralelas con necesidad de ajustes y/o recursos adicionales.

La consideración de ajustes en problemas de secuenciación se empezó a estudiar hace varias décadas. Uno de los primeros trabajos al respecto fue el de Johnson (1954), en el que se plantea un problema de dos estaciones (cada estación con una máquina) donde cada trabajo debe ser procesado en ambas estaciones y tiene asociado un tiempo de proceso y un tiempo de ajuste para cada estación de trabajo.

Décadas después, en Barnes y Vanston (1981) se presenta un problema de una sola máquina con necesidad de ajustes dependientes de la secuencia y el objetivo de minimizar el *makespan*. En este trabajo los autores usan algoritmos exactos de ramificación y acotación (*branch and bound*) para resolver óptimamente instancias de hasta 20 trabajos. Para este mismo problema, en Laguna et al. (1991) y Laguna y Glover (1993) se implementaron diferentes algoritmos de búsqueda tabú y se llegaron a resolver instancias de hasta 100 trabajos. En Feo et al. (1996) se resuelve el mismo problema con un algoritmo GRASP. Con este algoritmo, se resuelven instancias de hasta 165 trabajos y se comparan sus resultados con los métodos propuestos en los trabajos anteriores, mejorando los resultados de la búsqueda tabú y consiguiendo soluciones muy cercanas a las óptimas (o a cotas inferiores en aquellas instancias donde no se puede probar optimalidad) en muy poco tiempo de cómputo.

En Ovacik y Uzhoy (1993) se estudia un problema de máquinas paralelas idénticas con necesidad de ajustes que dependen de la secuencia. En este trabajo, los autores calculan cotas superiores al problema de minimizar el *makespan* asumiendo el peor escenario posible. Para este mismo problema, Kurz y Askin (2001) proponen un modelo de programación entera y cuatro algoritmos heurísticos diferentes para resolverlo. De los algoritmos propuestos, el que mejores resultados consigue es el algoritmo llamado Heurística de Múltiple Inserción

(*Multiple Insertion Heuristic*).

Por otro lado, en Lee y Guignard (1996) podemos encontrar un estudio para el problema con máquinas paralelas no relacionadas y ajustes asociados a la máquina en la que se procesa el trabajo. Los autores proponen primero una relajación lagrangiana, seguida de una descomposición lagrangiana para calcular cotas inferiores al problema de minimizar el *makespan*. Otro problema similar es el estudiado en Lee y Pinedo (1997), en el que se estudian máquinas paralelas idénticas con ajustes dependientes de la secuencia, pero con el objetivo de minimizar la suma de la tardanza ponderada. Para resolver el problema se utiliza una heurística que primero obtiene soluciones iniciales mediante reglas de despacho, y luego procesa esas soluciones con un algoritmo de Recocido Simulado (*Simulated Annealing*). En Kim et al. (2002) también se propone un algoritmo de recocido simulado para resolver el mismo problema, pero con el objetivo de minimizar la tardanza total.

Helal et al. (2006) analizan el problema con máquinas paralelas no relacionadas y ajustes dependientes de la secuencia y de la máquina con el objetivo de minimizar el *makespan*. Los autores resuelven el problema con una búsqueda tabú de dos fases; en la primera fase se realizan perturbaciones intra-máquinas a las soluciones, mientras que en la segunda fase se realizan perturbaciones entre-máquinas. En este trabajo llegan a resolver instancias de hasta 100 trabajos y 10 máquinas. Este mismo problema es trabajado en Rabadi et al. (2006), donde proponen una nueva metaheurística con la que llegan a resolver problemas de hasta 120 trabajos y 12 máquinas.

En Paula et al. (2007) se estudia un problema similar, pero con el objetivo de minimizar la suma del *makespan* más los retrasos ponderados. En este trabajo proponen una metaheurística VNS (*Variable Neighborhood Search*) con la que resuelven el problema con máquinas paralelas no idénticas con necesidad de ajustes y no relacionadas con necesidad de ajustes. Los autores comparan su metaherística con tres algoritmos GRASP y muestran que el algoritmo VNS propuesto da mejores resultados en los dos problemas estudiados.

Continuando con los estudios de máquinas paralelas no relacionadas con necesidad de

ajustes, en Arnaout et al. (2009) se desarrolla una metaherística de colonia de hormigas (*Ant Colony*) con la que mejoran los resultados obtenidos en Helal et al. (2006) y en Rabadi et al. (2006). Vallada y Ruiz (2011) proponen un modelo matemático y un algoritmo genético para resolver el problema. En este trabajo, los autores comparan su algoritmo con los propuestos previamente en Kurz y Askin (2001) y en Rabadi et al. (2006), obteniendo mejores resultados en todos los tamaños de instancias desarrolladas. Posteriormente, en Arnaout et al. (2014) se propone un segundo algoritmo de colonia de hormigas con el que mejoran el propuesto en Arnaout et al. (2009), además de mejorar los resultados de las metaheurísticas propuestas en Rabadi et al. (2006) y en Kim et al. (2002). Avalos-Rosales et al. (2014) proponen un modelo de programación lineal entera mixta y un algoritmo multipartida con dos búsquedas locales diferentes, mejorando tanto el modelo matemático como el algoritmo genético presentados en Vallada y Ruiz (2011). Con el modelo matemático propuesto llegan a resolver óptimamente instancias de hasta 60 trabajos y 4 máquinas. Siguiendo con el mismo problema, en Diana et al. (2015) se presenta un nuevo algoritmo genético basado en el algoritmo *Clonal Selection* propuesto por De Castro y Von Zuben (2002). Las soluciones iniciales del algoritmo genético se generan con un algoritmo GRASP. A continuación, las mejores soluciones tienen más probabilidad de ser seleccionadas en la fase de clonación. Los autores comparan los resultados del algoritmo propuesto con el algoritmo genético propuesto en Vallada y Ruiz (2011) y encuentran mejores soluciones en todos los tamaños de instancias estudiadas. En Allahverdi (2015) se puede encontrar una revisión bibliográfica reciente de problemas de secuenciación de máquinas con necesidad de ajustes. Es importante mencionar que en esta revisión no solo se tienen en cuenta los problemas de máquinas paralelas, sino también los problemas de secuenciación de máquinas más estudiados a lo largo de los años. Finalmente, cerrando el tema de máquinas paralelas con necesidad de ajustes con el objetivo de minimizar el *makespan*, en Fanjul-Peyró et al. (2019) se presentan modelos de programación lineal entera mixta y algoritmos basados en programación matemática que mejoran considerablemente los modelos matemáticos existentes y con los que logran resolver instancias de hasta 1000

trabajos y 8 máquinas con distancias relativas muy cercanas a cotas inferiores.

A pesar de que los problemas de máquinas paralelas con necesidad de recursos adicionales han sido menos estudiados, podemos encontrar algunos trabajos en la literatura. Chen (2005) presenta un problema de máquinas paralelas no relacionadas con ajustes y necesidad de recursos adicionales con el objetivo de minimizar el *makespan*, en el que se dividen los trabajos a secuenciar en diferentes clases o tipos de trabajos. En este problema, los recursos adicionales son necesarios si el trabajo secuenciado en una máquina es de diferente tipo al trabajo que lo precede en la misma máquina. Para resolver el problema se propone una heurística que consta de una fase constructiva basada en una serie de reglas de despacho, seguida de una búsqueda tabú usada para mejorar las soluciones obtenidas en la fase constructiva.

Otro trabajo interesante que tiene en cuenta recursos adicionales es el de Ruiz-Torres et al. (2007). En este trabajo se estudia un problema de máquinas paralelas idénticas sin ajustes, en el que se asignan recursos a cada máquina para acelerar el procesamiento de los trabajos, con el objetivo de minimizar el número de trabajos con tardanza. Los autores presentan dos versiones del problema; una en la que los trabajos son previamente asignados a las máquinas, el cual resuelven con un modelo matemático, y otro caso en el que no se tienen en cuenta asignaciones previas y es resuelto con un algoritmo heurístico. Este mismo problema, pero con el objetivo de minimizar el *makespan*, es estudiado en Edis y Oguz (2012). En este trabajo, los autores adaptan una técnica utilizada en Edis y Ozkarahan (2012) que consiste en combinar programación entera y programación por restricciones (*Constraint Programming*), mostrando que este enfoque es mucho más rápido y eficiente para resolver el problema que un enfoque en el que solo se usa programación lineal entera. Una completa revisión bibliográfica de problemas de máquinas paralelas con necesidad de recursos se puede encontrar en Edis et al. (2013), donde solo se estudian problemas sin necesidad de ajustes.

En cuanto a problemas con necesidad de ajustes y de recursos adicionales, en Ruiz y Andrés-Romano (2011) se estudia un problema de máquinas paralelas no relacionadas con ajustes dependientes de la secuencia y recursos adicionales que se asignan a los ajustes para

acelerar su procesamiento, con el objetivo de minimizar la combinación lineal entre *makespan* y consumo de recursos. Para resolver el problema se propone un modelo matemático y algoritmos heurísticos, mostrando que estos algoritmos pueden encontrar buenas soluciones en muy poco tiempo de cómputo. Bitar et al. (2014) proponen un algoritmo memético (*Memetic Algorithm*) para un problema de máquinas paralelas no relacionadas con ajustes y necesidad de recursos, donde los trabajos son agrupados según el tipo de recurso que requieran para ser procesados y no se puede procesar más de un trabajo del mismo grupo a la vez. Un problema similar, pero con el objetivo de minimizar el *makespan* es estudiado en Afzalirad y Rezaeian (2016). Para resolver el problema, los autores proponen un modelo de programación entera, un algoritmo genético y un algoritmo basado en un sistema inmunitario artificial (AIS por sus siglas en inglés).

Volviendo a los problemas que no tienen en cuenta la necesidad de ajustes, en los últimos años se pueden encontrar algunos estudios de máquinas paralelas con necesidad de recursos adicionales. En Zheng y Wang (2016) se estudia un problema de máquinas paralelas no relacionadas en la que los recursos son asignados al procesamiento de los trabajos y son restrictivos y renovables, es decir, no se puede utilizar más de un cierto número de recursos en cada instante de tiempo y una vez termina su uso, vuelven a estar disponibles. Para resolver el problema se propone una metaheurística y un modelo matemático. El mismo problema es estudiado en Fanjul-Peyró et al. (2017), donde los autores proponen modelos matemáticos y matheurísticas para resolver el problema, mejorando resultados obtenidos por modelos matemáticos adaptados de otros estudios. Posteriormente, esos resultados son mejorados en Arbaoui y Yalaoui (2018), donde utilizan programación con restricciones (*Constraint Programming*) para resolver el problema. En Villa et al. (2018) se presentan varios algoritmos heurísticos que mejoran los resultados obtenidos por los modelos matemáticos y las matheurísticas propuestas en Fanjul-Peyró et al. (2017). En Vallada et al. (2019) proponen metaheurísticas con las que mejoran los resultados de trabajos anteriores.

Por último, el problema que se estudia en esta tesis doctoral es presentado en Yepes-

Borrero et al. (2020). En este problema, se tienen ajustes dependientes de la secuencia y recursos restrictivos y renovables que son asignados a esos ajustes. Para resolver el problema, se propone un modelo matemático y un algoritmo GRASP que muestra ser eficiente al compararlo con adaptaciones de buenos algoritmos propuestos para el problema sin necesidad de recursos. Recientemente, en Fanjul-Peyró (2020) se presenta un modelo de programación lineal entera mixta y un algoritmo exacto, basado en programación por restricciones, para diferentes combinaciones de problemas de máquinas paralelas con necesidad de recursos (recursos en los ajustes, recursos en el procesamiento de los trabajos, recursos en los ajustes y en el procesamiento de los trabajos). Con el enfoque propuesto, se logran resolver instancias de hasta 400 trabajos con soluciones muy cercanas a la cota inferior. La Tabla 2.1 muestra un resumen con los trabajos más relevantes de máquinas paralelas con necesidad de ajustes y/o recursos adicionales.

Tabla 2.1: Tabla resumen de trabajos de máquinas paralelas con ajustes y/o recursos adicionales

Autores/Año	**Tipo máquinas**	**Ajustes/Recursos**	**Objetivo**
Barnes y Vanston (1981)	Una máquina	Sí/No	Minimizar *makespan*
Laguna et al. (1991)	Una máquina	Sí/No	Minimizar *makespan*
Laguna y Glover (1993)	Una máquina	Sí/No	Minimizar *makespan*
Feo et al. (1996)	Una máquina	Sí/No	Minimizar *makespan*
Ovacik y Uzhoy (1993)	Paralelas idénticas	Sí/No	Minimizar *makespan*
Kurz y Askin (2001)	Paralelas idénticas	Sí/No	Minimizar *makespan*

Continúa en la siguiente página

Tabla 2.1 – *Continuación de la página anterior*

Autores/Año	**Tipo máquinas**	**Ajustes/Recursos**	**Objetivo**
Lee y Guignard (1996)	Paralelas no relacionadas	Sí/No	Minimizar *makespan*
Lee y Pinedo (1997)	Paralelas idénticas	Sí/No	Minimizar tardanza ponderada
Kim et al. (2002)	Paralelas idénticas	Sí/No	Minimizar tardanza total
Chen (2005)	Paralelas no relacionadas	Sí/Sí	Minimizar *makespan*
Helal et al. (2006)	Paralelas no relacionadas	Sí/No	Minimizar *makespan*
Rabadi et al. (2006)	Paralelas no relacionadas	Sí/No	Minimizar *makespan*
Paula et al. (2007)	Paralelas no relacionadas	Sí/No	Minimizar *makespan* + retrasos
Arnaout et al. (2009)	Paralelas no relacionadas	Sí/No	Minimizar *makespan*
Vallada y Ruiz (2011)	Paralelas no relacionadas	Sí/No	Minimizar *makespan*
Edis y Oguz (2012)	Paralelas idénticas	No/Sí	Minimizar *makespan*
Arnaout et al. (2014)	Paralelas no relacionadas	Sí/No	Minimizar *makespan*
Avalos-Rosales et al. (2014)	Paralelas no relacionadas	Sí/No	Minimizar *makespan*
Diana et al. (2015)	Paralelas no relacionadas	Sí/No	Minimizar *makespan*
Afzalirad y Rezaeian (2016)	Paralelas no relacionadas	Sí/Sí	Minimizar *makespan*

Continúa en la siguiente página

Tabla 2.1 – *Continuación de la página anterior*

Autores/Año	**Tipo máquinas**	**Ajustes/Recursos**	**Objetivo**
Zheng y Wang (2016)	Paralelas no relacionadas	No/Sí	Minimizar *makespan*
Fanjul-Peyró et al. (2017)	Paralelas no relacionadas	No/Sí	Minimizar *makespan*
Arbaoui y Yalaoui (2018)	Paralelas no relacionadas	No/Sí	Minimizar *makespan*
Villa et al. (2018)	Paralelas no relacionadas	No/Sí	Minimizar *makespan*
Fanjul-Peyró et al. (2019)	Paralelas no relacionadas	Sí/No	Minimizar *makespan*
Vallada et al. (2019)	Paralelas no relacionadas	No/Sí	Minimizar *makespan*
Yepes-Borrero et al. (2020)	Paralelas no relacionadas	Sí/Sí	Minimizar *makespan*
Fanjul-Peyró (2020)	Paralelas no relacionadas	Sí/Sí	Minimizar *makespan*

2.2. Problemas de secuenciación de máquinas multiobjetivo

Debido a que los problemas de secuenciación de máquinas multi-objetivo han sido mucho menos estudiados en la literatura que los problemas con un solo objetivo, en esta sección tendremos en cuenta diferentes variantes de problemas de secuenciación de máquinas relevantes para el estudio del problema propuesto en esta tesis, no solo aquellos de máquinas paralelas.

Podemos encontrar algunas revisiones bibliográficas en las que se estudian problemas de

secuenciación de máquinas multi-objetivo en Nagar et al. (1995), T'kindt y Billaut (2001) y en Jones et al. (2002). En estas revisiones se tratan pocos problemas de máquinas paralelas, sin embargo, en Hoogeveen (2005) se encuentra una revisión un poco más reciente donde se estudian en su gran mayoría artículos de una sola máquina, o máquinas paralelas multi-objetivo. Dos revisiones bibliográficas más recientes, pero principalmente sobre el problema de taller de flujo (*Flow-shop*) se encuentran en Yenisey y Yagmahan (2014) y en la tesis doctoral de Minella (2014).

En Cochran et al. (2003) se estudia un problema de máquinas paralelas con el objetivo de minimización del *makespan* y la tardanza ponderada. Los autores proponen un algoritmo genético de múltiples poblaciones (MPGA), con el que mejoran los resultados obtenidos con una adaptación del algoritmo genético multi-objetivo (MOGA) propuesto en Murata et al. (1996). El algoritmo propuesto es probado también para resolver el problema con tres objetivos, agregando a los dos anteriores la minimización de los tiempos de terminación ponderados, donde también obtiene mejores resultados que el algoritmo MOGA. Para resolver el mismo problema, Bandyopadhyay y Bhattacharya (2013) proponen una modificación del algoritmo NSGA-II (*Non-dominated Sorting Genetic Algorithm-II*), propuesto originalmente en Deb et al. (2002). En este trabajo los autores comparan su algoritmo con el NSGA-II original, mostrando que la modificación propuesta funciona mejor para la resolución de este problema.

En Choobineh et al. (2006) se estudia el problema con una sola máquina y necesidad de ajustes con el objetivo de minimizar el *makespan* y la tardanza total. Se propone un algoritmo multi-objetivo de búsqueda tabú, con el que obtienen soluciones Pareto óptimas, o muy cercanas a la frontera Pareto óptima obtenida por un modelo matemático propuesto también en este trabajo.

Torabi et al. (2013) estudian un problema de máquinas paralelas no relacionadas con necesidad de ajustes y de recursos que se asignan al procesamiento de los trabajos, además de incertidumbre en los tiempos de proceso. Los recursos son limitados, por lo que para poder

procesar un trabajo en un instante de tiempo, debe haber disponibilidad. Para minimizar el tiempo de flujo total, la tardanza total y la variación total de la carga de las máquinas, los autores proponen un algoritmo multi-objetivo tipo *Particle Swarm Optimization* (MOPSO) con el que se muestra que logran resolver de forma eficiente diferentes tamaños de instancias del problema.

Siguiendo con problemas que tienen en cuenta recursos adicionales, Wang y Liu (2015) adaptan el algoritmo NSGA-II para resolver un problema de máquinas paralelas con mantenimientos preventivos y recursos adicionales en estos mantenimientos. El objetivo en este problema es la minimización del *makespan* y de la inhabilidad de las máquinas debido al mantenimiento.

Otro problema similar es el propuesto en Liu y Tsai (2015). En este problema de máquinas paralelas, los tiempos de proceso se pueden acelerar aumentando el coste de la producción. Para minimizar el coste total de producción y la tardanza total, los autores proponen un modelo matemático que luego extienden para agregar la minimización del *makespan* como tercer objetivo a optimizar.

Rostami et al. (2015) proponen un modelo basado en *constraint programming* para resolver un problema de máquinas paralelas con el objetivo de minimizar el *makespan* y el *earliness/tardiness* total. En este estudio, los trabajos pueden deteriorarse dependiendo del tiempo que estén esperando en cola a ser procesados. Para evaluar el modelo propuesto, los autores adaptan un algoritmo de ramificación y acotación (*Branch and Bound*) multi-objetivo, mostrando que el modelo es más eficiente, especialmente en instancias de gran tamaño.

En estudios más recientes, podemos encontrar el problema de máquinas paralelas no relacionadas con necesidad de ajustes y recursos adicionales en los procesos, estudiado en Manupati et al. (2017). Con el objetivo de minimizar el *makespan*, el número de trabajos con tardanza, el tiempo total de flujo y la variación total de la carga de las máquinas, los autores proponen una adaptación del algoritmo NSGA-II con el que mejoran los resultados

obtenidos por el NSGA-II original y por el algoritmo MOPSO propuesto en Torabi et al. (2013).

Se pueden encontrar otros estudios multi-objetivo de problemas diferentes a los de máquinas paralelas. Uno de los más comunes es el problema del taller de flujo (*Flow-shop*), entre los que encontramos trabajos como el de Daniels y Chambers (1990), donde se presentan algoritmos heurísticos buscando minimizar el *makespan* y la tardanza total para problemas hasta con 10 máquinas. Adicionalmente, se propone un algoritmo de ramificación y acotación (*Branch and Bound*) para resolver las instancias más pequeñas, mostrando que los algoritmos heurísticos encuentran soluciones muy cercanas a las óptimas. El mismo problema es estudiado años después en Armentano y Arroyo (2004), donde los autores proponen un algoritmo multi-objetivo basado en búsqueda tabú con varias soluciones en paralelo que manejan cada una su lista tabú independiente de las otras. Para instancias de solo dos máquinas, los autores comparan su metaheurística con el algoritmo de ramificación y acotación (*Branch and Bound*) de Daniels y Chambers (1990), mientras que para las instancias más grandes, se comparan con las metaheurísticas propuestas en Hansen (2000) y en Ishibuchi y Murata (1998). En ambos casos, la metaheurística propuesta se muestra eficiente al encontrar soluciones muy cercanas a las óptimas en las instancias de dos máquinas y, superando a las otras dos metaheurísticas en las instancias más grandes.

En Framinan y Leisten (2008) se estudia también el problema del taller de flujo, pero con el objetivo de minimizar el *makespan* y el tiempo de flujo. En este trabajo se propone un algoritmo *iterated greedy* multi-objetivo y lo comparan con el algoritmo de recocido simulado multi-objetivo propuesto en Varadharajan y Rajendran (2005). El algoritmo propuesto muestra mejores resultados en los diferentes tamaños de instancias del conocido banco de instancias propuesto originalmente en Taillard (1993).

Finalmente, Minella et al. (2011) proponen un algoritmo inspirado en el *iterated greedy* multi-objetivo de Framinan y Leisten (2008) para resolver los dos últimos problemas mencionados. Este algoritmo agrega un mecanismo de selección de soluciones a destruir y re-

construir con el que ayudan a que el algoritmo se concentre en soluciones prometedoras para mejorar la frontera de Pareto obtenida. Adicionalmente agregan un mecanismo de reinicio del algoritmo con el que evitan la rápida convergencia del algoritmo al estancarse en óptimos locales. En este trabajo, los autores hacen una comparación con los algoritmos presentados en Armentano y Arroyo (2004), Arroyo y Armentano (2005), Varadharajan y Rajendran (2005) y con el algoritmo presentado en Framinan y Leisten (2008), mejorando considerablemente los resultados de estos en diferentes indicadores de rendimiento multi-objetivo. La Tabla 2.2 muestra un resumen con los trabajos más relevantes mencionados en esta revisión bibliográfica.

Tabla 2.2: Tabla resumen de trabajos de secuenciación de máquinas multi-objetivo.

Autores/Año	**Tipo problema**	**Ajustes/Recursos**	**Objetivos**
Cochran et al. (2003)	Máquinas Paralelas no relacionadas	No/No	Minimizar *makespan* y tardanza
Choobineh et al. (2006)	Una máquina	No/No	Minimizar *makespan* y tardanza
Torabi et al. (2013)	Máquinas Paralelas no relacionadas	Sí/Sí	Minimizar flujo total, tardanza y variación total
Wang y Liu (2015)	Máquinas Paralelas no relacionadas	Sí/Sí	Minimizar *makespan* e inhabilidad en máquinas
Liu y Tsai (2015)	Máquinas Paralelas no relacionadas	No/No	Minimizar costes, tardanza y *makespan*

Continúa en la siguiente página

Tabla 2.2 – *Continuación de la página anterior*

Autores/Año	**Tipo problema**	**Ajustes/Recursos**	**Objetivos**
Manupati et al. (2017)	Máquinas Paralelas no relacionadas	Sí/Sí	Minimizar *makespan* y # trabajos con retraso
Daniels y Chambers (1990)	Taller de flujo	Sí/Sí	Minimizar *makespan* y # trabajos con retraso
Armentano y Arroyo (2004)	Taller de flujo	Sí/Sí	Minimizar *makespan* y # trabajos con retraso
Framinan y Leisten (2008)	Taller de flujo	No/No	Minimizar *makespan* y tiempo de flujo
Minella et al. (2011)	Taller de flujo	No/No	Minimizar *makespan* y tiempo de flujo

Capítulo 3

Problemas de secuenciación de máquinas paralelas

Entre la gran variedad de problemas de secuenciación de máquinas estudiados en la literatura, uno de los más importantes es el de secuenciación de máquinas paralelas. En esta sección se explicará brevemente cada una de las variaciones más estudiadas de este grupo de problemas hasta llegar al caso particular a estudiar en esta tesis doctoral. En todas las variaciones de problemas de máquinas paralelas que se explicarán en esta sección, se tiene un conjunto de máquinas paralelas $M = \{1, 2, \ldots, m\}$ para procesar un conjunto de trabajos $N = \{1, 2, \ldots, n\}$. Cada trabajo puede ser procesado por una sola máquina y todas las máquinas están disponibles desde el instante cero y en todo momento que no estén procesando un trabajo. Igualmente, cada máquina solo puede procesar un trabajo a la vez. Por otro lado, el tiempo de proceso de un trabajo $j \in N$ en una máquina $i \in M$ es conocido desde el inicio y se denota como p_{ij}. Además, una vez se inicia el procesamiento del trabajo j en la máquina i, este no puede ser interrumpido y debe ser finalizado. Entre la gran variedad de objetivos a optimizar, como la minimización de la tardanza (*tardiness*) o del adelanto (*earliness*), uno de los más estudiados a lo largo de los años, ha sido la minimización del tiempo de finalización del último trabajo en ser procesado, también llamado *makespan* ($C_{\text{máx}}$). A lo largo de esta tesis doctoral, se trabajará este objetivo, con excepción del Capítulo 5 en el que, como se explicó en la Sección 1.3, se trabajará la minimización de dos

objetivos simultáneamente.

Es importante mencionar que el caso específico más simple del problema que se trata en esta tesis doctoral, el de máquinas paralelas idénticas con objetivo de minimizar el *makespan* está clasificado como un problema $\mathcal{NP}$-Hard (Garey y Johnson, 1990), incluso el problema considerando solo dos máquinas está clasificado como $\mathcal{NP}$-Hard (Lenstra et al., 1977). Debido a esta complejidad computacional, a lo largo de esta tesis doctoral se buscará resolver los problemas estudiados con algoritmos eficientes que encuentren soluciones cercanas a las soluciones óptimas.

3.1. Máquinas paralelas idénticas

El primer y menos complejo de los problemas de secuenciación de máquinas paralelas, es el de máquinas paralelas idénticas, denotado como $P//C_{\text{máx}}$ según la notación propuesta en Graham et al. (1979). En este problema las máquinas se asumen iguales entre sí, por lo que el tiempo de proceso de un trabajo i es el mismo sin importar en qué máquina sea procesado ($p_{1j} = p_{2j} = \cdots = p_{mj}$). Como se explica en Fanjul-Peyró (2010), al ser idénticos los tiempos de proceso para un trabajo en todas las máquinas, minimizar el *makespan* equivale a equilibrar la carga de trabajo en las máquinas disponibles.

A continuación se muestra un ejemplo del problema con tres máquinas paralelas idénticas y seis trabajos a procesar. La Tabla 3.1 muestra los tiempos de proceso p_{ij} y la Figura 3.1 muestra dos posibles soluciones al problema. Se puede observar que en ambas soluciones el tiempo de finalización es el mismo, sin importar que los trabajos que se procesan en las máquinas i_1 y i_2 cambien de máquina. Otra característica importante a destacar, es que el orden dentro de cada máquina también es indiferente, lo que quiere decir que el tiempo de finalización de una máquina i (C_i), solo depende de los trabajos que tenga asignados y no de la secuencia (u orden) en la que se procesen.

	j_1	j_2	j_3	j_4	j_5	j_6
i_1	3	4	8	2	5	2
i_2	3	4	8	2	5	2
i_3	3	4	8	2	5	2

Tabla 3.1: *p_{ij} para tres máquinas paralelas idénticas.*

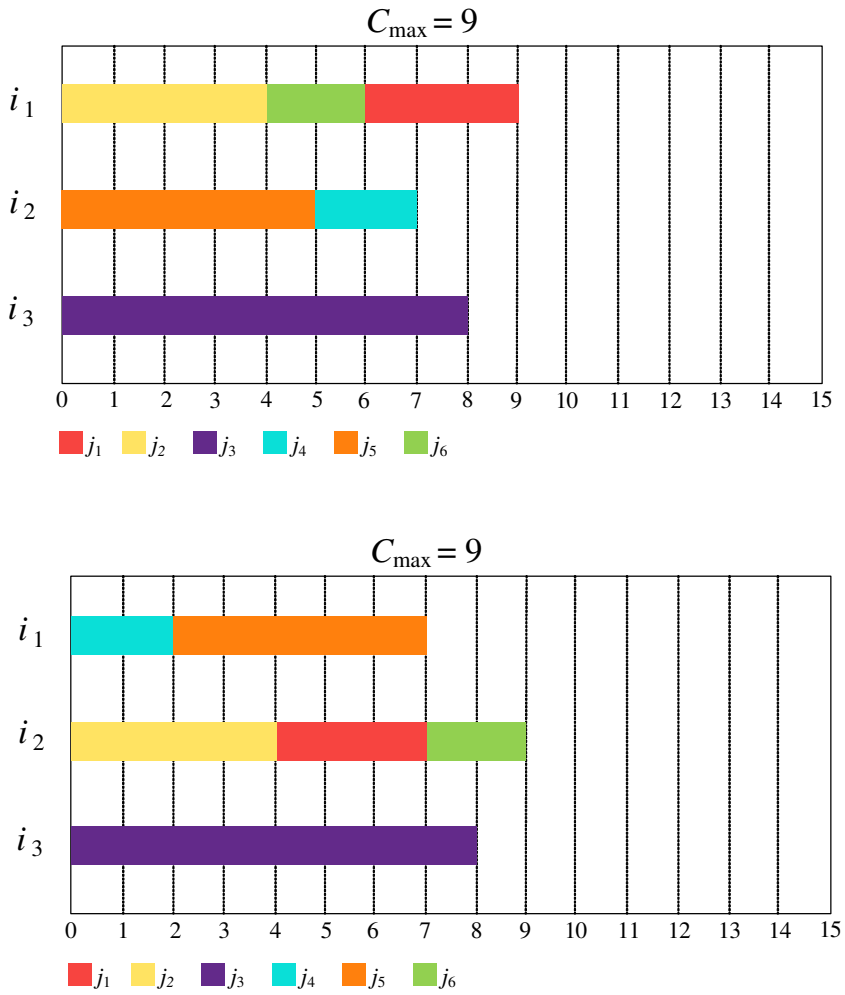

Figura 3.1: *Ejemplo de soluciones del problema de máquinas paralelas idénticas.*

3.2. Máquinas paralelas uniformemente relacionadas

Esta variación del problema, identificada en la literatura como $Q//C_{\text{máx}}$, es una generalización del caso anterior donde las máquinas tienen velocidad de procesamiento diferentes, pero proporcionales entre sí. Teniendo máquinas más rápidas que otras, una manera intuitiva de resolver el problema puede ser intentar ubicar los trabajos con mayor tiempo de procesamiento en las máquinas más rápidas.

El siguiente ejemplo muestra un problema con tres máquinas paralelas uniformemente relacionadas y seis trabajos a procesar. La Tabla 3.2 muestra los tiempos de proceso p_{ij}. En esa tabla se puede observar que la máquina 2 es dos veces más rápida que la máquina 1 y tres veces más rápida que la máquina 3.

	j_1	j_2	j_3	j_4	j_5	j_6
i_1	4	8	2	2	10	4
i_2	2	4	1	1	5	2
i_3	6	12	3	3	15	6

Tabla 3.2: *p_{ij} para tres máquinas paralelas uniformemente relacionadas.*

La Figura 3.2 muestra dos posibles soluciones al problema. Se puede observar que en este caso, sí depende en qué máquina se procesa un trabajo, ya que si en lugar de procesar el trabajo j_5 en la máquina i_2 se procesa en i_3, el tiempo de procesamiento es mucho mayor, lo que hace que el *makespan* incremente considerablemente. También es importante mencionar que, al igual que en el caso anterior, el orden (o secuencia) en que se procesen los trabajos en las máquinas no afecta el tiempo de finalización de estas.

3.3. Máquinas paralelas no relacionadas

Un caso más general del problema es el de máquinas paralelas no relacionadas, denotado en la literatura como $R//C_{\text{máx}}$. La diferencia con el caso de máquinas uniformemente relacionadas consiste en que ahora, las velocidades de las máquinas no están relacionadas y pueden variar dependiendo del trabajo que procesen. Es decir, una máquina puede ser

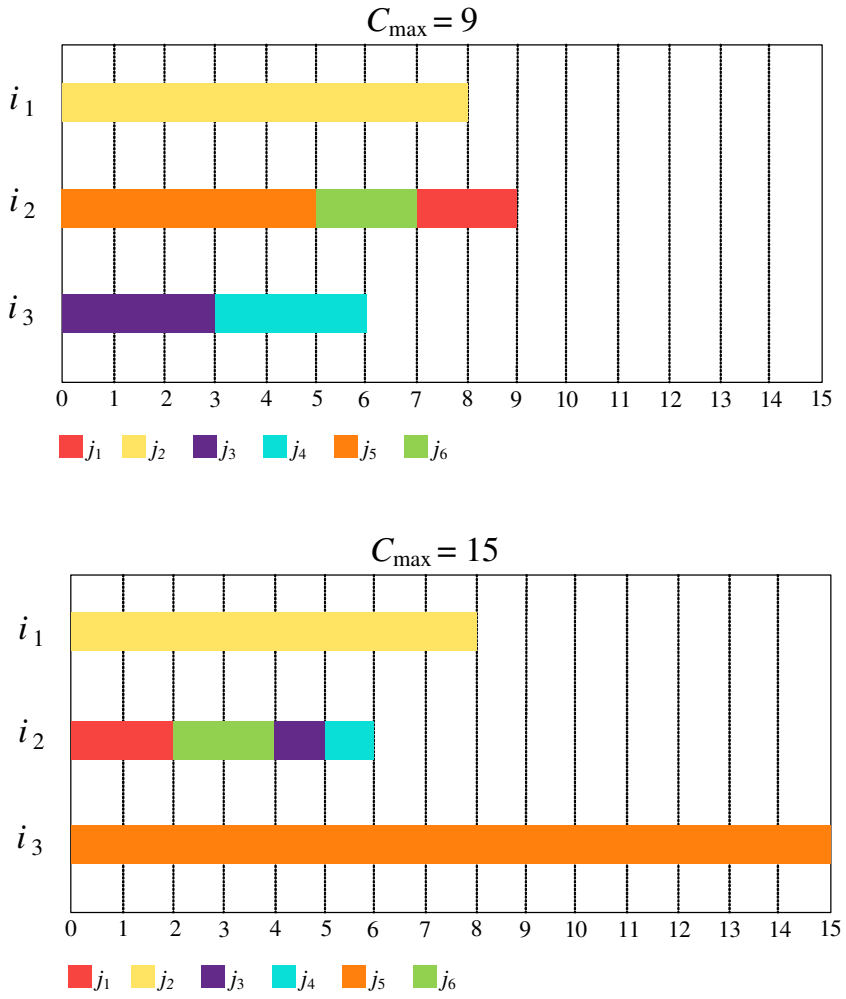

Figura 3.2: *Ejemplo de soluciones del problema de máquinas paralelas uniformemente relacionadas.*

rápida procesando un trabajo y a la vez lenta procesando otro.

En el siguiente ejemplo se muestra un problema con tres máquinas paralelas no relacionadas y seis trabajos a procesar. La Tabla 3.3 muestra los tiempos de proceso p_{ij}. En la tabla se puede observar que no hay ninguna relación en la velocidad de las máquinas.

	j_1	j_2	j_3	j_4	j_5	j_6
i_1	4	9	5	2	3	4
i_2	3	2	4	5	15	5
i_3	7	1	1	12	4	2

Tabla 3.3: *p_{ij} para tres máquinas paralelas no relacionadas.*

La Figura 3.3 muestra dos posibles soluciones al problema. Al igual que en el caso de máquinas uniformemente relacionadas, se observa que si un trabajo se procesa en dos máquinas distintas, el tiempo de procesamiento puede ser diferente. Así mismo, al igual que en los casos anteriores, la secuencia dentro de cada máquina no afecta el tiempo de finalización de la misma.

3.4. Máquinas paralelas no relacionadas con necesidad de ajustes

Una variación más recientemente estudiada en la literatura es el problema de máquinas paralelas no relacionadas con necesidad de ajustes (o *setups*), identificado como $R/s_{ijk}/C_{\text{máx}}$ en la literatura. En este problema, se agrega la necesidad de configurar las máquinas entre el procesamiento de dos trabajos sucesivos. Para presentar este problema necesitamos definir el concepto de trabajos sucesores y predecesores.

Definición 1. Se dice que un trabajo k es el (único) sucesor del trabajo j si el trabajo k se procesa después del trabajo j en la misma máquina i, y entre los trabajos j y k, la máquina i no procesa ningún otro trabajo. De igual forma, j es predecesor de k si k es el sucesor de j.

Los tiempos de ajuste dependen de la secuencia, es decir, el tiempo de ajuste necesario

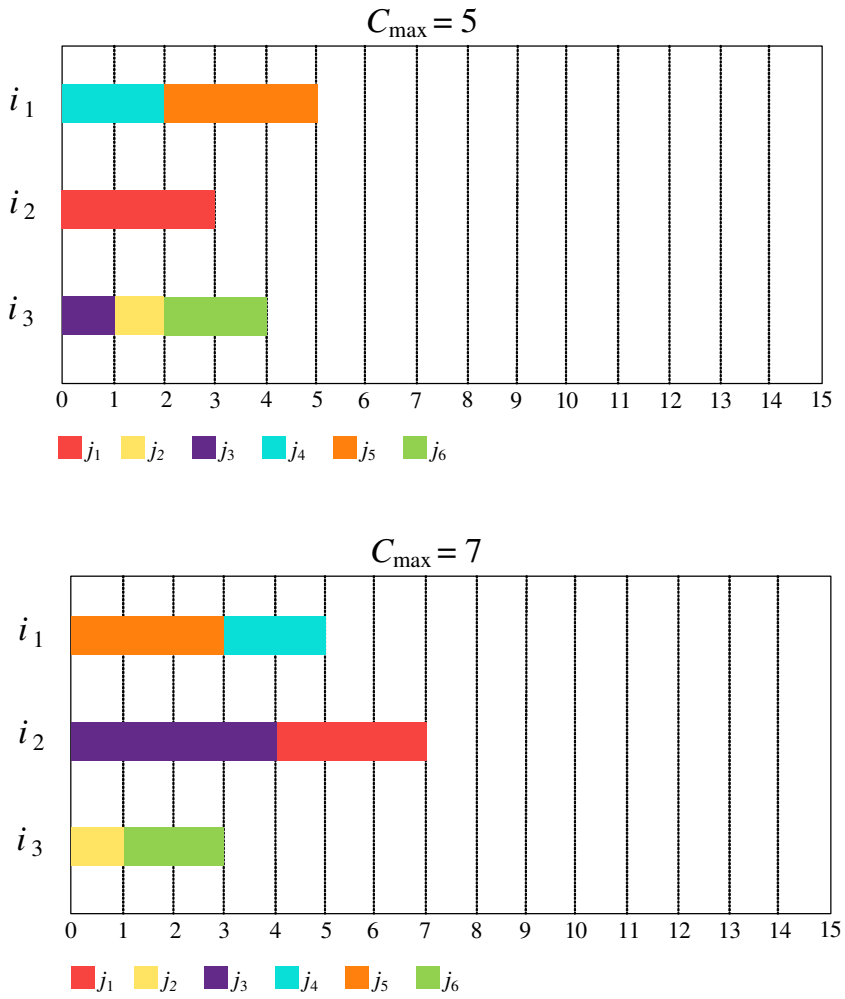

Figura 3.3: *Ejemplo de soluciones del problema de máquinas paralelas no relacionadas.*

para la máquina i entre los trabajos j y k (a partir de ahora se notará como s_{ijk}) puede ser diferente al tiempo de ajuste necesario entre k y j (s_{ikj}). Debido a esto, y a diferencia de los casos anteriores, en este problema sí es importante el orden en que se procesen los trabajos en cada máquina.

En el siguiente ejemplo se muestra un problema con tres máquinas paralelas no relacionadas con necesidad de ajustes y seis trabajos a procesar. La Tabla 3.4 muestra los tiempos de proceso p_{ij}, mientras que las Tablas 3.5, 3.6 y 3.7 muestran los tiempos de ajuste s_{ijk} en cada máquina.

	j_1	j_2	j_3	j_4	j_5	j_6
i_1	4	9	5	2	3	4
i_2	3	2	4	5	15	5
i_3	7	1	1	12	4	2

Tabla 3.4: *p_{ij} para tres máquinas paralelas no relacionadas con necesidad de ajustes.*

	j_1	j_2	j_3	j_4	j_5	j_6
j_1	0	4	5	2	5	3
j_2	3	0	8	2	4	5
j_3	2	1	0	5	1	4
j_4	3	2	3	0	3	2
j_5	8	7	2	1	0	1
j_6	3	1	8	2	4	0

Tabla 3.5: *Tiempos de ajuste en la máquina 1 (s_{1jk}).*

	j_1	j_2	j_3	j_4	j_5	j_6
j_1	0	5	1	4	2	6
j_2	2	0	7	3	4	4
j_3	1	3	0	2	2	3
j_4	6	6	8	0	1	1
j_5	2	4	1	5	0	2
j_6	4	2	2	3	5	0

Tabla 3.6: *Tiempos de ajuste en la máquina 2 (s_{2jk}).*

La Figura 3.4 muestra dos posibles soluciones al problema. En el ejemplo se puede observar la diferencia en los tiempos de finalización en las máquinas 1 y 3 al cambiar el

	j_1	j_2	j_3	j_4	j_5	j_6
j_1	0	2	1	1	4	1
j_2	2	0	5	2	2	3
j_3	3	5	0	1	3	1
j_4	5	1	1	0	3	6
j_5	2	2	5	4	0	3
j_6	6	3	5	4	2	0

Tabla 3.7: *Tiempos de ajuste en la máquina 3 (s_{3jk}).*

orden de procesamiento de los trabajos asignados. Esto quiere decir que ahora se tienen que tomar dos decisiones para cada trabajo j; la decisión de en qué máquina se procesa el trabajo, y en qué orden se procesa el trabajo con respecto a los otros trabajos asignados en la misma máquina.

3.5. Máquinas paralelas no relacionadas con tiempos de ajuste y necesidad de recursos en los ajustes

Por último, el problema que se aborda en esta tesis doctoral es el de máquinas paralelas no relacionadas con tiempos de ajuste y necesidad de recursos adicionales en los ajustes, al que llamaremos UPMSR-S por las siglas en inglés de *Unrelated Parallel Machine scheduling problem with Setup times and additional limited Resources in the Setups*. Este problema es una generalización de los explicados en las secciones anteriores, donde se busca acercarse un poco más a problemas reales en el día a día de la industria. En este caso se asume que los ajustes que se deben hacer en las máquinas para procesar un trabajo k después de un trabajo j, son realizados por recursos adicionales (por ejemplo, operarios en una planta de producción), es decir, cada ajuste (s_{ijk}) tiene asociado un número de recursos (r_{ijk}) necesario para ser realizado. Estos recursos son limitados, por lo que ahora se tiene una restricción que es el número máximo de recursos disponibles identificado como $R_{\text{máx}}$. Esta nueva restricción implica que existan soluciones no factibles, ya que es posible que existan secuencias que necesiten más recursos de los disponibles. Además de ser limitados, los recursos adicionales son renovables, es decir, en el momento en que termina la utilización del recurso adicional,

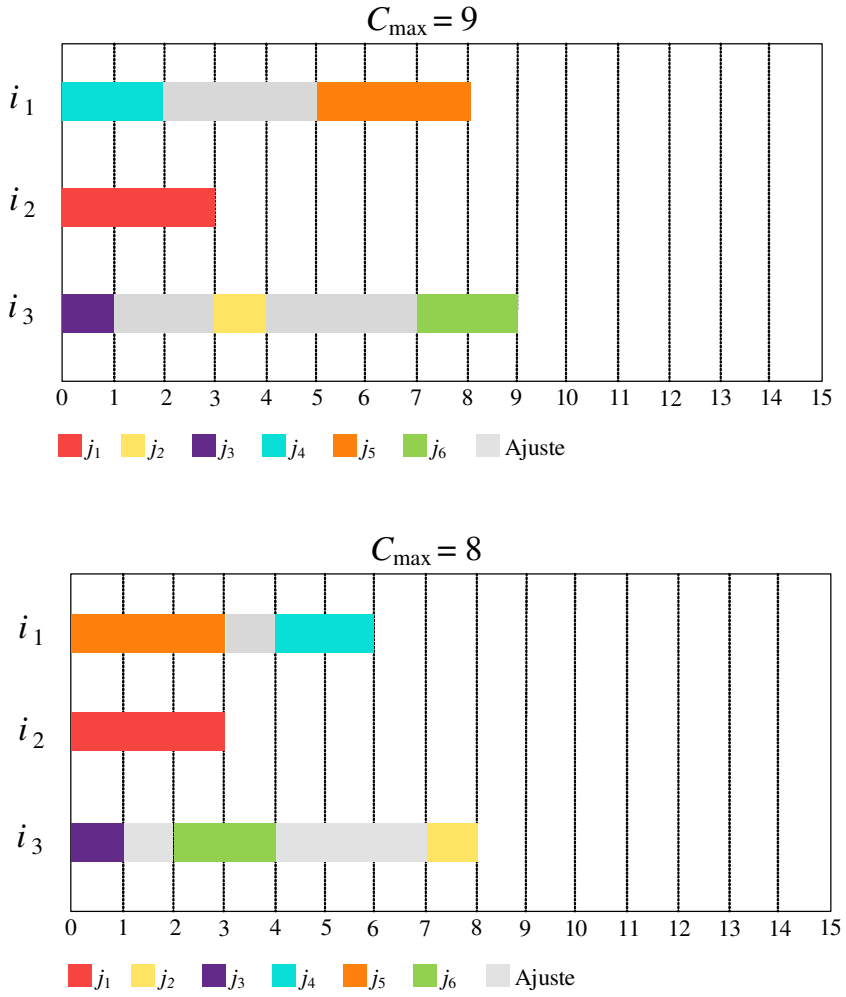

Figura 3.4: *Ejemplo de soluciones del problema de máquinas paralelas no relacionadas con necesidad de ajustes.*

este vuelve a estar disponible para ser utilizado de nuevo.

Para entender mejor el problema, se continúa con el ejemplo anterior de la Sección 3.4, agregando el consumo y la restricción de recursos. Las Tablas 3.8, 3.9 y 3.10 muestran el número de recursos r_{ijk} necesarios para realizar cada ajuste. Se supondrá que se cuenta con un número máximo de recursos disponibles $R_{\text{máx}} = 3$. Al consumo total de recursos en el instante de tiempo t se le llamará R_t.

	j_1	j_2	j_3	j_4	j_5	j_6
j_1	0	1	1	1	2	3
j_2	1	0	2	1	1	3
j_3	1	1	0	3	1	2
j_4	1	2	2	0	2	2
j_5	1	3	3	2	0	1
j_6	2	3	3	3	2	0

Tabla 3.8: *Necesidad de recursos en la máquina 1 (r_{1jk}).*

	j_1	j_2	j_3	j_4	j_5	j_6
j_1	0	1	1	1	2	3
j_2	1	0	2	3	2	3
j_3	1	3	0	2	3	2
j_4	2	3	2	0	1	3
j_5	2	3	1	3	0	2
j_6	3	3	2	3	3	0

Tabla 3.9: *Necesidad de recursos en la máquina 2 (r_{2jk}).*

	j_1	j_2	j_3	j_4	j_5	j_6
j_1	0	2	1	1	4	1
j_2	2	0	5	2	2	3
j_3	3	5	0	1	3	1
j_4	5	1	1	0	3	6
j_5	2	2	5	4	0	3
j_6	6	3	5	4	2	0

Tabla 3.10: *Necesidad de recursos en la máquina 3 (r_{3jk}).*

En la Figura 3.5 se pueden ver dos soluciones al problema, una factible y la otra no factible con el número de recursos disponibles. Se observa que la primera solución no cumple

con la restricción de recursos, ya que en el instante de tiempo 5, se deben procesar dos ajustes que requieren más recursos de los disponibles. Al existir secuencias que generan soluciones no factibles, aparece la obligación de agregar tiempos ociosos en las máquinas como se observa en la máquina i_2 de la segunda solución de la Figura 3.5. Que los ajustes no siempre se puedan procesar sin parar las máquinas, genera una nueva decisión a tomar a la hora de resolver el problema; el instante de tiempo en el que se inicia el procesamiento del ajuste.

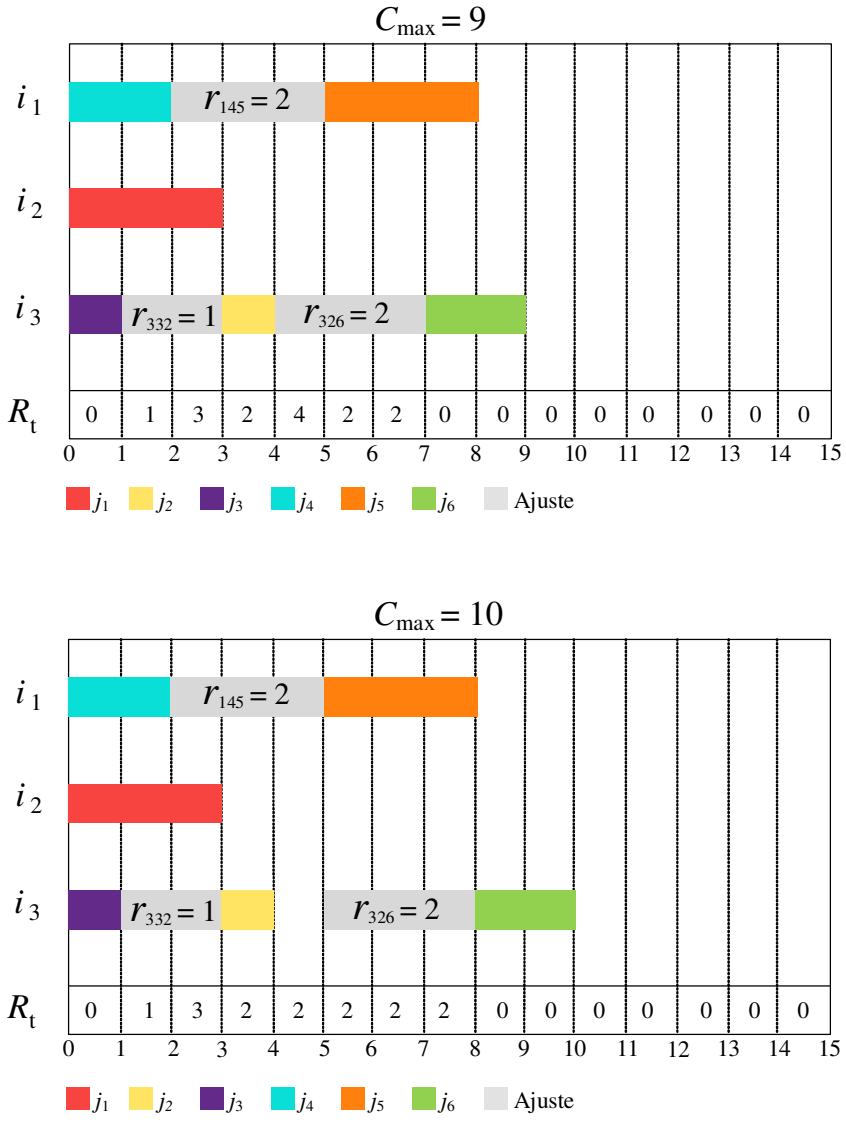

Figura 3.5: *Ejemplo de soluciones del problema de máquinas paralelas no relacionadas con tiempos de ajuste y necesidad de recursos en los ajustes.*

Capítulo 4

Problema de secuenciación de máquinas paralelas no relacionadas con necesidad de ajustes y recursos adicionales

El primer problema que resolvemos en esta tesis doctoral es el problema llamado UPMSR-S (*Unrelated Parallel Machine scheduling problem with Setup times and additional limited Resources in the Setups*) explicado en la Sección 3.5 con el objetivo de minimizar el *makespan*. En este capítulo se define formalmente el problema, se propone un modelo de programación lineal entera mixta y algoritmos eficientes para la resolución del problema UPMSR-S.

4.1. Definición formal del problema

Para definir formalmente el problema, primero es necesario introducir los siguientes conjuntos y parámetros, que se suponen conocidos y deterministas:

- Conjunto $N = \{1, \ldots, n\}$ de trabajos que deben ser procesados, indexado por las letras j, k y ℓ.
- Conjunto $M = \{1, \ldots, m\}$ de máquinas paralelas no relacionadas, indexado por la letra i.
- El parámetro p_{ij} es el tiempo de proceso del trabajo j en la máquina i.

- El parámetro s_{ijk} es el tiempo de ajuste en la máquina i entre el procesamiento de los trabajos j y k.

- El parámetro r_{ijk} es el número de recursos necesario para hacer el ajuste en la máquina i entre el procesamiento de los trabajos j y k.

- El parámetro $R_{\text{máx}}$ es el número total de recursos disponibles para procesar los ajustes.

Las m máquinas están disponibles todo el horizonte temporal a partir del instante 0 y cada máquina solo puede procesar un trabajo al mismo tiempo. En este problema no se tiene en cuenta ninguna restricción de precedencia en la secuencia, por lo que cualquier trabajo puede ser procesado en cualquier momento, una vez terminado el correspondiente ajuste de la máquina. Como se explicó en la Sección 3.5, tener un número limitado de recursos puede generar soluciones no factibles, ya que una secuencia podría requerir más recursos de los disponibles en algún instante de tiempo.

4.2. Modelo de programación lineal entera mixta

El modelo matemático propuesto para resolver el problema es el presentado en Perea et al. (2016). Antes de introducir el modelo, es necesario definir las siguientes variables:

- La variable binaria Y_{ij} que toma el valor 1 si el trabajo j es procesado en la máquina i. La variable toma el valor 0 en otro caso.

- La variable binaria X_{ijk} que toma el valor 1 si el trabajo k es sucesor del trabajo j en la máquina i. La variable toma el valor 0 en otro caso.

- La variable binaria H_{ijkt} que toma el valor 1 si el ajuste en la máquina i, entre los trabajos j y k, termina en el instante de tiempo t. La variable toma el valor 0 en otro caso.

  Adicionalmente, es necesario definir $\bar{M}$ como una constante con valor muy grande.

Además, se debe calcular el parámetro $t_{\text{máx}}$, que es un límite superior para el *makespan*. Después de calcular $t_{\text{máx}}$, se puede definir el conjunto $T = \{1, \ldots, t_{\text{máx}}\}$ de unidades de tiempo, indexado por la letra t. Así mismo, es necesario definir el conjunto $N_0 = N \cup \{0\}$, donde 0 es un trabajo ficticio con el que todas las máquinas empiezan y terminan, siendo $s_{i0k} = s_{ik0} = r_{i0k} = r_{ik0} = p_{i0} = 0$, $\forall i \in M$ y $\forall k \in N_0$.

El modelo propuesto es el siguiente:

$$\text{mín } C_{\text{máx}} \tag{4.1}$$

s.t.

$$\sum_{k \in N} X_{i0k} \leq 1, \ i \in M \tag{4.2}$$

$$\sum_{i \in M} Y_{ij} = 1, \ j \in N \tag{4.3}$$

$$Y_{ij} = \sum_{k \in N_0, j \neq k} X_{ijk}, \ i \in M, j \in N \tag{4.4}$$

$$Y_{ik} = \sum_{j \in N_0, j \neq k} X_{ijk}, \ i \in M, k \in N \tag{4.5}$$

$$\sum_{t \leq t_{\text{máx}}} H_{ijkt} = X_{ijk}, \forall i \in M, j \in N_0, k \in N, k \neq j \tag{4.6}$$

$$\sum_{t} tH_{ijkt} \geq \sum_{\ell \in N_0} \sum_{t \leq t_{\text{máx}}} H_{i\ell jt}(t + s_{ijk} + p_{ij}) - \bar{M}(1 - X_{ijk}),$$
$$\forall \ i \in M, j \in N_0, k \in N, k \neq j \tag{4.7}$$

$$\sum_{i \in M, j \in N_0, k \in N, k \neq j, t' \in \{t, \ldots, t + s_{ijk} - 1\}} r_{ijk} H_{ijkt'} \leq R_{\text{máx}}, \forall \ t \leq t_{\text{máx}} \tag{4.8}$$

$$\sum_{t \leq t_{\text{máx}}} tH_{ijkt} \leq C_{\text{máx}}, \forall i \in M, j \in N_0, k \in N_0, k \neq j \tag{4.9}$$

$$X_{ijk} \geq 0, \ Y_{ij} \geq 0, H_{ijkt} \in \{0, 1\}$$

La función objetivo (4.1) minimiza el *makespan*. La restricción (4.2) establece que se asignará como mucho un trabajo en la primera posición de cada máquina. La restricción (4.3) asegura que cada trabajo sea asignado a una sola máquina. La restricción (4.4) garantiza que cada trabajo j procesado en la máquina i solo tenga un sucesor k. Así mismo, la restricción

(4.5) asegura que cada trabajo k procesado en la máquina i solo tenga un predecesor j. La restricción (4.6) garantiza que el ajuste entre los trabajos j y k en la máquina i termina en un solo instante de tiempo menor a $t_{\text{máx}}$. La restricción (4.7) establece que el ajuste entre los trabajos j y k en la máquina i termina como mínimo después del tiempo de finalización del ajuste anterior, en la máquina i, más el tiempo de proceso del trabajo j en la máquina i. La restricción (4.8) asegura que para todo instante de tiempo, el número de recursos utilizados no sea superior a $R_{\text{máx}}$. Por último, la restricción (4.9) obliga a que el *makespan* sea mayor o igual que el instante de tiempo en el que todos los ajustes han terminado, incluyendo el ajuste ficticio entre el último trabajo procesado y el trabajo ficticio 0. Es importante mencionar que debido a la estructura del problema, las variables X y Y se pueden relajar. Como H es una variable binaria, (4.6) implica que la variable X sea entera, por lo tanto, (4.2) implica que X sea binaria. Del mismo modo, (4.4)-(4.5) implica que la variable Y sea entera, y agregando (4.3) se obliga a que Y sea binaria.

Como se verá más adelante en la Sección 4.6, este modelo solo puede resolver instancias de tamaño muy pequeño, lo cual nos obliga a buscar otros métodos de resolución eficientes para resolver instancias de gran tamaño. En las siguientes secciones se explicarán los algoritmos heurísticos y metaheurísticos que se utilizan para resolver el problema UPMSR-S con el objetivo de minimizar el *makespan*.

4.3. Algoritmos heurísticos propuestos

Para resolver el problema UPMSR-S se deben resolver los siguientes tres subproblemas:

- Problema de asignación, en el que se decide en qué máquina se va a procesar cada trabajo $j \in N$.
- Problema de secuenciación, en el que se decide el orden en el que se van a procesar los trabajos asignados en cada máquina $i \in M$.
- Problema de temporalidad, en el que se decide el instante de tiempo en el cual se inicia

el ajuste de la máquina i entre el procesamiento de los trabajos j y k.

Los algoritmos heurísticos que se proponen en esta sección se dividen en dos fases: 1) fase constructiva y 2) fase de reparación o factibilización de soluciones. La fase constructiva resuelve los subproblemas de asignación y secuenciación, mientras que la fase de reparación resuelve el problema de temporalidad.

4.3.1. Fase constructiva

Para la fase constructiva se presentan tres algoritmos siguiendo dos enfoques diferentes en el momento de construir soluciones. El primer enfoque consiste en construir soluciones ignorando la información de los recursos necesarios para procesar los ajustes, únicamente buscando buenas soluciones en términos de la minimización del *makespan*. En otras palabras, los algoritmos constructivos que siguen este enfoque buscan buenas soluciones para el problema con necesidad de ajustes sin recursos adicionales (UPMS), por lo que se adaptan dos algoritmos de estudios previos que mostraron buenos resultados en este problema. Por otro lado, el segundo enfoque sí tiene en cuenta la información del consumo de recursos, buscando generar soluciones equilibradas entre el número de recursos necesarios y el *makespan*. Es importante mencionar que las soluciones generadas con los dos enfoques pueden ser soluciones no factibles, ya que el consumo de recursos en algún instante de tiempo puede ser superior a $R_{\text{máx}}$. Por ello, es necesaria la fase de reparación que será explicada más adelante.

Además de los dos enfoques mencionados en la sección anterior, se probó un tercer enfoque en el que se busca generar soluciones factibles desde la fase constructiva, evaluando el consumo de recursos antes de cada posible asignación. Sin embargo, este enfoque fue rápidamente descartado por no ser eficiente. El algoritmo propuesto con este tercer enfoque y los resultados obtenidos son explicados en detalle en la Sección 4.5.

4.3.2. Primer enfoque de algoritmo constructivo

En este enfoque, la idea principal es probar si algoritmos que han sido propuestos con éxito para el problema sin recursos, siguen siendo buenos al agregar esta restricción. Es por esto que adaptamos los dos algoritmos que reportaron mejores resultados de entre los que se han propuesto para el problema UPMS. Los algoritmos adaptados son los propuestos en Diana et al. (2015) y en Avalos-Rosales et al. (2014).

Constructivo 1

El primer algoritmo constructivo está basado en la fase constructiva del algoritmo propuesto en Diana et al. (2015). En este algoritmo, inicialmente se crea un conjunto, llamado N^*, con todos los trabajos pendientes de asignación. Una vez creado el conjunto N^*, la idea es probar la inserción de cada uno de sus elementos en cada una de las posibles posiciones de la solución parcial, es decir, probar la inserción de los elementos antes y después de cada trabajo previamente asignado en cada una de las máquinas. En cada posible inserción, se calcula el tiempo de finalización que tendría la máquina i si se asignara ese trabajo j en esa posición k (C'_{ijk}, $\forall\ i, j, k$). Una vez todos los trabajos de N^* han sido probados en todas las posiciones, la iteración termina con la asignación del trabajo que al ser insertado genera menor C'_{ijk} y la eliminación de ese trabajo del conjunto N^*. Cada vez que un trabajo es asignado al final de cada iteración, se actualiza el tiempo de finalización en la máquina i (C_i) en la que fue asignado. Este proceso se repite hasta que el conjunto N^* se queda sin elementos. El Algoritmo 1 muestra el pseudocódigo de este algoritmo constructivo.

Constructivo 2

El segundo algoritmo constructivo diseñado está basado en el algoritmo propuesto en Avalos-Rosales et al. (2014). En este algoritmo, al igual que en el Constructivo 1, lo primero que se hace es crear el conjunto de trabajos pendientes de asignación N^* y ordenar los elementos en orden no creciente de acuerdo al tiempo medio de procesamiento en todas las máquinas ($\bar{p_j} = \sum_{i \in M} p_{ij}/m$). Una vez ordenado el conjunto N^*, se toma únicamente

Algoritmo 1: Pseudocódigo Constructivo 1.

```
1  N* ← N
2  mientras N* ≠ ∅ hacer
3  |  para cada j ∈ N* hacer
4  |  |  para cada i ∈ M hacer
5  |  |  |  Probar la inserción del trabajo j en cada posición k y calcular C'_ijk;
6  |  |  fin
7  |  fin
8  |  (i*, j*, k*) = arg mín_{i,j,k}{C'_ijk};
9  |  Asignar j* en i* en la posición k* y actualizar C_i de la máquina i*;
10 |  N* ← N* \ {j*};
11 fin
```

el primer elemento de éste, se prueba su inserción en todas las posibles posiciones de la solución parcial, calculando el tiempo de finalización que tendría la máquina i si se asignara ese trabajo j en esa posición k (C'_{ijk}). Después de probar la inserción del elemento en todas las posiciones, la iteración termina con la asignación del trabajo en la posición que al ser insertado genera menor C'_{ijk}, actualizando el tiempo de finalización en la máquina i y eliminado el trabajo del conjunto N^*. La siguiente iteración comienza tomando el siguiente trabajo del conjunto, repitiendo el proceso hasta que el conjunto N^* quede sin elementos. El Algoritmo 2 muestra el pseudocódigo del Constructivo 2.

Algoritmo 2: Pseudocódigo Constructivo 2.

```
1  N* ← N
2  mientras N* ≠ ∅ hacer
3  |  Calcular p̄_j = Σ_{i∈M} p_ij/m ∀j ∈ N*;
4  |  j* = arg máx_{j∈N*}{p̄_j};
5  |  para cada i ∈ M hacer
6  |  |  Encontrar la mejor posición k para insertar j*, y guardar C'_ijk;
7  |  fin
8  |  (i*, k*) = arg mín_{i,k}{C'_ij*k};
9  |  Asignar j* en i* en la posición k* y actualizar C_i en la máquina i*;
10 |  N* ← N* \ {j*};
11 fin
```

4.3.3. Segundo enfoque de resolución

Se puede notar que los constructivos del primer enfoque, propuestos en la sección anterior (Constructivo 1 y Constructivo 2), ignoran todo lo referente a los recursos adicionales. A diferencia del primer enfoque, en éste sí se tiene en cuenta la información de los recursos mientras se construyen las soluciones. Siguiendo este segundo enfoque, se propone un nuevo algoritmo constructivo que se explica a continuación.

Constructivo 3

La idea principal en este algoritmo es buscar soluciones equilibradas entre el *makespan* y el número total de recursos utilizados. Buscando lograr este equilibrio, se propone el cálculo de un nuevo valor, llamado λ_{ijk}, el cual tiene información del tiempo de procesamiento del trabajo j en la máquina i, del tiempo de ajuste necesario para poder procesar el trabajo j en la posición k, y del número de recursos que requiere ese ajuste. Este nuevo valor podría ser considerado como el coste de secuenciar el trabajo j en la posición k de la máquina i. El procedimiento en este algoritmo constructivo es similar al del Constructivo 1 explicado en la Sección 4.3.2, con la diferencia de que en este constructivo, en cada inserción se calcula el valor λ_{ijk} (explicado más adelante) en lugar de C'_{ijk}, asignando al final de cada iteración la inserción que genera menor λ_{ijk}. El Algoritmo 3 muestra el pseudocódigo del Constructivo 3.

Para este algoritmo, con el fin de encontrar el mejor equilibrio entre *makespan* y consumo de recursos, se probaron tres maneras diferentes de calcular λ_{ijk}, las cuales se explican a continuación.

Cálculo de λ_{ijk}

Antes de explicar las diferentes maneras de calcular λ_{ijk}, es importante aclarar que, si se tiene una secuencia en una máquina con ℓ trabajos $(j_1, j_2, \ldots, j_{k-1}, j_k, \ldots, j_\ell)$, y se inserta el trabajo j en la posición k, en general, la nueva secuencia sería $(j_1, j_2, \ldots, j_{k-1}, j, j_k, \ldots, j_\ell)$. En la nueva secuencia se puede ver que hay dos nuevos ajustes, el ajuste entre los trabajos

Algoritmo 3: Pseudocódigo Constructivo 3.

```
N* ← N
mientras N* ≠ ∅ hacer
    para cada j ∈ N* hacer
        para cada i ∈ M hacer
            Probar la inserción del trabajo j en cada posición k y calcular λ_ijk;
        fin
    fin
    (i*, j*, k*) = arg mín_{i,j,k}{λ_ijk};
    Asignar j* en i* en la posición k* y actualizar C_i en la máquina i*;
    N* ← N* \ {j*};
fin
```

j_{k-1} y j, y el ajuste entre los trabajos j y j_k. Así mismo, se puede observar que el ajuste que se hacía en la secuencia original, entre los trabajos j_{k-1} y j_k ya no es necesario hacerlo.

Para el cálculo de λ_{ijk} es necesario definir los siguientes parámetros:

- $C'_{i,j}$ es el tiempo de finalización de la máquina i, en la solución parcial, cuando el trabajo j es insertado en dicha máquina.
- $\theta_{s(i,j,k-1,k)}$ es el tiempo necesario para realizar el nuevo ajuste entre los trabajos en las posiciones $k-1$ y k, cuando el trabajo j es insertado en la máquina i en la posición k. Se define análogamente $\theta_{s(i,j,k,k+1)}$.
- $\theta_{r(i,j,k-1,k)}$ es el número de recursos necesarios para realizar el nuevo ajuste entre los trabajos en las posiciones $k-1$ y k, cuando el trabajo j es insertado en la máquina i en la posición k. Se define análogamente $\theta_{r(i,j,k,k+1)}$.
- $\gamma_{s(i,k)}$ es el tiempo del ajuste que ya no es necesario hacer, cuando se inserta un nuevo trabajo en la máquina i en la posición k.
- $\gamma_{r(i,k)}$ es el número de recursos que se necesitaban para hacer el ajuste que se deja de hacer al insertar un trabajo en la máquina i en la posición k.

Las tres propuestas para el cálculo de λ_{ijk} son las siguientes:

1. $\lambda^1_{ijk} = C'_{i,j} + p_{ij} + (\theta_{s(i,j,k-1,k)} + \theta_{r(i,j,k-1,k)}) + (\theta_{s(i,j,k,k+1)} + \theta_{r(i,j,k,k+1)}) - (\gamma_{s(i,k)} + \gamma_{r(i,k)})$. Este caso es aditivo, simplemente se suman los tiempos de ajuste nuevos en las máquinas y los recursos asociados a éstos, y se restan los ajustes que no se deben hacer y sus respectivos recursos asociados.

2. $\lambda^2_{ijk} = C'_{i,j} + p_{ij} + (\theta_{s(i,j,k-1,k)} \cdot \theta_{r(i,j,k-1,k)}) + (\theta_{s(i,j,k,k+1)} \cdot \theta_{r(i,j,k,k+1)}) - (\gamma_{s(i,k)} \cdot \gamma_{r(i,k)})$. En este caso, multiplicando los tiempos de los nuevos ajustes necesarios con sus recursos asociados, se busca "penalizar" las inserciones que tengan tiempos de ajuste muy altos y/o consumo de recursos altos.

3. $\lambda^3_{ijk} = C'_{i,j} + p_{ij} \cdot ((\theta_{s(i,j,k-1,k)} + \theta_{r(i,j,k-1,k)}) + (\theta_{s(i,j,k,k+1)} + \theta_{r(i,j,k,k+1)}))/(\gamma_{s(i,k)} + \gamma_{r(i,k)})$. Este caso se busca una combinación de los dos casos anteriores, mezclando un modelo aditivo y uno multiplicativo.

En la Sección 4.6 se prueban las tres maneras de calcular λ_{ijk} para seleccionar la que tenga mejores resultados en la minimización del *makespan*.

Es importante decir que, a pesar de tener en cuenta el consumo de recursos en la construcción de las soluciones, este algoritmo, al igual que los dos del primer enfoque, no garantiza que las soluciones cumplan con la restricción de recursos. Por lo tanto, a todos los algoritmos constructivos propuestos en esta sección, se les agrega la fase de reparación o de factibilización de soluciones explicada en la siguiente sección.

4.3.4. Fase de reparación de soluciones

Una vez todos los trabajos son asignados y secuenciados en las máquinas, es necesario evaluar el consumo de recursos en cada instante de tiempo para verificar si la solución es factible (satisface que el consumo de recursos no supere $R_{\text{máx}}$ en ningún instante de tiempo). En caso de que el consumo de recursos sea mayor a $R_{\text{máx}}$ en algún instante de tiempo, la solución deberá ser reparada.

Para reparar las soluciones, se propone un algoritmo que identifica, entre todos los ajustes que están ejecutándose en el instante de tiempo en el que se incumple la restricción, el ajuste

que inicie más tarde. El inicio de este ajuste se pospone hasta la finalización del primero de los ajustes que se estaban ejecutando cuando se generó el problema. Una vez pospuesto el inicio del ajuste, se regresa al procedimiento de evaluación de recursos desde el instante de tiempo en el que se había generado el problema. Si la restricción de recursos se satisface, se continúa la evaluación de recursos en los instantes de tiempo siguientes y se repite el proceso de reparación en caso de ser necesario. En caso de que haya dos o más ajustes que empiecen al mismo tiempo, se pospone el inicio del ajuste en la máquina que tenga menor C_i. En la Figura 4.1 se muestra un ejemplo de la fase de reparación en una solución. En el ejemplo se tiene un $R_{\text{máx}}$ de 2, por lo que en ningún instante de tiempo se pueden utilizar más de 2 recursos adicionales. La Figura 4.1 a) muestra la secuencia original antes de ser reparada. El procedimiento de reparación comenzaría a evaluar el consumo de recursos desde el primer instante de tiempo donde el consumo de recursos es cero ($R_t = 0$). Continúa con el instante de tiempo dos donde el consumo de recursos es de uno, por lo que la restricción de recursos se satisface ($R_t \leq R_{\text{máx}}$) y el algoritmo continúa con el siguiente instante de tiempo. Si observamos el siguiente instante de tiempo en la Figura 4.1 a), nos damos cuenta que el consumo de recursos es de 3, por lo que se supera el máximo de recursos permitido y se debe reparar la solución. Al encontrar un incumplimiento de la restricción de recursos, se pospone el inicio del ajuste en la máquina que haya iniciado más tarde (entre las máquinas que estén procesando ajustes en ese instante de tiempo). En el ejemplo, vemos que el ajuste de la máquina i_1 empezó después que el ajuste de la máquina i_3, por lo que se pospone su inicio hasta que el ajuste en la máquina i_3 haya terminado. La Figura 4.1 b) muestra la solución después de posponer el inicio del ajuste en la máquina i_1. El algoritmo continúa con la evaluación del consumo de recursos desde el instante de tiempo en el que se había superado anteriormente el número máximo de recursos permitido. Si nos fijamos en la Figura 4.1 b), nos damos cuenta que ahora se incumple la restricción de recursos en el instante de tiempo 5, ya que los ajustes que se están realizando en las máquinas i_1 e i_3 necesitan 2 recursos cada uno. A continuación, se pospone el inicio del ajuste de la máquina i_3, ya que

es el que empieza más tarde entre los ajustes que se están ejecutando en el instante 5. La Figura 4.1 c) muestra la solución después posponer el ajuste en la máquina i_3. En la figura podemos observar que la solución ahora es factible, ya que en todos los instantes de tiempo se cumple la restricción de recursos. El Algoritmo 4 muestra el pseudocódigo del algoritmo de reparación propuesto.

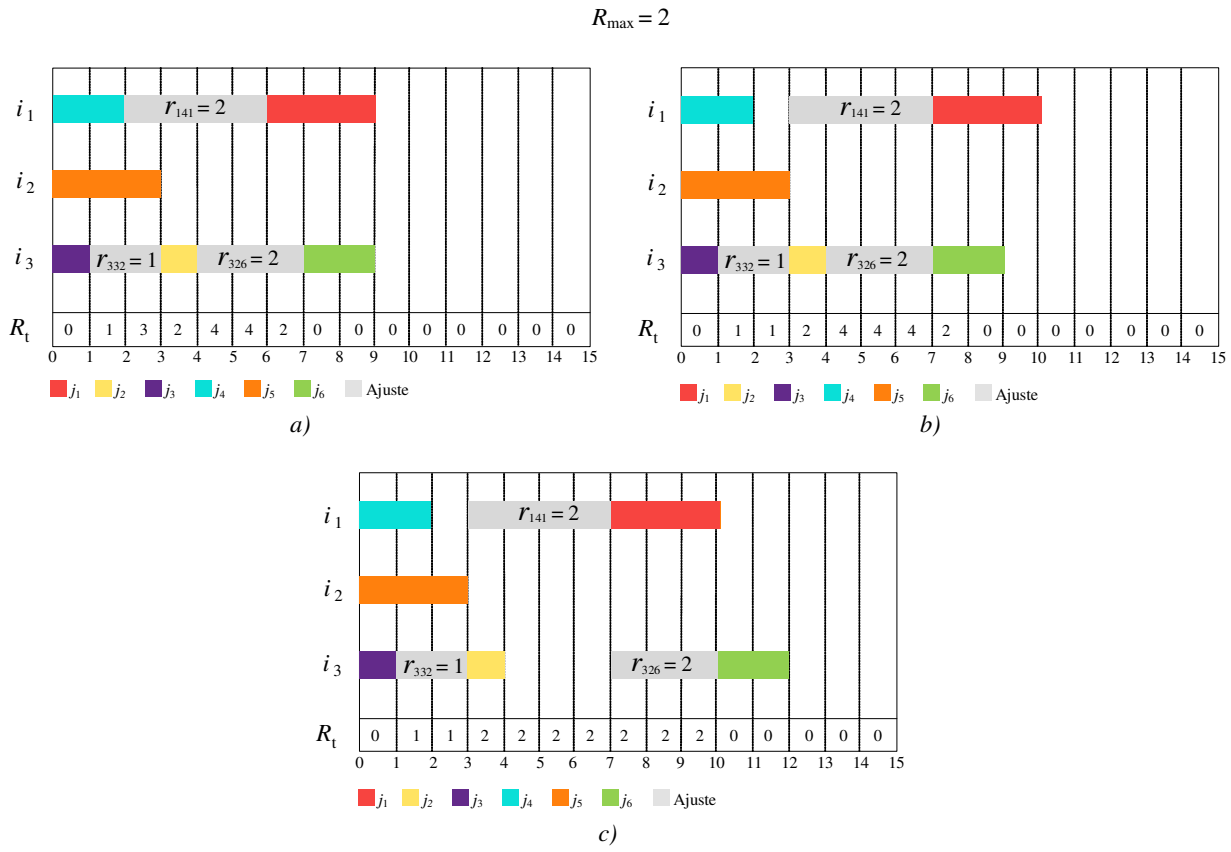

Figura 4.1: *Ejemplo de la fase de reparación de soluciones con* $R_{\text{máx}} = 2$.

Si combinamos los algoritmos constructivos propuestos con las tres formas propuestas de calcular λ y el algoritmo de reparación, tenemos los siguientes cinco algoritmos heurísticos para resolver el problema UPMSR-S:

- Heurística 1: Constructivo 1 + Algoritmo de reparación.
- Heurística 2: Constructivo 2 + Algoritmo de reparación.

Algoritmo 4: Algoritmo de reparación

```
para t < C_máx hacer
    Evaluar el consumo de recursos en el instante de tiempo t;
    si Consumo de recursos >R_máx entonces
        Posponer el inicio del ajuste en la máquina que empieza más tarde entre
         todas las máquinas que ejecutan ajustes en el instante de tiempo t;
        Actualizar C_máx;
    fin
fin
```

- Heurística 3: Constructivo 3 (con λ^1_{ijk}) + Algoritmo de reparación.
- Heurística 4: Constructivo 3 (con λ^2_{ijk}) + Algoritmo de reparación.
- Heurística 5: Constructivo 3 (con λ^3_{ijk}) + Algoritmo de reparación.

4.4. Algoritmo GRASP

Como se verá en la sección de resultados computacionales, en las instancias en las que se pueden obtener soluciones óptimas con el modelo de programación lineal entera mixta, los algoritmos heurísticos propuestos en la sección anterior obtienen soluciones lejanas a dichos óptimos. Por ello, en esta sección se propone un método multi-partida para obtener mayor variedad de soluciones con cada algoritmo propuesto. En concreto, diseñaremos algoritmos GRASP (*Greedy Randomized Adaptive Search Procedure*). Los algoritmos GRASP fueron propuestos originalmente por Feo y Resende (1989), y desde entonces han sido implementados con éxito en muchos estudios previos sobre secuenciación de máquinas (ver Feo et al. (1991), Aiex et al. (2003), Binato et al. (2002), Rajkumar et al. (2011), entre otros). El algoritmo GRASP tiene dos fases principales: la aleatorización de la fase constructiva y la búsqueda local.

La idea principal en el algoritmo GRASP es generar muchas soluciones diferentes, cada una guiada por la regla constructiva propuesta, pero con un grado de aleatoriedad para obtener variedad de soluciones. El algoritmo se ejecuta durante un tiempo establecido

quedándose con la mejor solución encontrada cuando el tiempo de ejecución termina. El Algoritmo 5 muestra el procedimiento general del algoritmo GRASP propuesto. Es importante mencionar que no se pone la fase de reparación en el pseudocódigo, ya que, como se explicará más adelante, en el procedimiento de búsqueda local propuesto está incluida esta fase.

Algoritmo 5: Pseudocódigo algoritmo GRASP.

```
1 MejorSolucion = BigNumber
2 mientras TiempoEjecucion < LimiteTiempo hacer
3 |   Solucion ← ∅
4 |   Solucion ← FaseConstructivaAleatorizada
5 |   Solucion ← BusquedaLocal
6 |   si Solucion < MejorSolucion entonces
7 |   |   MejorSolucion ← Solucion
8 |   fin
9 fin
```

A continuación, veremos con detalle cada una de las partes que integran el GRASP implementado.

4.4.1. Aleatorización de la fase constructiva

La aleatorización durante la construcción de soluciones ha mostrado ser muy útil para obtener variedad de soluciones y para evitar quedar atrapado en óptimos locales. El proceso de aleatorización propuesto en este trabajo consiste en, para cada algoritmo propuesto en la Sección 4.3, no asignar en cada iteración el mejor candidato de acuerdo a la regla de asignación, sino elegir aleatoriamente una asignación entre una lista de candidatos restringida llamada RCL (*Restricted Candidate List*). El tamaño de RCL depende de un valor α ($\alpha \in [0, 1]$), donde valores más grandes de α permiten un tamaño más grande de la lista de candidatos. Si α vale 1, RCL contiene a todos los posibles candidatos, por lo que en este caso particular, el algoritmo sería completamente aleatorio. Del mismo modo, si α toma valor 0, RCL solo contiene al mejor de los candidatos, por lo que no habría aleatoriedad y

los algoritmos serían iguales a los propuestos en la Sección 4.3. Cuando todos los trabajos son asignados en esta fase, la solución generada es procesada en la búsqueda local.

4.4.2. Búsqueda local

Una vez implementada la aleatorización de la fase constructiva, se aplica la búsqueda local para intentar mejorar la solución obtenida. La búsqueda local propuesta consta de tres fases diferentes que se aplican en el siguiente orden:

1. Intercambio intra-máquinas
2. Intercambio entre-máquinas
3. Inserción entre-máquinas

Antes y después de cada una de las fases, la solución obtenida debe ser evaluada y reparada para que cumpla la restricción de recursos. Una vez la solución es factible, se compara el *makespan* actual con el de la solución antes de la búsqueda local. Si la solución es mejorada, se actualiza la mejor solución actual y se continua el proceso. En caso contrario, se desecha el cambio y se pasa a la siguiente fase de la búsqueda local. El proceso se repite hasta que una solución pase por las tres fases sin ninguna mejora. Es importante aclarar que al reparar una solución, ésta puede tener tiempos ociosos en las máquinas para cumplir con la restricción de recursos, por lo que antes de aplicar cada fase de la búsqueda local, es necesario eliminar estos tiempos ociosos justificando la solución hacia la izquierda con un procedimiento que llamamos "justificarIzq()". Al justificar la solución hacia la izquierda, ésta puede perder la factibilidad, y es por lo que la fase de reparación se debe repetir después de cada operación de la búsqueda local. La Figura 4.2 muestra un ejemplo de una solución antes y después de ser justificada a la izquierda. En esa figura podemos observar que, una vez justificada la solución, el consumo máximo de recursos aumenta, por lo que en este ejemplo la nueva solución podría no cumplir la restricción de recursos. El Algoritmo 6 muestra el pseudocódigo del procedimiento general de la búsqueda local propuesta.

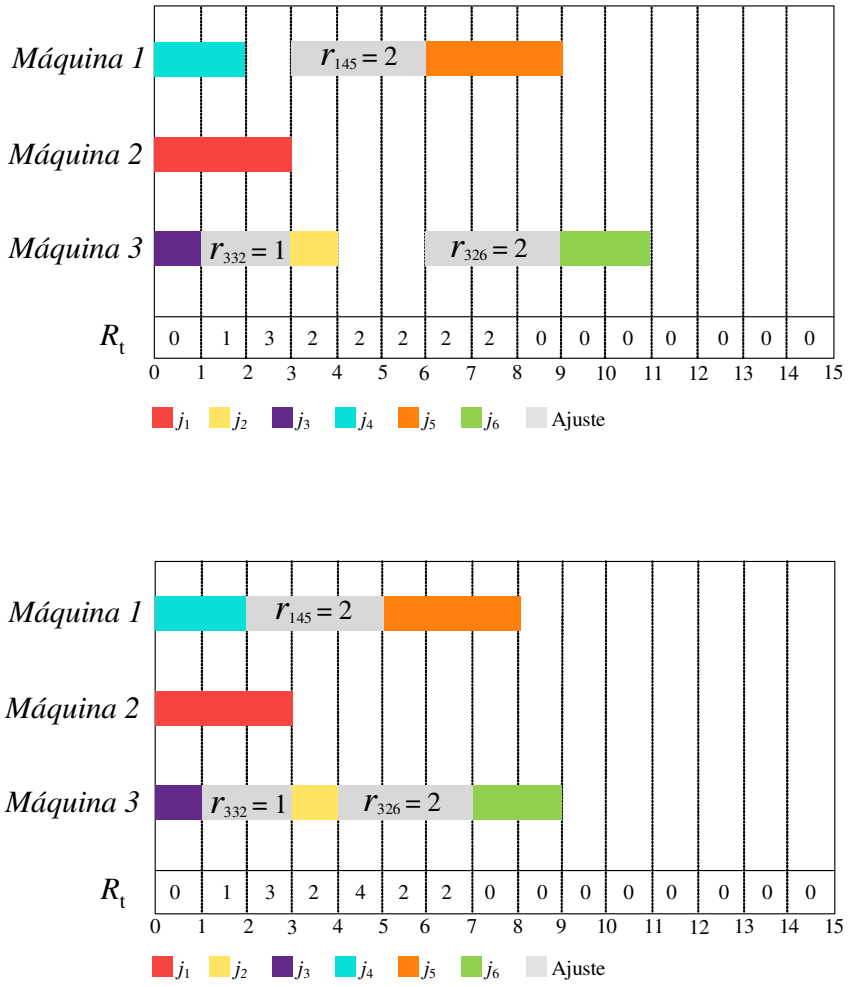

Figura 4.2: *Ejemplo de una solución justificada a la izquierda.*

Algoritmo 6: Procedimiento general de la búsqueda local.

```
1  Solución de trabajo ← Solución inicial
2  Solución de trabajo ← Aplicar Reparación
3  Mejor solución ← Solución de trabajo
4  CriterioParada ← False
5  mientras CriterioParada = False hacer
6  |  CriterioParada ← True
7  |  justificarIzq(Solución de trabajo)
8  |  Solución de trabajo ← Aplicar Intercambio intra-máquinas
9  |  Solución de trabajo ← Aplicar Reparación
10 |  si Solución de trabajo < Mejor solución entonces
11 |  |  Mejor solución ← Solución de trabajo
12 |  fin
13 |  justificarIzq(Solución de trabajo)
14 |  Solución de trabajo ← Aplicar Intercambio entre-máquinas
15 |  Solución de trabajo ← Aplicar Reparación
16 |  si Solución de trabajo < Mejor solución entonces
17 |  |  Mejor solución ← Solución de trabajo
18 |  |  CriterioParada ← False
19 |  fin
20 |  justificarIzq(Solución de trabajo)
21 |  Solución de trabajo ← Aplicar Inserción entre-máquinas
22 |  Solución de trabajo ← Aplicar Reparación
23 |  si Solución de trabajo < Mejor solución entonces
24 |  |  Mejor solución ← Solución de trabajo
25 |  |  CriterioParada ← False
26 |  fin
27 fin
```

Es importante mencionar que, para medir la posible mejora en cada una de las operaciones de la búsqueda local propuesta, se proponen valores que siguen la misma idea del valor λ propuesto para el constructivo 3 de la Sección 4.3.3 (tener en un solo valor información sobre tiempos de proceso, tiempos de ajuste y consumo de recursos). La decisión de usar estos valores la tomamos ya que, como se verá en los resultados computacionales de la Sección 4.6, el enfoque de construcción de soluciones que sigue la idea de tener en cuenta la información de recursos, mostró resultados mucho mejores a los del otro enfoque. En las tres operaciones propuestas en esta búsqueda local, los valores usados para medir la posible mejora penalizan los intercambios o las inserciones de trabajos que generen tiempos de ajuste o consumo de recursos altos. Así mismo, favorecen los cambios en la solución que generen mayores reducciones en tiempos de ajuste o consumo de recursos.

A continuación se explica en detalle cada fase de la búsqueda local.

Intercambio intra-máquinas

En este procedimiento se busca mejorar una solución intercambiando las posiciones de los trabajos en una misma máquina. Para cada máquina, la idea es tomar cada trabajo asignado en ésta, e intercambiar su posición con los demás trabajos de la misma máquina. Al realizar cada intercambio entre los trabajos j y k, se calcula un valor al que llamamos $S_{(j,k)}$ definido como:

$$S_{(j,k)} = (\gamma_{s(j,k)} \cdot \gamma_{r(j,k)}) - (\theta_{s(j,k)} \cdot \theta_{r(j,k)}),$$

donde:

- $\gamma_{s(j,k)}$ es la suma del tiempo de los ajustes que ya no se deben hacer al aplicar el intercambio de los trabajos j y k.
- $\gamma_{r(j,k)}$ es la suma del número de recursos que se necesitaban para realizar los ajustes que ya no se deben hacer al aplicar el intercambio de los trabajos j y k.

- $\theta_{s(j,k)}$ es la suma del tiempo de los nuevos ajustes que se deben realizar al aplicar el intercambio de los trabajos j y k.

- $\theta_{r(j,k)}$ es la suma del número de recursos necesarios para los nuevos ajustes que se deben realizar al aplicar el intercambio de los trabajos j y k.

Después de probar todos los intercambios en una máquina, se mantiene el intercambio con mayor valor de $S_{(j,k)}$ y se repite el proceso en la misma máquina con la nueva secuencia. Si un intercambio da valor de $S_{(j,k)}$ menor a cero, se entiende que la nueva secuencia es peor, ya que consume más recursos y/o tiene un ajuste más largo que en la secuencia antes del intercambio. Así mismo, un intercambio con valor $S_{(j,k)}$ igual a cero, significaría que el intercambio generaría los mismos consumos de tiempo de ajustes y de recursos, sin generar ningún beneficio. Por esto, si todos los intercambios probados en una máquina dan valores de $S_{(j,k)}$ menores o iguales a cero, no se hace ningún intercambio y se repite el proceso en la siguiente máquina. El Algoritmo 7 muestra el pseudocódigo con el procedimiento general del intercambio intra-máquinas.

Algoritmo 7: Intercambio intra-máquinas.

```
1  para cada i ∈ M hacer
2      Mejor intercambio ← 0
3      para cada j ∈ i hacer
4          para cada k ∈ i y k ≠ j hacer
5              Probar el intercambio entre j y k y calcular S_(j,k)
6              si S_(j,k) > Mejor intercambio entonces
7                  Mejor intercambio ← S_(j,k)
8              fin
9          fin
10     fin
11     si Mejor intercambio ≠ 0 entonces
12         Hacer Mejor intercambio;
13         CriterioParada ← False;
14         i = i − 1;
15     fin
16 fin
```

La operación de la línea 14 del Algoritmo 7 es necesaria ya que, si se realiza un intercambio, el proceso se repite en la misma máquina. Al restar 1 al contador i, cuando éste avance porque acabó el ciclo en el que estaba (línea 1), volverá a la máquina en la que se hizo el cambio en la iteración anterior.

Es importante mencionar que, al igual que con el valor λ_{ijk} de la fase constructiva, probamos diferentes maneras de calcular $S_{(j,k)}$. La forma de calcular que se explicó, sigue la misma estrategia multiplicativa del λ_{ijk} utilizado en la Heurística 4. La decisión de quedarnos con este valor fue tomada después de pruebas preliminares, y como se verá en la Sección 4.6, está sustentada al ver que la estrategia multiplicativa (Heurística 4) tiene mejores resultados que las otras dos estrategias planteadas. Esto es igual con los valores a calcular en las otras dos operaciones de la búsqueda local que se explicarán a continuación.

Intercambio entre-máquinas

Para explicar el procedimiento del intercambio entre-máquinas, definimos i' como la máquina que proporciona el *makespan*, es decir, la máquina que termina más tarde en la solución actual. En esta operación, se prueba el intercambio de cada trabajo j asignado en i' con cada trabajo k de las otras máquinas $i \neq i'$. Al realizar cada intercambio entre los trabajos $j \in i'$ y $k \in i$, se calcula un valor al que llamamos $S_{(j,k)}$ definido como:

$$S_{(j,k)} = (\rho_{(j,k)} + \gamma_{s(j,k)} \cdot \gamma_{r(j,k)}) - (\phi_{(j,k)} + \theta_{s(j,k)} \cdot \theta_{r(j,k)}),$$

donde:

- $\rho_{(j,k)}$ es la suma de los tiempos de proceso de los trabajos $j \in i'$ y $k \in i$ en la secuencia original.
- $\phi_{(j,k)}$ es la suma de los tiempos de proceso de los trabajos j y k después de hacer el intercambio.
- $\gamma_{s(j,k)}$ es la suma de los tiempos de los ajustes (en i' y en i) que no deben realizarse al hacer el intercambio entre los trabajos j y k.

- $\gamma_{r(j,k)}$ es la suma del número de de recursos necesarios en los ajustes que no deben realizarse al hacer el intercambio entre los trabajos j y k.

- $\theta_{s(j,k)}$ es la suma de los tiempos de los nuevos ajustes que deben realizarse (en i' y en i) cuando se hace el intercambio entre los trabajos j y k.

- $\theta_{r(j,k)}$ es la suma del número de de recursos necesarios en los nuevos ajustes que deben realizarse (en las dos máquinas) cuando se hace el intercambio entre los trabajos j y k.

Después de probar todos los intercambios posibles de cada trabajo de la máquina i', se deja el intercambio con mayor valor $S_{(j,k)}$ y se vuelve a calcular i' para repetir el proceso. El proceso termina cuando, después de probar todos los intercambios, ninguno de ellos da valores de $S_{(j,k)}$ mayores a cero, ya que, igual que en el intercambio intra-máquinas, un valor de $S_{(j,k)}$ menor a cero, mostraría que la nueva secuencia es peor que la anterior. El Algoritmo 8 muestra el pseudocódigo con el procedimiento general del intercambio entre-máquinas.

Inserción entre-máquinas

La última operación que se aplica a la solución, consiste en probar la inserción de cada trabajo de la máquina que da el *makespan* (i') en cada una de las posiciones de las otras máquinas. En cada inserción se calcula un valor que llamamos $S_{(j,k)}$ definido como:

$$S_{(j,k)} = (C_{max} + \gamma_{s(j,k)} \cdot \gamma_{r(j,k)}) - (C^*_{(i,j,k)} + \theta_{s(j,k)} \cdot \theta_{r(j,k)}),$$

donde:

- $C_{\text{máx}}$ es el *makespan* en la secuencia original antes de la inserción.

- $C^*_{(i,j,k)}$ es el tiempo de finalización de la máquina i después de la inserción del trabajo j (anteriormente asignado en i') en la posición k.

Algoritmo 8: Intercambio entre-máquinas.

```
1  CriterioParada ← False
2  mientras CriterioParada = False hacer
3  |  CriterioParada ← True
4  |  Mejor intercambio ← 0
5  |  i' ← Máquina makespan
6  |  para cada j ∈ i' hacer
7  |  |  para cada i ∈ M \ {i'} hacer
8  |  |  |  para cada k ∈ i hacer
9  |  |  |  |  Probar el intercambio entre j ∈ i' y k ∈ i y calcular S_(j,k)
10 |  |  |  |  si S_(j,k) > Mejor intercambio entonces
11 |  |  |  |  |  Mejor intercambio ← S_(j,k)
12 |  |  |  |  fin
13 |  |  |  fin
14 |  |  fin
15 |  fin
16 |  si Mejor intercambio ≠ 0 entonces
17 |  |  Hacer Mejor intercambio;
18 |  |  Calcular i';
19 |  |  CriterioParada ← False;
20 |  fin
21 fin
```

- $\gamma_{s(i,j,k)}$ es la suma de los tiempos de los ajustes (en i' y en i) que no se deben hacer después de realizar la inserción del trabajo j (anteriormente asignado en i') en la posición k de la máquina i.

- $\gamma_{r(i,j,k)}$ es la suma del número de de recursos necesarios en los ajustes que no deben realizarse después de hacer la inserción del trabajo j (anteriormente asignado en i') en la posición k de la máquina i.

- $\theta_{s(i,j,k)}$ es la suma de los tiempos de los nuevos ajustes que deben realizarse (en i' y en i) después de hacer la inserción del trabajo j (anteriormente asignado en i') en la posición k de la máquina i.

- $\theta_{r(i,j,k)}$ es la suma del número de de recursos necesarios en los nuevos ajustes que deben realizarse (en las dos máquinas) después de hacer la inserción del trabajo j (anteriormente asignado en i') en la posición k de la máquina i.

Después de probar la inserción de cada uno de los trabajos en i', se deja la inserción con mayor valor $S_{(j,k)}$. Al igual que en las operaciones anteriores, si ninguna inserción genera $S_{(j,k)}$ mayor a cero, no se realiza ningún movimiento y se finaliza la operación. El Algoritmo 9 muestra el pseudocódigo con el procedimiento general de la inserción entre-máquinas.

Aplicando la aleatorización y la búsqueda local a los algoritmos heurísticos explicados en la Sección 4.3, obtenemos las siguientes metaheurísticas:

- GRASP 1: Heurística 1 + Aleatorización + Búsqueda local.

- GRASP 2: Heurística 2 + Aleatorización + Búsqueda local.

- GRASP 3: Heurística 3 + Aleatorización + Búsqueda local.

- GRASP 4: Heurística 4 + Aleatorización + Búsqueda local.

- GRASP 5: Heurística 5 + Aleatorización + Búsqueda local.

Algoritmo 9: Inserción entre-máquinas.

```
1  Mejor Inserción ← 0
2  i' ← Máquina makespan
3  para cada j ∈ i' hacer
4  |  para cada i ∈ M \ {i'} hacer
5  |  |  para cada k ∈ i hacer
6  |  |  |  Probar la inserción del trabajo j, en la posición k de la máquina i, y
   |  |  |   calcular S_(i,j,k)
7  |  |  |  si S_(i,j,k) > Mejor Inserción entonces
8  |  |  |  |  Mejor Inserción ← S_(i,j,k)
9  |  |  |  fin
10 |  |  fin
11 |  fin
12 fin
13 si Mejor inserción ≠ 0 entonces
14 |  Hacer Mejor inserción;
15 fin
```

4.5. Tercer enfoque de resolución

Como se mencionó en la Sección 4.3.1, además de los dos enfoques de resolución explicados a lo largo de este capítulo, se probó un tercer enfoque en el que se busca generar soluciones factibles durante la fase constructiva, eliminando la fase de reparación al no ser necesaria.

El algoritmo propuesto con este enfoque consiste en evaluar el consumo de recursos de toda la secuencia en cada posible asignación de un trabajo. Inicialmente se crea el conjunto de trabajos pendientes de asignación N^* y se organiza en orden no creciente de acuerdo al tiempo medio de procesamiento en todas las máquinas ($\bar{p_j} = \sum_{i \in M} p_{ij}/m$). Una vez ordenado el conjunto, se toma el primer elemento y se prueba su asignación en la siguiente posición disponible de cada máquina. Por ejemplo, si en una máquina i hay 2 trabajos j y k previamente asignados, se probará la asignación del nuevo trabajo en la tercera posición de esa máquina, después de la finalización del trabajo k. Se prueba la asignación del trabajo elegido en todas las máquinas, iniciando el ajuste necesario en la máquina inmediatamente

después de la finalización del último trabajo secuenciado en esta. Si el inicio de ese ajuste genera que la restricción de recursos no se cumpla, se prueba iniciando el ajuste un instante de tiempo después. Esto se repite hasta que la asignación del trabajo cumple con la restricción de recursos y la solución parcial es factible. Al final de cada iteración, se asigna el trabajo a la máquina en la que el tiempo de finalización sea menor.

Este enfoque de resolución resultó ser poco eficiente computacionalmente, ya que requiere evaluar constantemente el consumo de recursos en cada instante de tiempo. Debido a su poca eficiencia, hace imposible aplicar métodos de aleatorización en la fase constructiva, ya que se necesitaría evaluar más asignaciones, lo cual requeriría mucho tiempo de cómputo. Además, como se verá en la sección de resultados, las soluciones obtenidas por este algoritmo son mucho peores que las de los algoritmos heurísticos propuestos. Este enfoque fue desechado rápidamente al no mostrar resultados iniciales prometedores en la resolución del problema.

4.6. Resultados computacionales

Para evaluar el rendimiento de los algoritmos propuestos, en esta sección se presenta una comparativa de los resultados obtenidos en un grupo de instancias generadas aleatoriamente. El grupo de instancias utilizado está basado en las instancias propuestas en Vallada y Ruiz (2011), utilizadas para el problema de secuenciación de máquinas paralelas no relacionadas con necesidad de ajustes sin recursos adicionales (UPMS). A las instancias originales se les agregó la información sobre el consumo de recursos (r_{ijk}) y el número máximo de recursos disponibles ($R_{\text{máx}}$). Las instancias se dividen en dos grupos: uno de instancias pequeñas, y otro de instancias grandes.

En las instancias originales, el grupo de instancias pequeñas cuenta con 640 instancias con:

- $n \in \{6, 8, 10, 12\}$.
- $m \in \{2, 3, 4, 5\}$.

El grupo de instancias grandes cuenta con 1000 instancias con:

- $n \in \{50, 100, 150, 200, 250\}$.
- $m \in \{10, 15, 20, 25, 30\}$.

Para los dos grupos, los tiempos de ajuste (s_{ijk}) son generados aleatoriamente siguiendo una distribución uniforme discreta en los conjuntos $\{1, \ldots, 9\}$, $\{1, \ldots, 49\}$, $\{1, \ldots, 99\}$ y $\{1, \ldots, 124\}$. Los tiempos de proceso (p_{ij}) son generados aleatoriamente siguiendo una distribución uniforme discreta en el conjunto $\{1, \ldots, 99\}$. Combinando los diferentes valores de n, de m y de s_{ijk}, obtenemos $4 \times 4 \times 4 = 64$ configuraciones en las instancias pequeñas, y $5 \times 5 \times 4 = 100$ configuraciones para las instancias grandes. Cada posible configuración es replicada 10 veces generando 640 instancias pequeñas y 1000 instancias grandes.

Por último, a estas instancias originales de Vallada y Ruiz (2011), se les agrega el consumo de recursos y el número máximo de recursos disponibles. En el grupo de instancias pequeñas, el número máximo de recursos disponibles ($R_{\text{máx}}$) se generó aleatoriamente siguiendo una distribución uniforme discreta en el conjunto $\{1, 2\}$, mientras que en el grupo de instancias grandes fue generado aleatoriamente siguiendo una distribución uniforme discreta en el conjunto $\{3, 4\}$. Así mismo, los consumos de recursos (r_{ijk}) fueron generados para cada instancia aleatoriamente, siguiendo una distribución uniforme en el conjunto $\{1, \ldots, R_{\text{máx}}\}$.

Para comparar los resultados de los algoritmos, para cada uno de ellos se calcula la desviación porcentual relativa o *Relative Percentage Deviation* (RPD), definida como:

$$RPD(alg) = \frac{C_{\text{máx}}(alg) - C_{\text{máx}}(best)}{C_{\text{máx}}(best)} \cdot 100,$$

donde $C_{\text{máx}}(alg)$ es el *makespan* de la solución obtenida por el algoritmo ejecutado, y $C_{\text{máx}}(best)$ es el *makespan* de la mejor solución encontrada (por cualquier algoritmo) para esa instancia.

Todos los algoritmos propuestos en este capítulo, fueron codificados en C# bajo el mismo .NET framework en Visual Studio 2013. Los algoritmos se ejecutaron en un ordenador con

sistema operativo Windows 10 Enterprise de 64 Bits, con procesador Intel core i7 de 2.60GHz y 8GB de memoria RAM.

4.6.1. Resultados de los algoritmos heurísticos comparados con soluciones encontradas por el modelo de programación lineal entera mixta

Como se adelantó en la Sección 4.2, el modelo de programación lineal entera mixta ($MILP$) solo resuelve óptimamente instancias muy pequeñas. Por ello, el modelo solo fue utilizado para resolver las instancias con $n = 6$. Para implementar el modelo se utilizó CPLEX 12.6. Cada instancia fue ejecutada con un tiempo máximo de 3600 segundos, tiempo en el que el solver utilizado encontró 100 soluciones óptimas en un total de 160 instancias.

Resultados en instancias resueltas con optimalidad

La Tabla 4.1, muestra la comparativa de los cinco algoritmos heurísticos propuestos en la Sección 4.3 con las soluciones óptimas encontradas por el modelo de programación lineal entera mixta. En esa tabla solo se consideran las instancias en las que el $MILP$ probó optimalidad.

		Primer Enfoque		Segundo Enfoque			
		Heurística 1	Heurística 2	Heurística 3	Heurística 4	Heurística 5	Modelo $MILP$
Tamaño	#	RPD	RPD	RPD	RPD	RPD	Av. $t(s)$
6x2	16	13.43	16.09	13.21	12.56	14.43	1881.90
6x3	24	23.46	20.45	26.94	24.40	27.52	1069.10
6x4	27	28.02	14.61	29.61	29.51	32.61	716.61
6x5	33	27.61	8.00	29.26	28.47	29.98	1093.03
Prom.		23.13	14.79	24.76	23.74	26.13	1190.16

Tabla 4.1: *RPD promedio para los algoritmos heurísticos con respecto a las soluciones óptimas.*

La Tabla 4.1 muestra el RPD promedio entre el *makespan* de las soluciones encontradas por los diferentes heurísticos y el *makespan* óptimo (garantizado). En esta tabla, la columna llamada "#" muestra el número de instancias en las que el modelo de programación lineal entera mixta encontró la solución óptima en cada tamaño de instancia $n \times m$. La columna

("Av. $t(s)$") muestra el tiempo medio de ejecución del $MILP$ en segundos. No se vieron diferencias significativas en los tiempos de ejecución de los cinco algoritmos heurísticos, estando los tiempos entre 4.81 y 5.5 milisegundos. En la tabla se observa que ningún algoritmo está muy cerca de las soluciones óptimas. En promedio, el algoritmo que está más cerca de las soluciones óptimas es la Heurística 2 a un 14.79 %, mientras que no se ven grandes diferencias en el rendimiento de los otros algoritmos.

Para comprobar si las diferencias vistas entre los algoritmos son estadísticamente significativas, se hace un análisis de varianza $ANOVA$ (Montgomery, 2006). La variable respuesta en el $ANOVA$ es el RPD y el único factor es el tipo de algoritmo con cinco niveles (Heurística 1, Heurística 2, Heurística 3, Heurística 4 y Heurística 5). La Figura 4.3 muestra los intervalos HSD de Tukey al 95 % de confianza para el factor estudiado. En esta figura, si dos intervalos no se solapan, quiere decir que existen diferencias estadísticamente significativas entre los dos grupos. Podemos observar que, a pesar de tener mejores resultados con la Heurística 2, no hay diferencias estadísticamente significativas entre los algoritmos heurísticos propuestos.

Resultados en instancias con factibilidad sin optimalidad

La Tabla 4.2 muestra el GAP promedio entre los métodos de resolución y la cota inferior dada por el modelo matemático en las instancias en las que el $MILP$ encontró soluciones factibles, pero no garantiza optimalidad. En la tabla podemos observar que, al igual que con las instancias en las que se encontraron soluciones óptimas, la Heurística 2 es la que mejor rendimiento tiene, con un GAP del 13.66 %. También observamos que los algoritmos que siguen el primer enfoque rinden mejor que los del segundo enfoque, mientras que el GAP promedio del modelo matemático es mucho más alto que el de los cinco algoritmos propuestos.

La Figura 4.4 muestra los intervalos HSD de Tukey al 95 % de confianza del GAP promedio entre los algoritmos heurísticos y la cota inferior dada por el modelo matemático. En la figura podemos observar que, al igual que sucede en las instancias en las que se

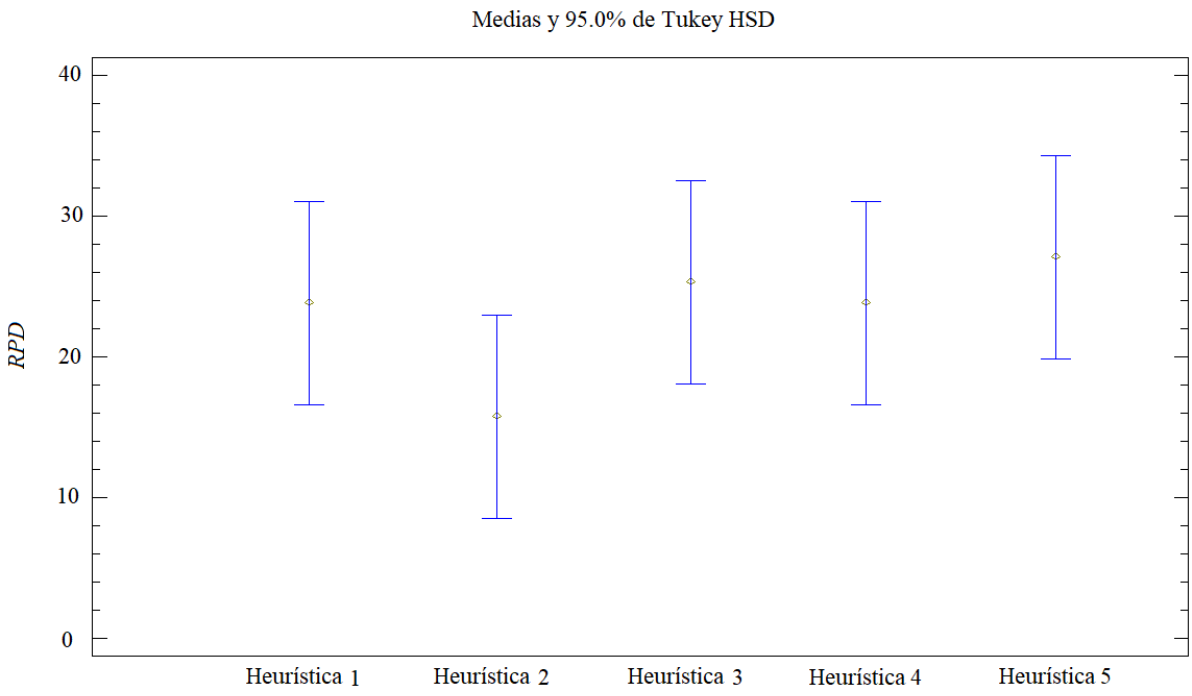

Figura 4.3: *Intervalos HSD de Tukey del RPD promedio para los algoritmos heurísticos en las instancias resueltas con optimalidad por el MILP.*

		Primer Enfoque		Segundo Enfoque			
Tamaño	#	Heurística 1	Heurística 2	Heurística 3	Heurística 4	Heurística 5	Modelo $MILP$
6x2	23	8.49	14.18	12.23	10.69	11.12	59.82
6x3	15	21.10	14.50	25.87	26.23	28.33	55.71
6x4	9	23.02	16.20	29.45	27.00	31.15	90.05
6x5	5	22.53	4.13	25.12	22.53	22.39	99.68
Prom.		15.99	13.66	20.38	19.13	20.63	76.31

Tabla 4.2: *GAP promedio para los algoritmos heurísticos con respecto a la cota inferior dada por el modelo $MILP$ en las instancias donde el MILP no probó optimalidad.*

encontraron soluciones óptimas, no hay diferencias significativas entre los cinco algoritmos heurísticos. Sin embargo, las soluciones de todos los algoritmos heurísticos propuestos son significativamente mejores que las del modelo matemático en este grupo de instancias.

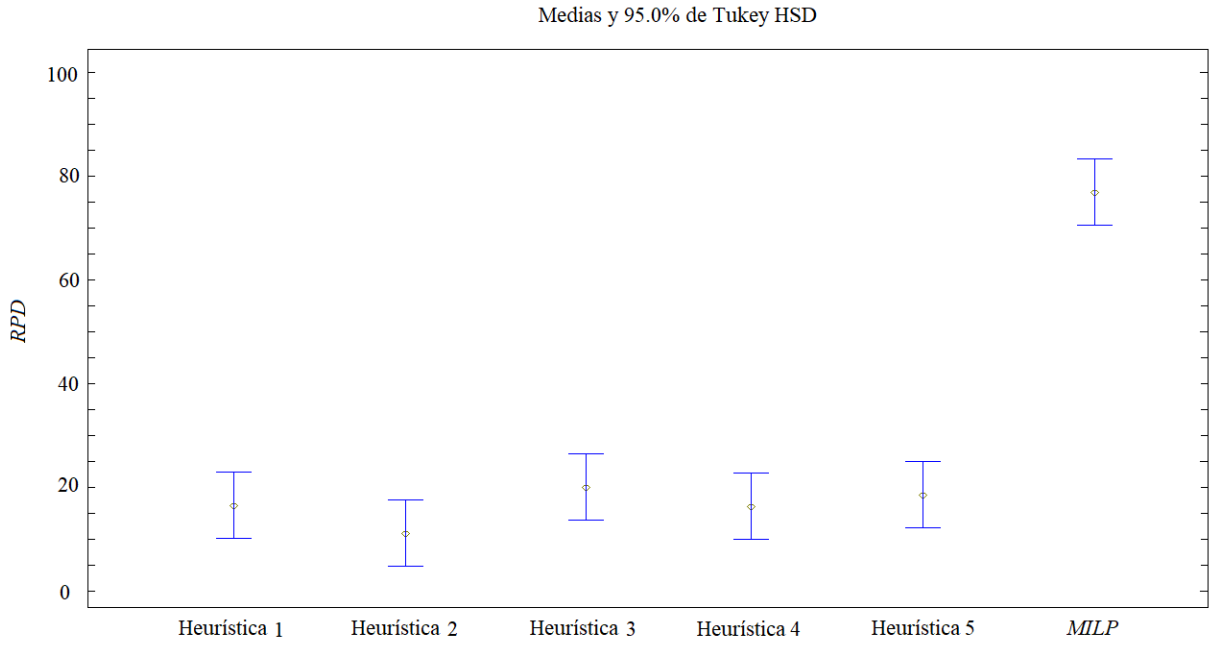

Figura 4.4: *Intervalos HSD de Tukey para el GAP promedio de los algoritmos heurísticos con respecto a la cota inferior dada por el modelo MILP en las instancias donde el MILP no probó optimalidad.*

Resultados en todas las instancias con factibilidad

La Tabla 4.3 muestra el GAP promedio para los cinco algoritmos heurísticos propuestos y las cotas inferiores dadas por el modelo matemático en todas las instancias con $n = 6$, excepto en las 8 instancias en las que el modelo no fue capaz de encontrar ninguna solución factible después de los 3600 segundos de ejecución. La columna "$t(s)$ Mod" muestra el tiempo promedio en segundos en el que se ejecutó el modelo matemático. En la tabla podemos observar que la Heurística 2 tiene mejor rendimiento promedio que los demás métodos, incluyendo el modelo de programación lineal entera mixta. También podemos ver que no hay mucha diferencia entre el GAP del modelo con respecto al GAP de los algoritmos del

segundo enfoque, teniendo un rendimiento ligeramente mejor la Heurística 4.

		Primer Enfoque		Segundo Enfoque				
Tamaño	#	Heur 1	Heur 2	Heur 3	Heur 4	Heur 5	Modelo $MILP$	$t(s)$ Mod
6x2	39	10.31	11.83	14.18	10.92	12.23	35.28	2516.55
6x3	39	16.86	14.14	24.77	18.12	29.41	21.43	1914.53
6x4	36	21.98	8.84	31.58	23.48	33.11	22.51	1448.80
6x5	38	23.81	7.35	26.22	24.61	23.12	13.12	1123.72
Prom.		18.13	10.59	24.18	19.16	24.46	23.16	1750.90

Tabla 4.3: *GAP promedio para los algoritmos heurísticos con respecto a la cota inferior dada por el modelo MILP en las instancias en las que el MILP encontró solución factible.*

La Figura 4.5 muestra los intervalos HSD de Tukey al 95 % de confianza del *GAP* promedio para los cinco algoritmos heurísticos propuestos y las cotas inferiores dadas por el modelo matemático en este conjunto de instancias. Podemos observar que los resultados de la Heurística 2 son significativamente mejores que los del modelo matemático y que los de las Heurísticas 3 y 5. Estos resultados nos muestran que, si tenemos en cuenta el tiempo de ejecución, que en el caso de las heurísticas es de pocos milisegundos, mientras que en el modelo llega a ser en promedio de casi 30 minutos, la alternativa de utilizar algoritmos heurísticos tendría gran utilidad.

4.6.2. Resultados de los algoritmos heurísticos en el total de las instancias

Para hacer una comparación completa de los algoritmos heurísticos, continuamos la comparación en todo el conjunto de instancias, distinguiendo entre el grupo de pequeñas y de grandes.

Comparación de los algoritmos heurísticos en el grupo de instancias pequeñas

La Tabla 4.4 muestra el *RPD* promedio para cada tamaño de instancias y los tiempos de cómputo de los cinco algoritmos heurísticos en las 640 instancias pequeñas. Cada tamaño de instancias $n \times m$ tiene 40 instancias. En esa tabla podemos ver que, al igual que en las instancias resueltas óptimamente, la Heurística 2 es la que encuentra mejores soluciones en

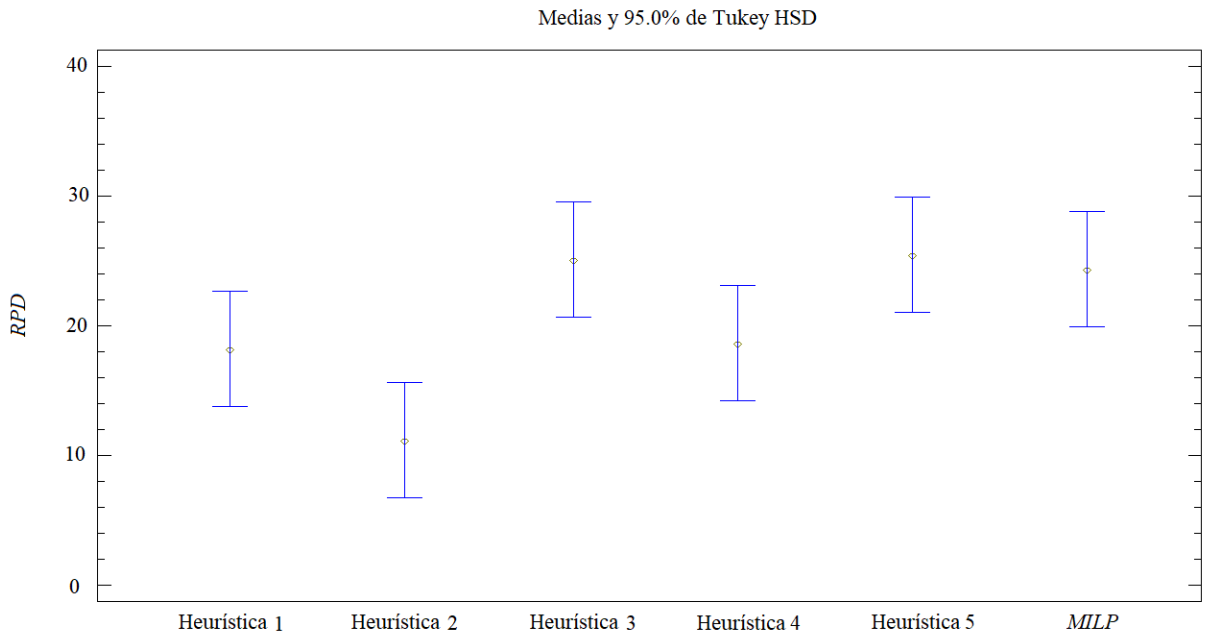

Figura 4.5: *Intervalos HSD de Tukey para el GAP promedio para los algoritmos heurísticos con respecto a la cota inferior dada por el modelo MILP en las instancias en las que el MILP encontró solución factible.*

promedio. Sin embargo, no se observan grandes diferencias en el RPD promedio entre las heurísticas propuestas. Por otro lado, las heurísticas que siguen el primer enfoque parecen ser ligeramente más rápidas que las heurísticas que siguen el segundo enfoque.

La Figura 4.6 muestra los intervalos HSD de Tukey al 95 % de confianza del RPD promedio de los diferentes algoritmos. En la figura podemos observar que, a pesar de tener mejores resultados promedio con la Heurística 2, no hay diferencias estadísticamente significativas entre todas las heurísticas propuestas.

Comparación de los algoritmos heurísticos en el grupo de instancias grandes

A continuación, probamos el rendimiento de los algoritmos heurísticos en el grupo de instancias grandes. La Tabla 4.5 muestra el RPD promedio de los cinco algoritmos heurísticos en las 1000 instancias grandes. Al igual que en el grupo de instancias pequeñas, cada tamaño de instancias $n \times m$ tiene 40 instancias. En la tabla podemos observar que, al contrario que

	Primer Enfoque				Segundo Enfoque					
	Heurística 1		Heurística 2		Heurística 3		Heurística 4		Heurística 5	
Tamaño	RPD	$t(ms)$	RPD	$t(ms)$	RPD	$t(ms)$	RPD	$t(ms)$	RPD	$t(ms)$
6x2	8.06	5.32	9.58	5.41	9.11	5.68	8.67	6.01	10.17	5.75
6x3	12.67	5.35	8.57	5.49	14.47	6.47	13.85	7.01	14.83	6.69
6x4	18.86	5.21	6.07	5.21	20.60	7.98	20.20	7.13	21.97	6.80
6x5	18.30	5.23	1.46	4.45	19.45	7.06	19.03	6.91	20.17	7.50
8x2	3.13	5.42	9.32	5.53	5.49	8.01	3.96	7.13	4.01	7.21
8x3	9.56	6.3	11.71	5.62	10.28	7.96	9.75	7.34	10.85	7.83
8x4	17.03	6.32	10.08	5.74	15.98	7.95	15.47	7.41	16.19	8.28
8x5	16.05	5.92	11.04	5.69	19.99	6.87	19.62	7.03	20.79	7.78
10x2	5.26	6.52	7.42	6.58	5.50	7.54	5.40	7.29	6.02	6.94
10x3	8.45	6.6	5.85	6.9	8.76	7.46	8.29	7.35	8.80	6.56
10x4	11.31	6.65	12.02	6.88	15.15	7.90	12.94	7.37	12.98	7.95
10x5	13.34	5.98	8.92	5.59	13.68	7.60	13.28	7.01	16.24	7.96
12x2	5.18	9.01	8.93	9.45	5.45	9.37	3.58	8.56	4.10	8.07
12x3	8.03	8.89	10.88	9.23	8.09	9.40	6.51	9.12	7.36	9.49
12x4	8.03	7.9	13.85	8.53	7.44	10.14	6.93	9.23	7.70	9.37
12x5	12.88	7.84	18.98	7.86	10.58	8.46	9.74	8.03	11.71	8.70
Prom.	11.01	6.53	9.67	6.51	11.88	7.87	11.07	7.49	12.12	7.68

Tabla 4.4: *RPD y tiempos de cómputo para los algoritmos heurísticos en el grupo de instancias pequeñas.*

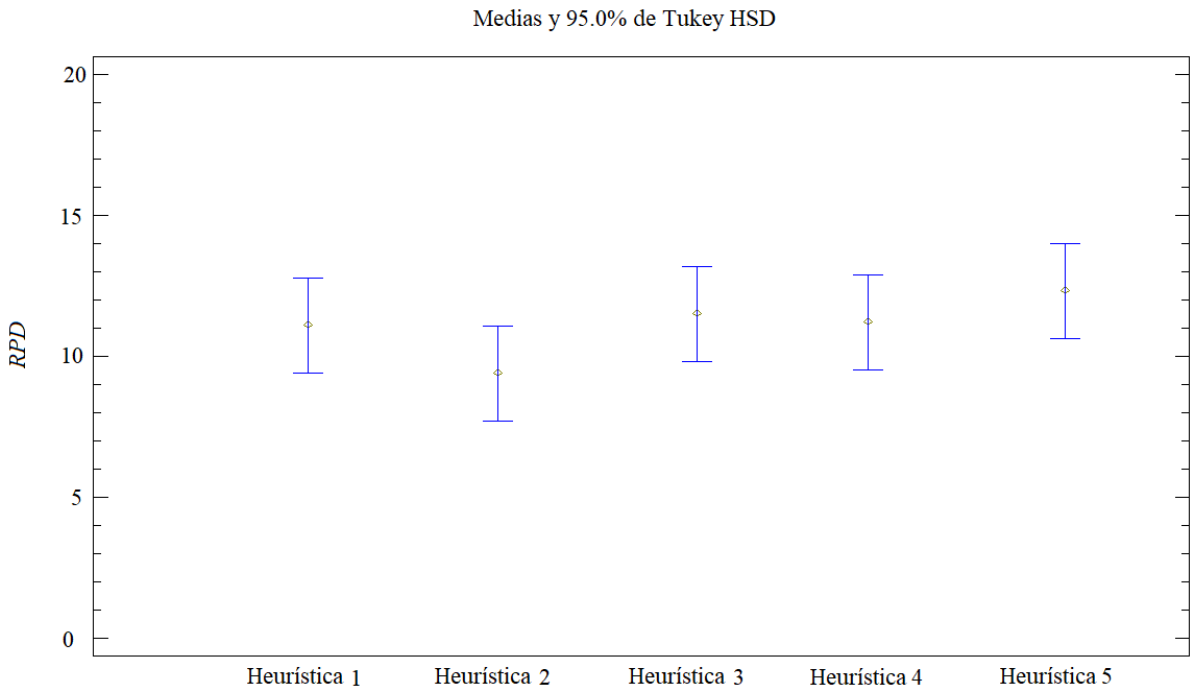

Figura 4.6: *Intervalos HSD de Tukey para los algoritmos heurísticos en las instancias pequeñas.*

en las instancias pequeñas, en este grupo sí vemos gran diferencia entre los dos enfoques de resolución del problema. Los algoritmos del segundo enfoque (aquel que tiene en cuenta la información de los recursos durante la fase constructiva) tienen un rendimiento mucho mejor que los algoritmos del primer enfoque. Mientras que la Heurística 4 tiene los mejores resultados con un RPD promedio de 0.19 %, las heurísticas del primer enfoque tienen un RPD promedio cercano al 40 % y al 70 %, respectivamente.

Al igual que con las instancias pequeñas, realizamos un $ANOVA$ para comprobar si las diferencias vistas en la Tabla 4.5 son significativas. La Figura 4.7 muestra los intervalos HSD de Tukey al 95 % de confianza para los diferentes algoritmos. Observando el gráfico, podemos confirmar que las diferencias mostradas entre los dos enfoques son significativas.

La Figura 4.8 muestra un acercamiento de la Figura 4.7 donde solo se muestran solo las heurísticas 3, 4 y 5. En la figura podemos observar que las diferencias entre la Heurística 4 y las otras dos, son estadísticamente significativas, confirmando que el mejor algoritmo para

	Primer Enfoque				Segundo Enfoque					
	Heurística 1		Heurística 2		Heurística 3		Heurística 4		Heurística 5	
Tamaño	RPD	$t(s)$	RPD	$t(s)$	RPD	$t(s)$	RPD	$t(s)$	RPD	$t(s)$
50x10	39.84	0.010	49.11	0.009	6.55	0.088	2.39	0.014	5.64	0.134
50x15	38.51	0.009	57.37	0.007	5.12	0.067	0.19	0.010	3.76	0.113
50x20	38.52	0.012	73.44	0.010	4.82	0.026	0.12	0.021	3.42	0.112
50x25	38.98	0.013	67.99	0.008	4.70	0.078	0.52	0.023	4.24	0.080
50x30	39.30	0.018	55.02	0.009	5.51	0.059	0.58	0.020	4.19	0.142
100x10	37.53	0.042	42.85	0.012	5.57	0.092	0.78	0.045	4.62	0.101
100x15	38.08	0.039	65.05	0.014	4.92	0.114	0.12	0.047	3.80	0.141
100x20	37.43	0.041	75.89	0.020	4.76	0.066	0.00	0.052	3.66	0.113
100x25	38.15	0.040	74.99	0.019	4.58	0.112	0.00	0.049	3.62	0.187
100x30	37.72	0.043	80.11	0.020	4.34	0.102	0.00	0.050	3.08	0.131
150x10	37.37	0.081	57.64	0.070	4.41	0.138	0.01	0.091	3.69	0.174
150x15	37.54	0.080	68.98	0.075	4.56	0.098	0.00	0.089	3.57	0.162
150x20	37.87	0.078	76.29	0.071	4.97	0.156	0.00	0.084	3.66	0.195
150x25	38.35	0.083	75.82	0.070	4.92	0.138	0.00	0.094	3.50	0.237
150x30	38.21	0.089	78.88	0.078	4.08	0.177	0.00	0.124	3.53	0.277
200x10	39.07	0.120	53.04	0.090	4.96	0.405	0.07	0.353	3.63	0.477
200x15	40.44	0.140	70.78	0.099	4.42	0.447	0.00	0.362	3.38	0.490
200x20	41.20	0.138	74.14	0.109	4.00	0.435	0.00	0.342	3.70	0.497
200x25	42.18	0.233	81.30	0.098	4.17	0.520	0.00	0.456	3.48	0.587
200x30	42.23	0.288	85.32	0.094	4.90	0.547	0.00	0.488	3.35	0.615
250x10	42.95	0.399	59.09	0.204	4.84	0.559	0.03	0.501	3.25	0.579
250x15	43.48	0.322	73.25	0.284	4.65	0.609	0.00	0.531	3.02	0.701
250x20	44.27	0.343	80.12	0.293	4.06	0.587	0.00	0.553	3.70	0.677
250x25	44.34	0.464	82.01	0.286	4.10	0.616	0.00	0.609	3.83	0.687
250x30	45.61	0.589	83.12	0.400	4.75	0.732	0.00	0.700	3.61	0.780
Prom.	39.97	0.148	69.66	0.098	4.75	0.279	0.19	0.228	3.72	0.336

Tabla 4.5: *RPD y tiempos de cómputo para los algoritmos heurísticos en el grupo de instancias grandes.*

este grupo de instancias es la Heurística 4.

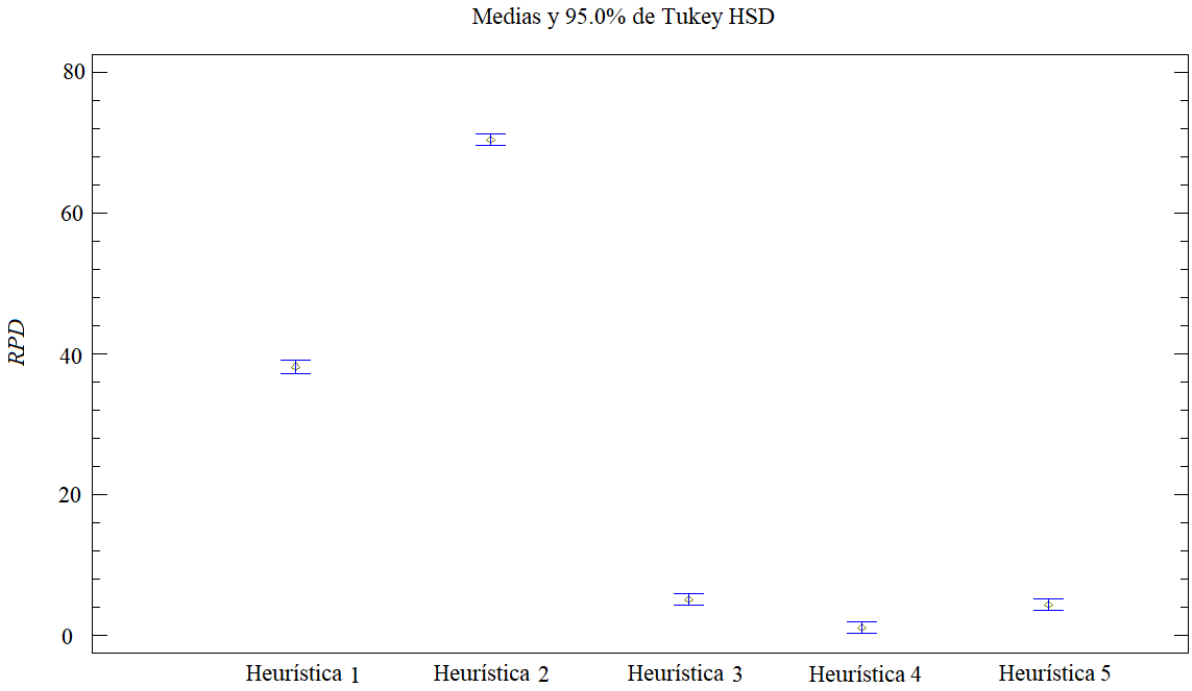

Figura 4.7: *Intervalos HSD de Tukey para los algoritmos heurísticos en las instancias grandes.*

Estos resultados nos muestran la importancia de no ignorar la información del consumo de recursos mientras se construyen las soluciones. Esto puede ser debido a que construir soluciones con menor consumo de recursos, hace más sencillo el proceso de reparación, teniendo que aplazar el inicio de menos ajustes y, en consecuencia, obteniendo soluciones con menos tiempos ociosos en las máquinas y menor *makespan*. Sin embargo, viendo la desviación promedio en términos de calidad del *makespan* entre los algoritmos heurísticos propuestos y las soluciones óptimas en instancias pequeñas, nos damos cuenta de la importancia de mejorar los algoritmos heurísticos, buscando reducir esas diferencias. Tal mejora se obtiene con los algoritmos GRASP introducidos en la Sección 4.4, tal y como se verá en los siguientes experimentos.

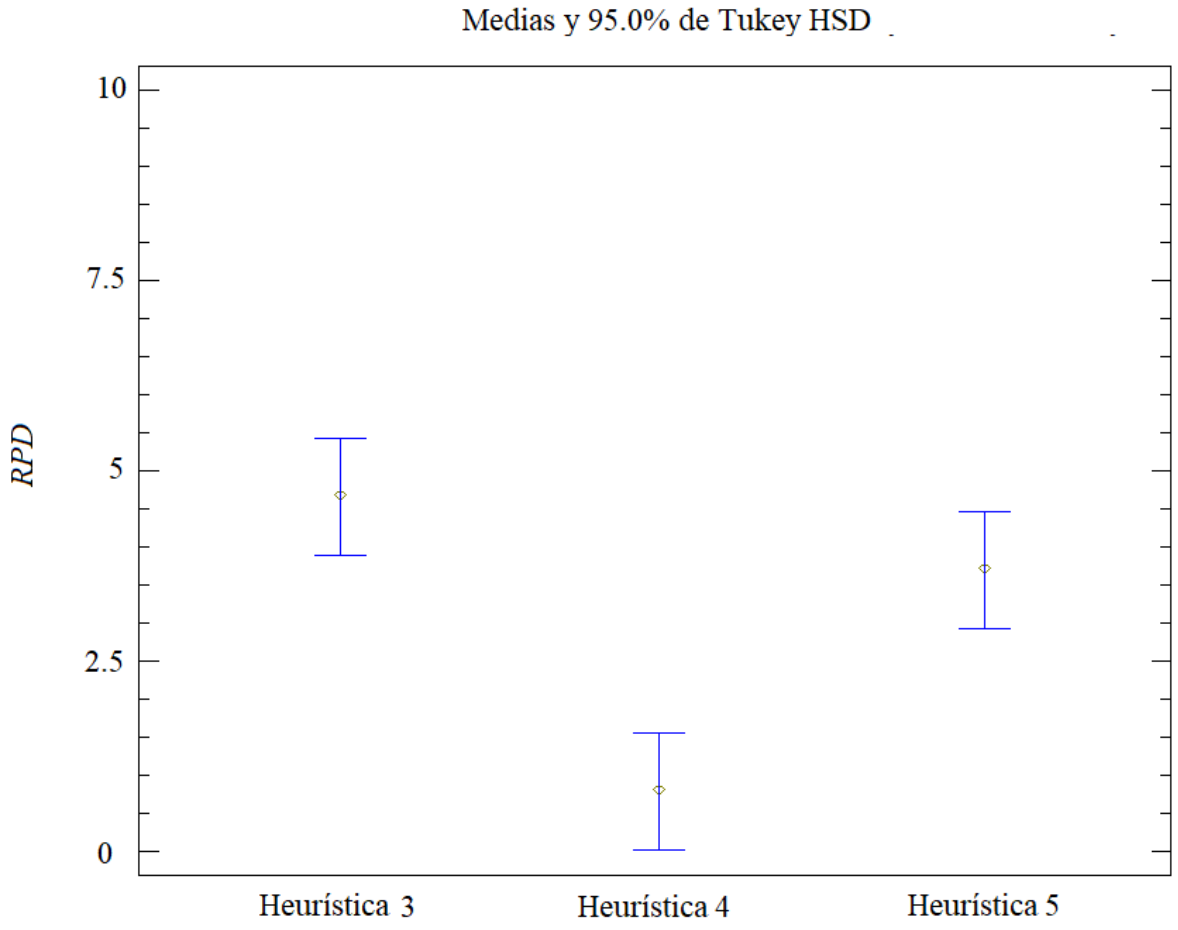

Figura 4.8: *Acercamiento de los intervalos HSD de Tukey para los algoritmos heurísticos en las instancias grandes.*

4.6.3. Resultados de los algoritmos metaheurísticos en las instancias resueltas por el $MILP$

Los algoritmos GRASP explicados en la Sección 4.4 fueron probados con tres valores diferentes de α (0.25, 0.5 y 0.75) buscando diferentes niveles de aleatoriedad en su ejecución. Después de ejecutar todos los algoritmos GRASP con cada uno de los valores de α en todas las instancias, el mejor valor para cada algoritmo fue:

- GRASP 1: $\alpha = 0.75$ en instancias pequeñas y $\alpha = 0.25$ en instancias grandes.
- GRASP 2: $\alpha = 0.75$ en instancias pequeñas y $\alpha = 0.25$ en instancias grandes.
- GRASP 3: $\alpha = 0.25$ en instancias pequeñas y grandes
- GRASP 4: $\alpha = 0.25$ en instancias pequeñas y grandes
- GRASP 5: $\alpha = 0.25$ en instancias pequeñas y grandes

Los resultados completos de todos los algoritmos para todos los valores de α se encuentran en el Apéndice A.

En la Tabla 4.6 se muestra el RPD promedio de los algoritmos GRASP (solo los resultados con el mejor valor α para cada algoritmo) comparados con las soluciones óptimas encontradas por el modelo de programación lineal entera mixta. La columna "$t(s)$" muestra el tiempo en segundos durante el cual los algoritmos fueron ejecutados en cada tamaño de instancias. El tiempo de ejecución de los algoritmos GRASP fue fijado dependiendo del tamaño de las instancias y es igual al utilizado en los últimos trabajos para el problema UPMS. En la tabla vemos que las metaheurísticas encuentran soluciones mucho más cercanas a las óptimas que los algoritmos heurísticos, donde el mejor algoritmo estaba a un 14.79 %, mientras que ahora, el mejor algoritmo está a un 2.77 %. Al contrario que en los resultados de los algoritmos heurísticos, el algoritmo que en promedio encuentra mejores soluciones es el GRASP 4, que sigue el segundo enfoque de resolución.

			Primer Enfoque		Segundo Enfoque		
Tamaño	#	$t(s)$	GRASP 1	GRASP 2	GRASP 3	GRASP 4	GRASP 5
6x2	16	3	6.08	1.91	3.89	3.53	4.01
6x3	24	3	8.13	5.97	3.42	3.57	3.61
6x4	27	3	7.91	2.53	3.67	2.40	3.88
6x5	33	3	10.82	2.99	2.09	1.59	3.00
Prom.			8.24	3.35	3.26	2.77	3.62

Tabla 4.6: *RPD para los algoritmos metaheurísticos con respecto a las soluciones en las instancias con $n = 6$ dónde el MILP probó optimalidad.*

Realizamos un *ANOVA* para comprobar si las diferencias vistas en la Tabla 4.6 son estadísticamente significativas. La Figura 4.9 muestra los intervalos HSD de Tukey al 95 % de confianza para el *RPD* promedio de los diferentes algoritmos en las instancias en las que se conoce la solución óptima. Podemos observar que el algoritmo GRASP 1 es significativamente peor que los demás algoritmos. Entre los otros algoritmos, a pesar de que el algoritmo GRASP 4 muestra mejores resultados en promedio con respecto a los demás, no se ven diferencias significativas entre ellos ya que los intervalos HSD solapan.

4.6.4. Resultados de los algoritmos metaheurísticos en el total de las instancias

A continuación, para ver el rendimiento de los algoritmos metaheurísticos en diferentes tamaños de problemas, continuamos con la comparación de estos en la totalidad de las instancias.

Comparación de los algoritmos metaheurísticos en el grupo de instancias pequeñas

La Tabla 4.7 muestra el *RPD* promedio de los cinco algoritmos GRASP en las 640 instancias pequeñas divididas en grupos tamaño $n \times m$, donde cada tamaño contiene 40 instancias. La columna "$t(s)$" muestra el tiempo en segundos durante el cual los algoritmos fueron ejecutados, variando dependiendo del tamaño de las instancias. En la tabla podemos ver que, confirmando lo visto en las instancias resueltas óptimamente, el GRASP 4 es el que

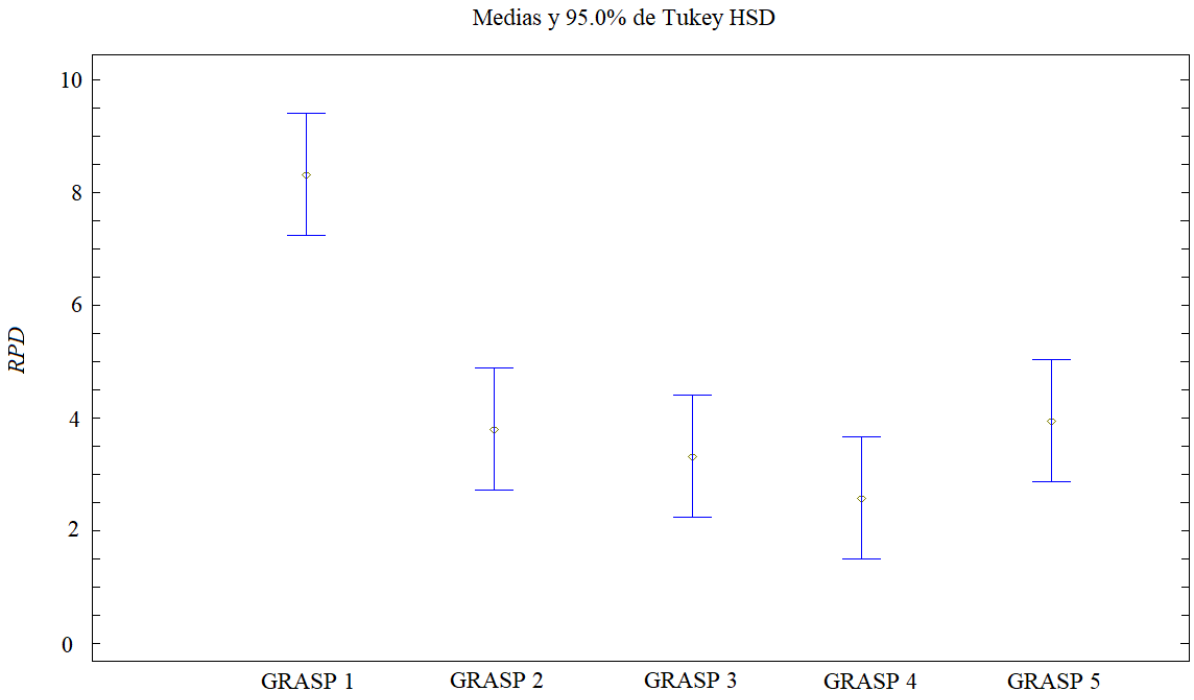

Figura 4.9: *Intervalos HSD de Tukey para los algoritmos metaheurísticos con respecto a las soluciones en las instancias con* $n = 6$ *dónde el MILP probó optimalidad.*

presenta mejores resultados comparado con los otros algoritmos. Sin embargo, en este grupo de instancias no se ven grandes diferencias en el rendimiento de los algoritmos, excepto por el GRASP 1 que rinde peor que las otras cuatro metaheurísticas. En este grupo de instancias no se observan grandes diferencias entre los dos enfoques de resolución, ya que el GRASP 2 (que sigue el primer enfoque) tiene un rendimiento muy parecido al del GRASP 4, superando a los algoritmos GRASP 3 y GRASP 5 (que siguen el segundo enfoque).

La Figura 4.10 muestra los intervalos HSD de Tukey para los algoritmos metaheurísticos en las instancias pequeñas. En esta figura podemos observar que no existen diferencias estadísticamente significativas entre los algoritmos, salvo por el algoritmo GRAPS 1 que es significativamente peor que los demás.

		Primer Enfoque		Segundo Enfoque		
Tamaño	$t(s)$	GRASP 1	GRASP 2	GRASP 3	GRASP 4	GRASP 5
6x2	3	4.76	1.16	3.23	2.65	3.24
6x3	3	6.91	4.45	2.83	2.43	2.69
6x4	3	7.74	2.35	2.82	2.23	2.29
6x5	3	10.53	2.71	1.56	1.51	2.30
8x2	3	3.73	2.55	2.13	1.19	1.36
8x3	3	8.63	2.53	4.98	4.05	4.98
8x4	3	11.77	3.41	4.41	3.88	3.88
8x5	3	18.15	7.14	5.35	5.31	6.12
10x2	5	3.99	1.97	2.29	2.08	2.90
10x3	5	7.29	4.30	4.60	3.70	3.92
10x4	5	7.00	5.44	3.15	2.18	2.73
10x5	5	10.97	6.23	6.16	5.41	6.31
12x2	5	3.55	3.18	3.36	3.06	3.27
12x3	5	5.51	4.46	4.36	3.43	4.04
12x4	5	7.16	4.18	6.23	6.23	6.27
12x5	5	11.51	4.09	6.08	5.28	5.96
Prom.		8.07	3.76	3.97	3.41	3.89

Tabla 4.7: *RPD promedio para los algoritmos metaheurísticos en el grupo de instancias pequeñas.*

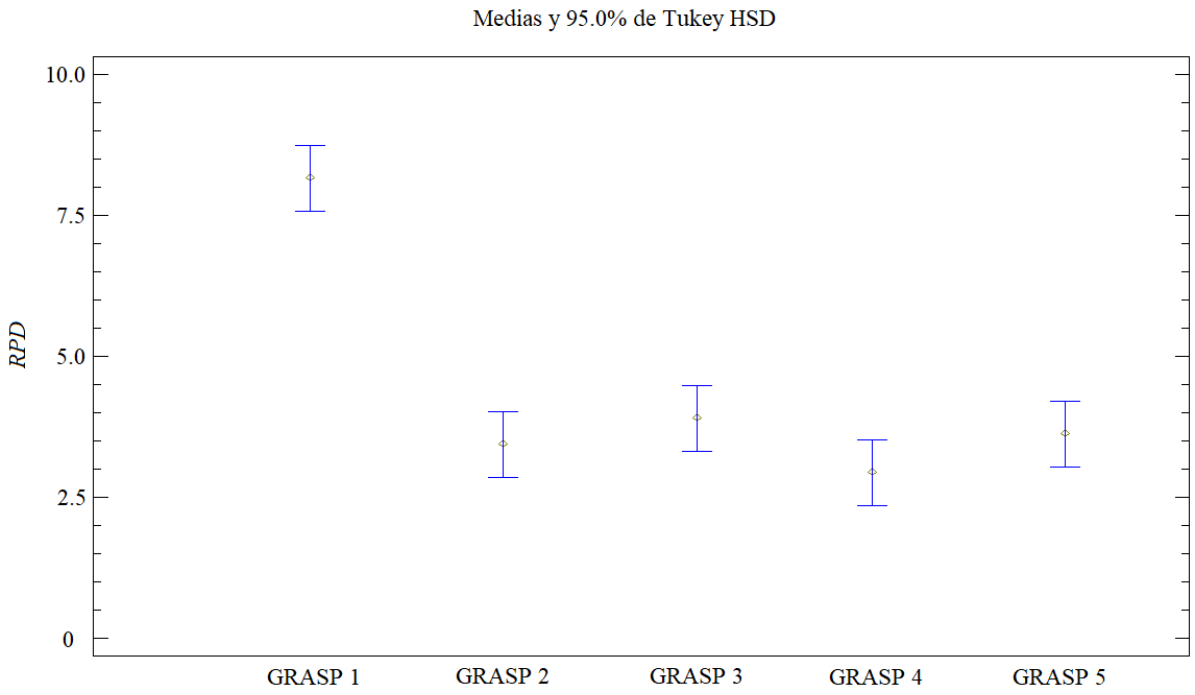

Figura 4.10: *Intervalos HSD de Tukey para los algoritmos metaheurísticos en las instancias pequeñas.*

Comparación de los algoritmos metaheurísticos en el grupo de instancias grandes

La comparación de los algoritmos metaheurísticos en el grupo de instancias grandes presentan un mayor interés, ya que su tamaño se ajusta más a lo que se puede encontrar en los problemas reales de producción. La Tabla 4.8 muestra el RPD promedio de los cinco algoritmos metaheurísticos en las 1000 instancias grandes divididas en grupos tamaño $n \times m$, donde cada tamaño contiene 40 instancias. En la tabla podemos confirmar lo visto con los algoritmos heurísticos, y los algoritmos que siguen el segundo enfoque tienen mucho mejor rendimiento que los algoritmos que siguen el primer enfoque, siendo el algoritmo GRASP 4 el que mejores resultados obtiene, con un RPD promedio de 0.52 %

Finalmente, al igual que en todas las comparaciones anteriores, realizamos un $ANOVA$ para comprobar si las diferencias son estadísticamente significativas. La Figura 4.11 muestra los intervalos HSD de Tukey para los algoritmos metaheurísticos en las instancias grandes.

		Primer Enfoque		Segundo Enfoque		
Tamaño	$t(s)$	GRASP 1	GRASP 2	GRASP 3	GRASP 4	GRASP 5
50x10	10	16.81	21.14	14.06	2.87	11.63
50x15	10	37.09	36.33	14.80	1.33	7.39
50x20	10	37.12	40.71	11.66	0.64	7.46
50x25	10	43.06	33.88	2.94	0.00	5.56
50x30	10	32.54	28.58	10.89	3.57	14.88
100x10	20	31.10	27.23	18.47	0.55	1.89
100x15	20	45.36	4061	14.14	0.21	7.90
100x20	20	52.56	48.52	8.45	0.00	3.62
100x25	20	54.66	54.03	10.29	0.18	3.62
100x30	20	57,46	48.12	17.95	0.72	4.45
150x10	30	19.85	27.26	13.90	0.23	9.82
150x15	30	34.90	43.83	4.57	0.34	5.27
150x20	30	41.71	51.74	9.40	0.69	15.93
150x25	30	42.26	51.44	18.97	0.31	11.00
150x30	30	40.46	53.36	7.53	0.07	0.09
200x10	40	20.29	26.22	15.16	0.35	14.86
200x15	40	35.46	41.22	8.02	0.17	9.96
200x20	40	37.34	44.70	1.20	0.05	13.93
200x25	40	41.78	48.61	15.14	0.06	2.10
200x30	40	46.51	58.54	11.53	0.17	15.77
250x10	50	22.32	28.44	0.24	0.03	2.44
250x15	50	35.97	39.98	13.62	0.11	12.52
250x20	50	48.69	48.69	14.21	0.11	2.60
250x25	50	42.85	51.93	17.38	0.08	14.72
250x30	50	43.88	55.88	16.50	0.21	13.49
Prom.		38.13	42.04	11.64	0.52	8.52

Tabla 4.8: *RPD para los algoritmos metaheurísticos en el grupo de instancias grandes.*

Podemos observar que el algoritmo GRASP 4 es significativamente mejor que los demás algoritmos. Además, podemos confirmar que existe diferencia significativa entre los algoritmos que siguen el primer enfoque y los que siguen el segundo enfoque, siendo mucho mejor el segundo.

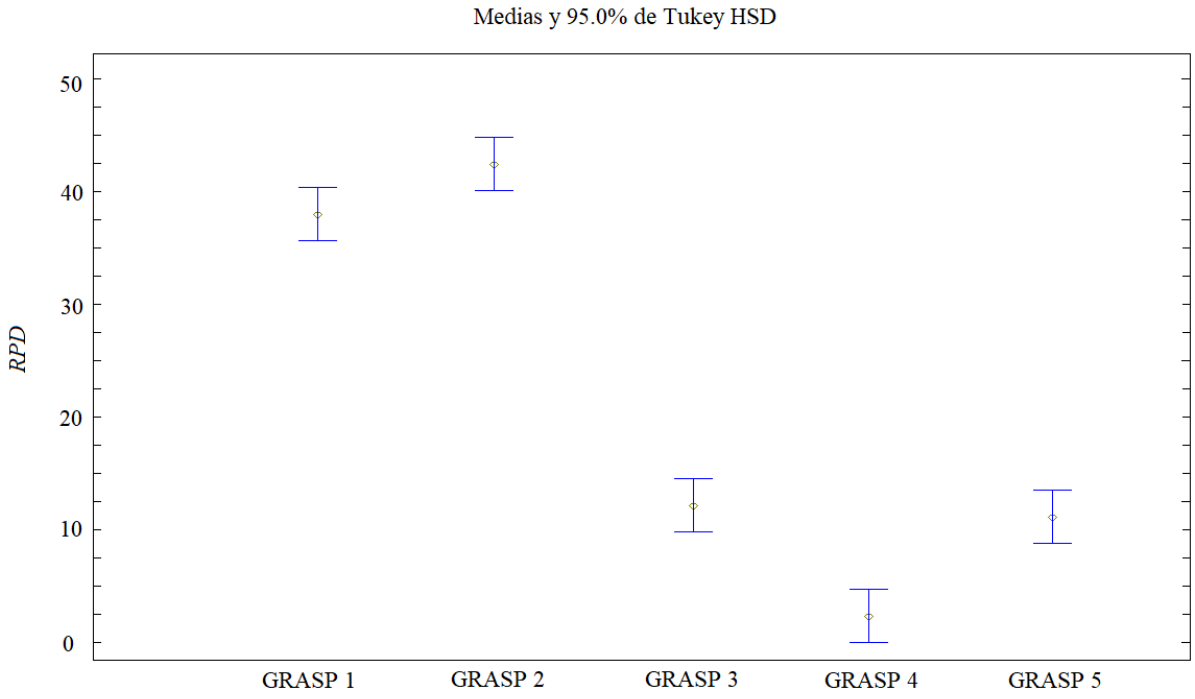

Figura 4.11: *Intervalos HSD de Tukey para los algoritmos metaheurísticos en las instancias grandes.*

4.6.5. Efecto de la búsqueda local en los algoritmos GRASP

Para verificar si la búsqueda local propuesta mejora los algoritmos GRASP, tomamos una muestra de 100 instancias del grupo de instancias grandes y las resolvemos con los algoritmos de dos maneras; aplicando la búsqueda local y sin aplicarla. Para la muestra de 100 instancias tomamos una instancia de cada una de las posibles configuraciones del conjunto de instancias grandes explicadas en la Sección 4.6. La Tabla 4.9 muestra la diferencia porcentual promedio para las 100 instancias entre los algoritmos con búsqueda local y los algoritmos sin búsqueda local. En la tabla podemos observar que para todos los algoritmos

la diferencia porcentual es positiva, lo que indica que para todos los casos la búsqueda local genera un efecto positivo en la mejora de las soluciones. Es interesante ver que el efecto de la búsqueda local es más grande en los algoritmos que siguen el primer enfoque constructivo. La razón de que veamos mayores diferencias en estos algoritmos puede deberse a que la búsqueda local propuesta tiene en cuenta el consumo de recursos, que al procesar soluciones que originalmente ignoraban totalmente este consumo, tiene mayor margen de mejora.

	GRASP 1	GRASP 2	GRASP 3	GRASP 4	GRASP 5
50x10	6.22	6.40	4.52	3.91	4.71
50x15	5.88	6.25	4.79	4.08	5.31
50x20	6.35	6.11	4.60	3.89	4.81
50x25	6.09	6.25	4.59	3.96	5.03
50x30	6.01	6.28	4.57	3.86	4.86
100x10	6.40	5.96	4.27	4.62	4.79
100x15	6.36	6.44	4.32	4.14	4.95
100x20	6.40	6.19	4.29	4.17	5.06
100x25	6.08	6.04	4.39	4.17	4.87
100x30	6.45	6.19	4.29	4.27	4.96
150x10	6.12	6.11	4.79	4.04	5.26
150x15	6.25	6.30	4.73	3.89	5.13
150x20	6.10	6.35	4.22	4.09	4.85
150x25	6.11	6.10	4.27	4.23	4.76
150x30	5.92	6.34	4.37	3.86	4.96
200x10	6.60	6.76	4.85	4.23	4.90
200x15	5.88	6.00	4.70	4.24	5.00
200x20	6.04	6.36	4.33	4.15	4.86
200x25	6.43	6.47	4.21	4.38	4.97
200x30	6.67	6.67	4.22	4.09	4.76
250x10	5.91	6.15	4.43	4.27	4.56
250x15	6.29	6.38	4.42	4.03	4.85
250x20	6.35	6.12	4.17	4.67	4.76
250x25	6.18	6.29	4.35	3.97	4.90
250x30	6.42	6.45	4.32	3.96	5.22
Promedio	6.22	6.28	4.44	4.13	4.92

Tabla 4.9: *Diferencias porcentuales entre las soluciones de los algoritmos GRASP con búsqueda local y sin búsqueda local.*

4.6.6. Comparación con el tercer enfoque de resolución

Como se dijo en la Sección 4.3.1, se probó un tercer enfoque de resolución, el cual fue rápidamente descartado debido a no mostrar resultados prometedores. La Tabla 4.10 muestra la comparación en las instancias grandes entre la heurística que sigue este enfoque y la mejor heurística de cada uno de los otros enfoques. En la tabla podemos observar que, el tiempo de cómputo del algoritmo del tercer enfoque es mucho más alto que en los otros dos enfoques. Esto es mucho más evidente en cuando aumenta el tamaño de las instancias, tardando hasta 12 minutos en terminar su ejecución. Además a la poca eficiencia en términos de tiempo, podemos observar que los resultados en el RPD también son mucho peores que los de los otros dos enfoques, siendo en promedio un 97 % peores que las mejores soluciones encontradas.

	Primer Enfoque		Segundo Enfoque		Tercer Enfoque	
Tamaño	RPD	$t(s)$	RPD	$t(s)$	RPD	$t(s)$
50x10	39.84	0.01	2.39	0.014	87.18	57.15
50x15	38.51	0.009	0.19	0.01	68.63	56.87
50x20	38.52	0.012	0.12	0.021	69.79	53.89
50x25	38.98	0.013	0.52	0.023	84.82	54.37
50x30	39.30	0.018	0.58	0.02	94.74	56.03
100x10	37.53	0.042	0.78	0.045	113.72	69.04
100x15	38.08	0.039	0.12	0.047	98.33	69.39
100x20	37.43	0.041	0.00	0.052	68.52	67.28
100x25	38.15	0.04	0.00	0.049	59.59	70.48
100x30	37.72	0.043	0.00	0.05	57.34	67.89
150x10	37.37	0.081	0.01	0.091	148.37	74.72
150x15	37.54	0.08	0.00	0.089	116.64	74.44
150x20	37.87	0.078	0.00	0.084	90.51	72.85
150x25	38.35	0.083	0.00	0.094	69.67	75.21
150x30	38.21	0.089	0.00	0.124	59.62	74.39
200x10	39.07	0.12	0.07	0.353	159.93	132.59
200x15	40.44	0.14	0.00	0.362	130.76	130.87
200x20	41.20	0.138	0.00	0.342	95.25	133.58
200x25	42.18	0.233	0.00	0.456	80.86	131.36
200x30	42.23	0.288	0.00	0.488	77.63	133.95
250x10	42.95	0.399	0.03	0.501	186.60	730.34
250x15	43.48	0.322	0.00	0.531	144.10	731.21
250x20	44.27	0.343	0.00	0.553	107.95	731.49
250x25	44.34	0.464	0.00	0.609	92.53	729.28
250x30	45.61	0.589	0.00	0.7	81.97	730.13
Prom.	39.97	0.148	0.19	0.228	97.80	212.35

Tabla 4.10: *RPD para los tres enfoques de resolución en el grupo de instancias grandes.*

Capítulo 5

Minimización simultánea del *makespan* y el número de recursos adicionales (Extensión multi-objetivo)

En este capítulo, definiremos formalmente y resolveremos el problema de secuenciación de máquinas paralelas no relacionadas con necesidad de ajustes y recursos adicionales asignados a los ajustes con dos objetivos simultáneos: 1) la minimización del *makespan*, y 2) la minimización del consumo de recursos. Este problema será denotado como MO-UPMSR-S, por sus siglas en inglés (*Multi-Objective Unrelated Parallel Machine scheduling problem with Setup times and additional limited Resources in the Setups*)

Presentaremos un nuevo algoritmo multi-objetivo para resolver el problema. Además, adaptaremos otros tres algoritmos multi-objetivo de la literatura para comparar y evaluar el rendimiento del algoritmo propuesto.

5.1. Definición formal del problema

Básicamente, la optimización multi-objetivo consiste en optimizar dos o más objetivos simultáneamente. Estos objetivos normalmente están en conflicto, es decir, la mejora en un objetivo implica el empeoramiento en otro. Una definición más formal del problema utilizando notación matemática sería:

Definición 2. Dadas n funciones $f_1, \ldots, f_n$, con $f_i : R \to R$, un problema de minimización

multi-objetivo consiste en mín $F(x) = (f_1(x), ..., f_n(x))$, s.a.: $x \in X$, siendo X la región factible del problema. Análogamente se definiría un problema de maximización multi-objetivo.

Para el MO-UPMSR-S, tendremos los mismos conjuntos y parámetros definidos en la Sección 4.1, salvo el parámetro $R_{\text{máx}}$, ya que en este problema multi-objetivo no tenemos restricción de recursos. En este caso, al no tener los recursos como una restricción, un consumo elevado de éstos no generará infactibilidad en la solución.

Existen varios enfoques de resolución de problemas de optimización multi-objetivo. El enfoque más común es el llamado "a priori". En este enfoque, los diferentes objetivos a optimizar se ponderan, teniendo como base el conocimiento previo del problema, formando un único objetivo, el cual se optimiza con métodos tradicionales de optimización mono-objetivo. Con notación matemática, dados unos pesos $\lambda_1, \ldots, \lambda_n$, un problema de minimización multi-objetivo se transforma en mín $\sum_i \lambda_i f_i(x)$; s.a.: $x \in X$. Un inconveniente que tiene este enfoque, es que se necesita información adicional sobre la importancia de los objetivos para poder ponderarlos. Esto sumado a que los objetivos pueden estar en unidades y escalas diferentes, hace aún más difícil su ponderación. Por otro lado, aunque se normalizasen los objetivos para subsanar los problemas de escala y unidades, el combinar objetivos hace difícil interpretar el resultado de la combinación dentro del contexto del problema.

El enfoque contrario, y el que se trabaja en esta tesis es el llamado "a posteriori". En este enfoque no se necesita conocimiento previo sobre la importancia de los diferentes objetivos, ya que se obtiene un conjunto de soluciones en las que se mide cada objetivo por separado. Con este conjunto de soluciones, se puede elegir la "mejor" solución dependiendo de lo que sea más conveniente. En este enfoque, al tener más de un objetivo a optimizar, se dificulta la comparación de soluciones, ya que una solución puede ser mejor que otra en un objetivo, pero peor en otro. Por ello, es imposible definir una solución como solución óptima y se define la frontera de Pareto. Si una solución forma parte de la frontera de Pareto, quiere decir que esta solución no es dominada por ninguna otra solución.

Definición 3. En un problema multi-objetivo, se dice que una solución factible S_a es do-

minada por otra solución factible S_b si S_b no es peor que S_a en ningún objetivo, y $S_a \neq S_b$. Con la notación anterior, S_a es dominada por S_b si $f_i(S_b) \leq f_i(S_a)$ y $S_a \neq S_b$. Una solución factible que no es dominada por ninguna solución es llamada *solución no dominada*. Al conjunto que contiene a todas las soluciones no dominadas se le llama *frontera de Pareto*

Las soluciones que forman parte de la frontera de Pareto no se pueden comparar entre sí, es decir, no se puede determinar qué solución es mejor entre ellas, ya que ninguna solución domina a otra.

Por otro lado, trabajar con un conjunto de soluciones hace más difícil la comparación de dos o más métodos de resolución, ya que cada uno devolverá una frontera de soluciones no dominadas. Posiblemente, el único caso en el que la comparación de dos conjuntos de soluciones es trivial, es cuando una de las fronteras de soluciones está totalmente dominada por otra, es decir, todas las soluciones de una de las fronteras son dominadas por al menos un elemento de la otra frontera de soluciones. La Figura Figura 5.1 (a) muestra un ejemplo de dos fronteras de Pareto en un problema con dos objetivos a minimizar. En la figura podemos observar que, ninguna solución de la Frontera 1 es dominada por alguna solución de la Frontera 2. Además, todas las soluciones de la Frontera 2, son dominadas por al menos una solución de la Frontera 1. En este ejemplo, es fácil comprobar que la Frontera 1 es mejor que la Frontera 2.

Por otra parte, puede haber casos en los que dos fronteras de Pareto no son fácilmente comparables, ya que algunas soluciones de una frontera pueden ser dominadas por soluciones de la otra frontera y viceversa. La Figura 5.1 (b) muestra un ejemplo de dos fronteras de Pareto difíciles de comparar en un problema con dos objetivos a minimizar. En la figura podemos ver que hay algunas soluciones de la Frontera 2 que dominan a algunas soluciones de la Frontera 1 y viceversa. En este caso no es nada intuitivo saber cuál de las dos fronteras de Pareto es mejor. Debido a esto, la comparación de fronteras de Pareto ha sido objeto de varios estudios. En Zitzler et al. (2003) y en la tesis doctoral de Minella (2014) se hace una revisión muy completa de las diferentes métricas utilizadas para comparar métodos multi-

objetivo, diferenciando entre métricas Pareto compatibles y métricas Pareto no compatibles. Como se explica en Knowles et al. (2006), una métrica es Pareto no compatible cuando puede obtenerse un mejor valor para una frontera de Pareto A con respecto a otra frontera de Pareto B, estando A dominada por B.

En esta tesis vamos a trabajar con dos métricas estudiadas en los trabajos de Zitzler et al. (2003) y Minella (2014): el indicador de hipervolumen y el indicador épsilon unario. Ambas métricas son Pareto compatibles como se demostró en Knowles et al. (2006).

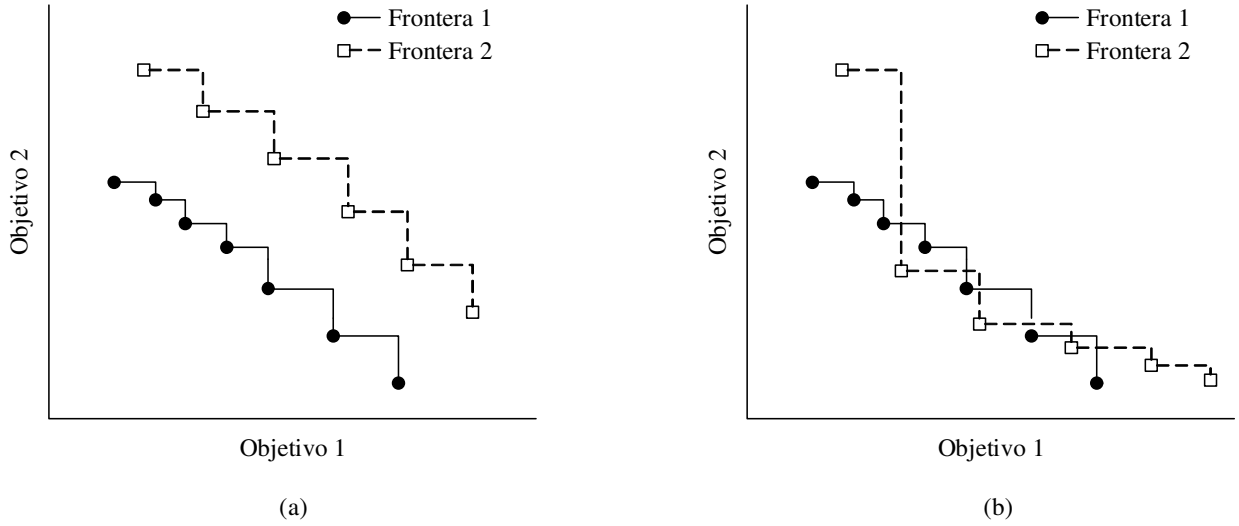

Figura 5.1: *Ejemplos comparación de soluciones multi-objetivo.*

5.1.1. Indicador de hipervolumen (I_H)

Este indicador fue introducido en Zitzler y Thiele (1999). La idea de este indicador es medir el volumen (o el área cuando se trabaja con dos objetivos) que hay entre una frontera de Pareto y un punto de referencia. A mayor valor del indicador, mejor es la frontera de Pareto evaluada. Para calcular este indicador, los valores en todos los objetivos son normalizados y escalados en el intervalo [0,1]. El valor en un objetivo de una solución x se normaliza con la siguiente fórmula: $f'_g(x) = (f_g(x) - f_g^{\text{mín}})/(f_g^{\text{máx}} - f_g^{\text{mín}})$, donde:

- $f_g(x)$ es el el valor de la solución x en el objetivo g.

- $f_g^{\text{mín}}$ es el valor mínimo en el objetivo g entre todas las soluciones encontradas entre todos los métodos de resolución a evaluar.

- $f_g^{\text{mín}}$ es el valor máximo en el objetivo g entre todas las soluciones encontradas entre todos los métodos de resolución a evaluar.

Como en el MO-UPMSR-S buscamos minimizar los dos objetivos propuestos, el punto de referencia elegido debe ser mayor que el valor máximo de ambos objetivos. Al estar normalizados, en ambos objetivos el valor máximo es 1. Teniendo en cuenta esto, y siguiendo el mismo procedimiento de cálculo del indicador de hipervolumen utilizado en Minella et al. (2011), elegimos $(1.2, 1.2)$ como punto de referencia para calcular el área. La Figura 5.2 muestra una representación gráfica del hipervolumen en un caso de minimización de dos objetivos. En esa figura, la zona sombreada representa el hipervolumen, que es el área comprendida entre la frontera de Pareto y el punto de referencia $(1.2, 1.2)$. Teniendo este punto de referencia, el valor máximo de I_H sería $1.2 \times 1.2 = 1.44$

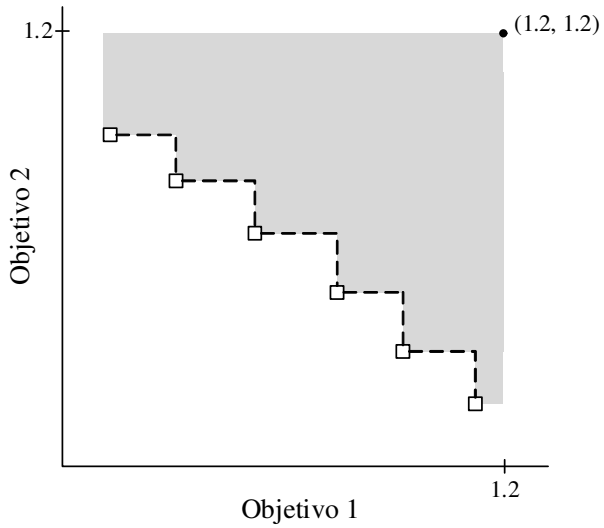

Figura 5.2: *Ejemplo del cálculo del indicador de hipervolumen (I_H).*

5.1.2. Indicador épsilon unario (I_ϵ^1)

Este indicador está basado en el indicador épsilon binario introducido en Zitzler et al. (2003), que mide la distancia entre dos fronteras de Pareto. En el caso binario, el indicador compara dos fronteras de Pareto, por lo que para comparar varios métodos de resolución, sería necesario calcular todos los pares posibles de métodos. La versión unaria del indicador épsilon (I_ϵ^1) es introducida en Knowles et al. (2006), donde una de las dos fronteras de Pareto a comparar es siempre la mejor frontera de Pareto conocida, es decir, una frontera de Pareto formada por todas las soluciones no dominadas entre todas las soluciones conocidas del problema estudiado. En este indicador, se estaría midiendo la distancia entre la frontera de Pareto de un método con respecto a la mejor frontera de Pareto conocida, por lo que se podría considerar como una extensión multi-objetivo del RPD utilizado en la Sección 4.6. A la mejor frontera de Pareto conocida la llamaremos frontera de referencia.

El indicador épsilon unario de un algoritmo dado en un problema donde se busca minimizar dos objetivos, se calcula de la siguiente manera:

$$I_\epsilon^1 = I_\epsilon(A, P) = \max_{p \in P} \min_{a \in A} \max_{1 \leq g \leq 2} \{f'_g(a)/f'_g(p)\}$$

donde:

- A es la frontera de Pareto obtenida por el algoritmo analizado.
- P es la mejor frontera de Pareto conocida, o frontera de referencia.
- $f'_g(a)$ es el valor normalizado en el objetivo g dado por la solución $a \in A$.
- $f'_g(p)$ es el valor normalizado en el objetivo g dado por la solución $p \in P$.

Es importante mencionar que, al igual que con el indicador de hipervolumen, para calcular el indicador épsilon unario también se normalizan y se escalan los valores en todos los objetivos. Sin embargo, para evitar divisiones por cero en el cálculo del indicador, los valores en todos los objetivos son normalizados en el intervalo [1,2] con la siguiente fórmula: $f'_g(x) = (f_g(x) - f_g^{\text{mín}})/(f_g^{\text{máx}} - f_g^{\text{mín}}) + 1$, donde:

- $f_g(x)$ es el el valor de la solución x en el objetivo g.
- $f_g^{\text{mín}}$ es el valor mínimo en el objetivo g entre todas las soluciones encontradas entre todos los métodos de resolución a evaluar.
- $f_g^{\text{mín}}$ es el valor máximo en el objetivo g entre todas las soluciones encontradas entre todos los métodos de resolución a evaluar.

Teniendo en cuenta que el problema mono-objetivo ya es considerado $\mathcal{NP}$-Hard como se dijo en el Capítulo 3, y la extensión multi-objetivo es aún más compleja, para resolver este problema propondremos algoritmos eficientes buscando soluciones aproximadas a la frontera de Pareto óptima.

5.2. Métodos de resolución del problema multi-objetivo

Para resolver el MO-UPMSR-S proponemos un algoritmo nuevo basado en el algoritmo llamado "*Restarted Iterated Pareto Greedy*" (RIPG) propuesto en Minella et al. (2011). El algoritmo que proponemos en esta tesis lo llamamos "*Truncated Restarted Iterated Pareto Greedy*" (T-RIPG) ya que, como se verá en la Sección 5.2.1, la modificación que se propone consiste en truncar una parte del algoritmo original para hacerlo más rápido y eficiente. Se eligió basarse en el algoritmo RIPG ya que éste, a su vez, es una adaptación multi-objetivo del algoritmo "*Iterated Greedy*" (IG) propuesto en Ruiz y Stützle (2007), que ha mostrado muy buenos resultados en varios estudios de secuenciación de máquinas. Además, el algoritmo RIPG mostró el mejor rendimiento al compararlo con 23 de los mejores algoritmos propuestos para problemas de secuenciación, identificados en la revisión bibliográfica hecha en Minella et al. (2008). Por último, en Ciavotta et al. (2013) se compara el RIPG con 17 métodos de optimización, mostrando también mejores resultados. Viendo estos resultados, decidimos tomar este método como base del algoritmo propuesto en este capítulo.

El algoritmo T-RIPG que proponemos para resolver el problema es comparado con otros tres algoritmos de la literatura, adaptados al MO-UPMSR-S. Los algoritmos adaptados son:

- El algoritmo NSGA-II propuesto en Deb et al. (2002).
- El algoritmo MOIGS propuesto en Framinan y Leisten (2008).
- El algoritmo RIPG original propuesto en Minella et al. (2011).

Estos tres algoritmos serán explicados en la Sección 5.2.2.

5.2.1. Algoritmo T-RIPG

El algoritmo T-RIPG propuesto para resolver el MO-UPMSR-S tiene cuatro fases principales: 1) fase de inicialización, 2) fase voraz (o *greedy* en inglés), 3) fase de temporización y 4) búsqueda local.

Además de estas cuatro fases, se incluye un operador de selección para elegir la solución que será procesada por las tres últimas fases. Finalmente, buscando evitar la rápida convergencia del algoritmo a óptimos locales, se agrega una fase de reinicio con la que se vuelve a la fase de inicialización y se crea un nuevo conjunto de soluciones iniciales. Durante todo el proceso, se trabaja con un conjunto de soluciones no dominadas, el cual se actualiza a medida que se encuentran nuevas soluciones no dominadas. Encontrar nuevas soluciones para este conjunto puede implicar la eliminación de soluciones que estaban anteriormente, ya que podría pasar que fueran dominadas por una de las nuevas soluciones encontradas. Este conjunto de soluciones no dominadas lo llamaremos a partir de ahora *conjunto de trabajo*. Es importante mencionar que este *conjunto de trabajo* es la misma frontera de Pareto parcial del algoritmo, por lo que, cuando el algoritmo termina, el *conjunto de trabajo* es la frontera de Pareto del algoritmo. La Figura 5.3 muestra el procedimiento general del algoritmo T-RIPG. En el diagrama podemos observar que, inmediatamente después de la fase de inicialización, se pasa a la fase voraz sin usar el operador de selección. Esto se hace para evitar que el algoritmo converja muy rápido, por lo que en la primera iteración pasan por la fase voraz todas las soluciones generadas en la fase de inicialización. También podemos observar que en un momento del algoritmo se decide si se aplica la fase de reinicio o no. Si

se decide no reiniciar el algoritmo, por no haber cumplido la condición para reiniciar, y no se ha cumplido la condición de finalización, se vuelve a seleccionar una solución del *conjunto de trabajo* para volver a la fase voraz con esa solución. Finalmente, podemos observar que si se reinicia el algoritmo, se vuelve a la fase de inicialización para generar nuevas soluciones que pasarán a la fase voraz.

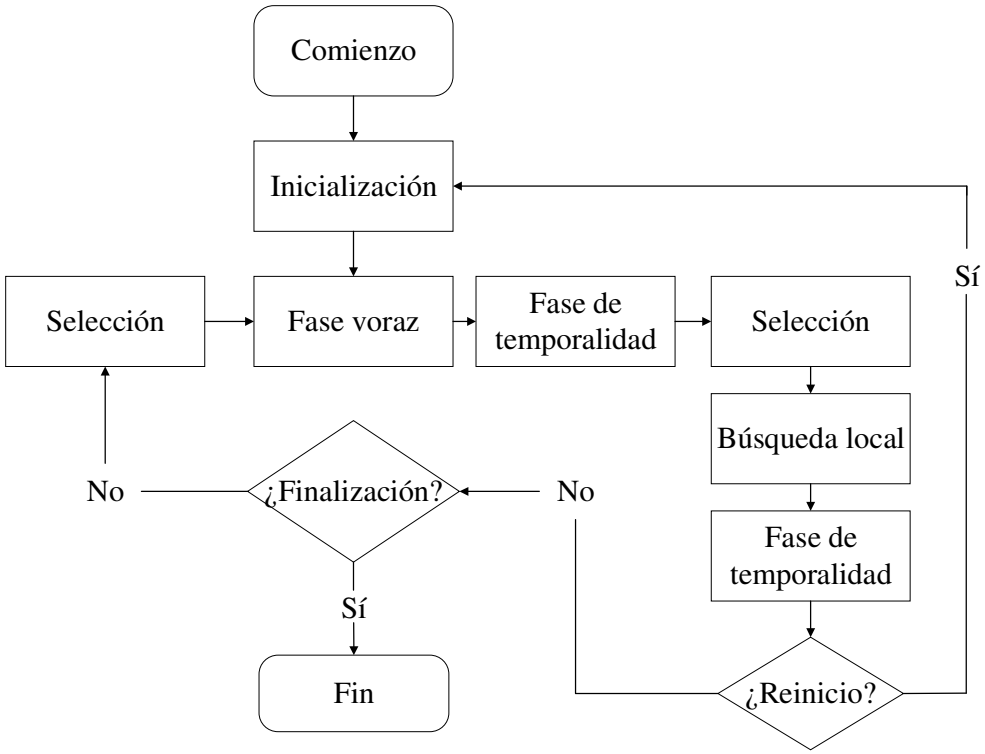

Figura 5.3: *Diagrama de flujo del algoritmo (T-RIPG).*

Inicialización

Para generar buenas soluciones iniciales en cuanto a *makespan* y a consumo de recursos, utilizamos el algoritmo GRASP 4 explicado en el Capítulo 4. La decisión de usar este algoritmo la tomamos porque fue, con diferencia, el mejor algoritmo resolviendo el problema minimizando únicamente el *makespan*, contando con una cantidad limitada de recursos (el problema UPMSR-S). Además, buscando soluciones con bajo *makespan* sin importar el consumo de recursos, se generan otras soluciones con el mismo algoritmo GRASP, pero

sin darle importancia a la información del consumo de recursos ($\theta_{r(i,j,k-1,k)} = \theta_{r(i,j,k,k+1)} = \gamma_{r(i,j,k)} = 1$ en el cálculo de $\lambda_{i,j,k}$, ver Sección 4.3.3).

En total, se generan cuatro soluciones iniciales distintas. Dos soluciones iniciales con el algoritmo GRASP 4 original, y dos soluciones más buscando únicamente tiempos de finalización bajos. Hacemos esto replicando el procedimiento del algoritmo RIPG original, donde generan $O \times 2$ soluciones, siendo O el número de objetivos a optimizar.

Finalmente, cada vez que el algoritmo pase por la fase de reinicio y vuelva a esta etapa de inicialización, el valor α utilizado (Sección 4.4) en el algoritmo GRASP aumenta para que sea un poco más aleatorio el proceso constructivo y las soluciones sean diferentes a las generadas en iteraciones previas. Recordemos que el valor α define el tamaño de la lista restringida de candidatos a asignar en la fase de construcción del algoritmo. Un valor bajo de α hace que la lista sea más pequeña, es decir, tendrá menos candidatos a elegir aleatoriamente en cada asignación. $\alpha = 0$ quiere decir que la lista solo contiene al mejor candidato y sería un algoritmo voraz. $\alpha = 1$ quiere decir que la lista contiene a todos los posibles candidatos, por lo que sería un algoritmo completamente aleatorio.

Es importante recordar que, para evitar que el algoritmo converja muy rápido, todas las soluciones que se generan en esta fase son procesadas por la fase voraz, ignorando el operador de selección como se ve en la Figura 5.3.

Fase voraz

En esta fase, la solución a procesar es destruida y reconstruida generando nuevas soluciones. El primer paso de esta fase consiste en quitar de la solución un grupo de d trabajos seleccionados aleatoriamente y ponerlos en un conjunto de trabajos a reasignar llamado D. Como el valor d puede afectar el rendimiento del algoritmo, este parámetro será calibrado junto a otros parámetros del algoritmo más adelante. Una vez los d trabajos han sido quitados, el siguiente paso es reconstruir la solución. En este paso es en el que se diferencia el algoritmo T-RIPG propuesto y el RIPG original (también del procedimiento IG general propuesto en Ruiz y Stützle, 2007). El algoritmo RIPG original toma un trabajo del conjun-

to D y prueba su reinserción en cada una de las posibles posiciones de la solución parcial. En cada prueba de reinserción se genera una nueva solución parcial (con el trabajo asignado en cada una de las posiciones en las que fue reinsertado). El siguiente paso es eliminar el trabajo que fue reinsertado en todas las posiciones del conjunto D. Para la siguiente iteración, el siguiente trabajo seleccionado de D para ser reinsertado es probado en todas las posiciones de todas las soluciones parciales generadas en el paso anterior. Al igual que antes, cada prueba de reinserción genera una nueva solución parcial con el nuevo trabajo insertado. Este proceso se repite hasta que todos los trabajos de D son reinsertados en la solución. Para hacer más rápido este proceso de reinserción, cuando un trabajo ha sido reinsertado en cada posición posible y se generan las nuevas soluciones parciales, éstas son evaluadas para eliminar de la lista de soluciones parciales todas aquellas que sean soluciones dominadas. Es importante aclarar que, cuando el último trabajo del conjunto D es reinsertado en todas las posiciones de todas las soluciones parciales, las soluciones que estas reinserciones generan son soluciones finales. Estas soluciones finales también son evaluadas para eliminar todas aquellas que sean dominadas.

Este procedimiento ha mostrado ser efectivo en los problemas en los que ha sido aplicado el RIPG anteriormente. Sin embargo, debido a la naturaleza del MO-UPMSR-S, las soluciones parciales obtenidas durante el proceso de reinserción de los trabajos, puede que no den suficiente información para saber si son buenas soluciones. Si se quisiera evaluar una solución parcial como buena o mala, sería necesario evaluar el consumo de recursos en cada instante de tiempo, por lo que en instancias grandes, con muchas soluciones parciales generadas, no sería eficiente desde un punto de vista computacional.

El nuevo procedimiento voraz, o "*greedy*", que proponemos es el siguiente: En lugar de probar la reinserción únicamente del primer trabajo del conjunto D, se prueba la reinserción de todos los trabajos de D en todas las posibles soluciones de la primera solución parcial (la solución que queda una vez se quitan los d trabajos). Una vez probada la reinserción de todos los trabajos en todas las posiciones, se asigna el mejor trabajo en la mejor posición

y se elimina este trabajo del conjunto D (se explica más adelante cómo saber cuál es el mejor trabajo). Este proceso se repite hasta que se han asignado $d-1$ trabajos, es decir, hasta que solo queda un trabajo en el conjunto D. Para decidir cuál de todas las posibles inserciones es la mejor, se utiliza el mismo valor λ_{ijk} de la Heurística 4 explicado en la Sección 4.5 y en la Sección 4.3.4. En cada prueba de reinserción, se calcula el valor λ_{ijk} y después de probar todos los trabajos en todas las posiciones, se asigna el trabajo con mejor valor λ_{ijk}. Es importante notar que durante todo este proceso, la asignación de cada trabajo solo genera una solución parcial, por lo que la siguiente iteración se hace insertando los trabajos restantes de D en todas las posiciones de esa única solución parcial.

Finalmente, el último trabajo de D es insertado en todas las posibles posiciones de la solución parcial, generando una solución final con cada inserción que se hace. El Algoritmo 10 muestra el procedimiento general de la fase voraz explicada en esta sección.

Todas las soluciones generadas en este paso pasan a la siguiente fase de temporalidad de soluciones.

Algoritmo 10: Fase voraz

```
Quitar aleatoriamente d trabajos de la solución y agregarlos al conjunto D;
mientras size(D) > 1 hacer
    para cada j ∈ D hacer
        Probar la inserción de j en cada posible posición k de cada máquina i y
         calcular λ_j = mín_ik λ_ijk;
    fin
    Asignar el trabajo con el menor λ_j y quitarlo de D;
fin
Insertar el último trabajo de D en todas las posibles posiciones de todas las
 máquinas y guardar cada inserción como solución;
```

Fase de temporalidad

Otra gran diferencia entre el algoritmo T-RIPG propuesto y el RIPG original, es la inclusión de una fase de temporalidad. En esta fase, buscando cubrir más el espacio de soluciones, se agregan tiempos ociosos en las máquinas antes del inicio de algunos ajustes, para reducir

el consumo de recursos, en ciertos instantes de tiempo de las soluciones generadas en la fase anterior. Para esta fase, utilizamos una adaptación del algoritmo de reparación explicado en la Sección 4.3.4.

El primer paso de esta fase es decidir si una solución será modificada por el algoritmo de reparación o no. Para esto, definimos una probabilidad p de que la solución sea reparada por el algoritmo. El parámetro p es otro de los parámetros que serán calibrados en la Sección 5.3.1. Ya que el problema explicado en el Capítulo 4 (UMPSR-S) tiene como parámetro inicial $R_{\text{máx}}$, mientras que en este caso multi-objetivo no tenemos esa restricción, para poder adaptar el algoritmo de reparación es necesario decidir el número máximo de recursos que tendrá la solución a reparar en esta fase. Este número máximo de recursos durante la reparación lo llamaremos $R^*_{\text{máx}}$.

Para elegir el valor de $R^*_{\text{máx}}$, lo primero que se hace es evaluar el consumo máximo de recursos de la solución antes de entrar a esta fase. A este valor lo denominamos r_u. Después, se elije un número aleatoriamente siguiendo una distribución uniforme discreta entre r_l y r_u, donde r_l es el máximo consumo de recursos de un ajuste en la solución (máximo r_{ijk}). Después de elegir el valor de $R^*_{\text{máx}}$, se utiliza el mismo algoritmo de reparación explicado en la Sección 4.3.4, utilizando $R^*_{\text{máx}}$ en lugar de $R_{\text{máx}}$. Es importante aclarar que cuando una solución pasa por esta fase, es posible que aumente su *makespan* dependiendo del número de instantes de tiempo en los que se modifica la solución.

Al terminar esta fase, todas las soluciones resultantes, las modificadas por este proceso y las que no, se agregan al *conjunto de trabajo*. Después de actualizar el *conjunto de trabajo*, todas las soluciones son evaluadas para eliminar las soluciones dominadas.

Fase de selección/Operador de selección

Para seleccionar la solución que pasará por la búsqueda local y por la fase voraz, se usa el mismo operador de selección del algoritmo RIPG original llamado "*Modified Crowding Distance Assignment*" (MCDA). Este operador está basado en el operador "*Crowding Distance*" (CD) propuesto en Deb et al. (2002). La idea de este operador es buscar las soluciones

más aisladas en todo el espacio de soluciones. Se buscan las soluciones más aisladas de las demás para favorecer la búsqueda en el vecindario de estas soluciones, intentando llenar los huecos que pueda tener el *conjunto de trabajo*. La Figura 5.4 muestra un ejemplo de un posible conjunto de soluciones no dominadas, donde una zona del espacio de soluciones está bastante cubierta, mientras que otra zona del espacio de soluciones no lo está. En el caso de ese ejemplo, el operador seleccionaría la solución S_6 buscando soluciones que cierren el hueco entre S_6 y las demás soluciones. La diferencia entre el operador MCDA y el operador CD original está en que en el MCDA se agrega un contador a cada solución del conjunto de soluciones. Este contador aumenta cada vez que una solución es seleccionada por el operador. Cada vez que el contador aumenta, la probabilidad de que se seleccione esa solución baja. Esto se hace para que el operador no seleccione siempre la misma solución y se exploren más zonas del espacio de soluciones.

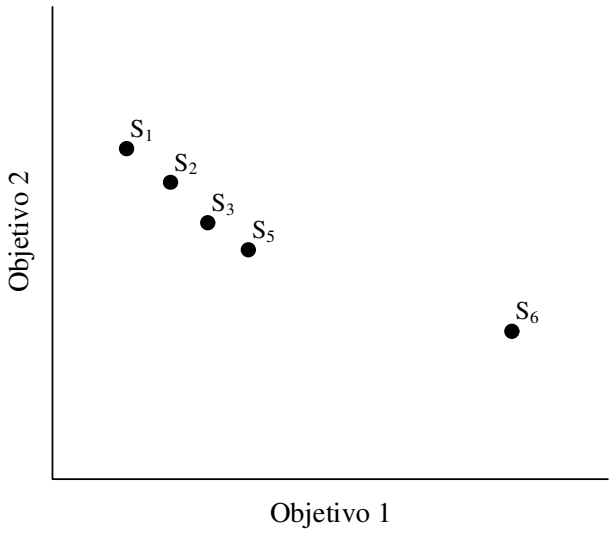

Figura 5.4: *Ejemplo de un conjunto de soluciones no dominadas.*

Búsqueda local

Una vez eliminadas las soluciones dominadas del *conjunto de trabajo*, el operador de selección elige una solución que pasará por la búsqueda local.

La búsqueda local propuesta en este algoritmo consiste en un procedimiento muy rápido y simple. Se quita aleatoriamente un trabajo de la máquina que da el *makespan* y se reinserta en todas las posibles posiciones de las otras máquinas, generando una nueva solución con cada reinserción del trabajo que se había quitado. Todas las soluciones generadas pasan a la fase de temporalidad, y posteriormente son evaluadas junto a las soluciones del *conjunto de trabajo* para eliminar las soluciones dominadas. Es importante decir que antes de aplicar la búsqueda local, es necesario justificar la solución hacia la izquierda, al igual que se explicó en la búsqueda local de la Sección 4.4.

Este proceso se repite hasta que después de ℓ iteraciones no se presenten mejoras en el *conjunto de trabajo*. Una iteración sin mejora en el *conjunto de trabajo* quiere decir que todas las soluciones generadas en la búsqueda local son dominadas por alguna solución del *conjunto de trabajo*. El parámetro ℓ también será calibrado en la Sección 5.3.1.

Fase de reinicio

La fase de reinicio consiste en obligar al algoritmo a volver a la fase de inicialización cuando se ha estancado y no encuentra nuevas soluciones para el *conjunto de trabajo*. Cuando el algoritmo lleva q iteraciones sin mejoras en el *conjunto de trabajo*, se activa la fase de reinicio. El parámetro q se calibrará en la Sección 5.3.1.

El primer paso consiste en guardar las soluciones del *conjunto de trabajo* antes de volver a la fase de inicialización. Una vez se generan otras cuatro soluciones nuevas en la fase de inicialización, éstas pasan por la fase voraz y por la fase temporalidad igual que en la primera iteración del algoritmo. Las soluciones resultantes de este proceso son agregadas al *conjunto de trabajo* (donde se encuentran las soluciones previamente guardadas) y evaluadas para eliminar las soluciones dominadas. Después de esto, el proceso del algoritmo continúa con el mismo procedimiento explicado a lo largo de esta sección.

El Algoritmo 11 muestra el proceso general del algoritmo T-RIPG.

Algoritmo 11: Pseudocódigo del algoritmo T-RIPG.

```
1  mientras TiempoEjecucion < LimiteTiempo hacer
2  |  ConjuntoIncial := FaseInicializacion;
3  |  para cada Solucion ∈ ConjuntoIncial hacer
4  |  |  FaseVoraz(Solucion);
5  |  |  FaseTemporalidad(Solucion);
6  |  fin
7  |  Borrar soluciones dominadas del conjunto de trabajo;
8  |  SolucionSeleccionada := FaseSeleccion;
9  |  mientras Iteraciones_BusquedaLocal_sin_mejora < ℓ hacer
10 |  |  BusquedaLocal(SolucionSeleccionada);
11 |  fin
12 |  mientras Iteraciones_sin_mejora < q hacer
13 |  |  SolucionSeleccionada := FaseSeleccion;
14 |  |  FaseVoraz(SolucionSeleccionada);
15 |  |  FaseTemporalidad(SolucionSeleccionada);
16 |  |  Borrar soluciones dominadas del conjunto de trabajo;
17 |  |  SolucionSeleccionada := FaseSeleccion;
18 |  |  mientras Iteraciones_BS_sin_mejora < ℓ hacer
19 |  |  |  BusquedaLocal(SolucionSeleccionada);
20 |  |  fin
21 |  fin
22 fin
```

5.2.2. Otros algoritmos multi-objetivo adaptados de la literatura

Como se dijo en la Sección 5.2, el algoritmo T-RIPG se compara con otros tres algoritmos adaptados de la literatura sobre optimización multi-objetivo, los cuales se explican a continuación. Para hacer una comparación más equitativa, a todos los algoritmos reimplementados se les agregó la fase de temporalidad, ya que, como se verá más adelante, esta fase mejora significativamente los resultados.

NSGA-II

El primer algoritmo adaptado para resolver el problema planteado en este capítulo, es el muy conocido algoritmo NSGA-II propuesto originalmente en Deb et al. (2002). En este algoritmo, se genera una población inicial de s soluciones con el mismo método con el que se generan las soluciones iniciales del algoritmo T-RIPG.

Una vez generadas las soluciones iniciales, se generan otras s soluciones a partir de las iniciales. Para esto, se utiliza el mismo operador de cruce del algoritmo genético propuesto en Vallada y Ruiz (2011). A estas soluciones generadas con el operador de cruce se les llama descendencia. Después de generar la descendencia, todas las soluciones (las iniciales y la descendencia) son agrupadas en grupos llamados "fronteras", donde el primer grupo contendrá todas las soluciones no dominadas. El segundo grupo, contendrá todas las soluciones dominadas únicamente por las soluciones de la primera frontera, pero no dominadas por otras soluciones. Se continúa con la creación de los siguientes grupos hasta que todas las soluciones están agrupadas.

El último paso de este algoritmo consiste en seleccionar las s mejores soluciones, empezando desde la primera frontera de soluciones no dominadas. Estas soluciones serán las soluciones iniciales de la siguiente iteración, es decir, las que se tomarán de base para utilizar el operador de cruce y generar nuevas soluciones.

Cuando se está llenando la lista de s soluciones, y una frontera tiene más elementos que los que faltan para completar el total, se utiliza el operador llamado "*crowding distance*",

que busca seleccionar las soluciones más aisladas. Este operador se usa hasta completar las s soluciones. Estas soluciones serán las s soluciones iniciales de la siguiente iteración del algoritmo. Este proceso se repite hasta que el tiempo de ejecución del algoritmo termina. Es importante aclarar que siempre que se genera una solución nueva, la fase de temporalidad es aplicada a esa solución. El Algoritmo 12 muestra el pseudocódigo de la adaptación de este algoritmo.

Algoritmo 12: Pseudocódigo de la adaptación del algoritmo NSGA-II.

```
1  P ← Generar s soluciones iniciales;
2  mientras TiempoEjecucion < LimiteTiempo hacer
3      Q ← ∅;
4      T ← ∅;
5      Aplicar operador de cruce a P para generar s nuevas soluciones;
6      Agregar las s nuevas soluciones a P;
7      Aplicar fase de temporalidad a todas las soluciones de P;
8      ContadorSoluciones = 0;
9      mientras ContadorSoluciones ≤ s hacer
10         Evaluar soluciones de P y guardar soluciones no dominadas;
11         Agregar a Q las soluciones no dominadas y eliminar éstas de P;
12         si size(Q) ≤ s-ContadorSoluciones entonces
13             T ← T ∪ Q;
14             ContadorSoluciones = ContadorSoluciones + size(Q);
15             Q ← ∅;
16         en otro caso
17             mientras ContadorSoluciones ≤ s hacer
18                 Seleccionar una solución de Q con el operador "crowding distance" y
                    agregarla a T;
19                 Eliminar solución seleccionada de Q;
20                 ContadorSoluciones = ContadorSoluciones + 1;
21             fin
22         fin
23     fin
24     P ← ∅;
25     P ← T;
26 fin
```

MOIGS

La adaptación del algoritmo MOIGS tiene algunas modificaciones con respecto al algoritmo general propuesto en Framinan y Leisten (2008). En el primer paso, al igual que en el algoritmo original, se generan dos soluciones iniciales. A continuación, se procesa cada solución por separado quitando aleatoriamente d trabajos que son agregados a un conjunto de trabajos a reasignar llamado D. Posteriormente, se toma el primer trabajo de este conjunto y se reinserta en todas las posibles posiciones de la solución destruida. Cada reinserción del trabajo genera una nueva solución parcial. Cuando el primer trabajo ha sido reinsertado en todas las posiciones, se evalúan las soluciones parciales para eliminar todas aquellas dominadas. Se continúa con el siguiente trabajo del conjunto D y se reinserta en todas las posibles posiciones de todas las soluciones parciales no dominadas generadas en el paso anterior. Este proceso se repite hasta que todos los trabajos de D hayan sido reinsertados.

El algoritmo MOIGS original repite este proceso pero a partir de la segunda iteración, con todas las soluciones no dominadas generadas en la iteración anterior. Esto puede causar que en las siguientes iteraciones, el grupo de soluciones iniciales a procesar sea numeroso, lo que puede generar que la evaluación constante del consumo de recursos sea costosa. Debido a esto, a partir de la segunda iteración el algoritmo es modificado para que sea más eficiente computacionalmente.

El cambio consiste en usar el mismo procedimiento voraz utilizado en el algoritmo T-RIPG, en el que después de quitar los d trabajos, se prueba la reinserción de todos los trabajos en todas las posibles posiciones de la solución parcial. Después de probar todos los trabajos, se asigna el mejor candidato y se repite el proceso hasta que queda solo un trabajo en D. El último trabajo es reasignado en todas las posiciones de la solución parcial y cada reinserción genera una nueva solución final. Todas las soluciones generadas son evaluadas para eliminar las soluciones dominadas. Esto se repite hasta que se hayan procesado todas las soluciones que iniciaron la iteración. Al acabar con todas las soluciones que iniciaron la iteración, el proceso se repite con las nuevas soluciones no dominadas mientras quede tiempo

de cómputo disponible.

Al final de cada iteración, cuando ya se tienen soluciones finales se aplica la fase de temporalidad antes de evaluar las soluciones y eliminar las dominadas. El Algoritmo 13 muestra el pseudocódigo de la adaptación del algoritmo MOIGS en este trabajo.

RIPG

Finalmente, para comprobar la relevancia de la nueva fase voraz propuesta en el algoritmo T-RIPG, se compara también con el algoritmo RIPG original propuesto en Minella et al. (2011). La diferencia entre el T-RIPG y el RIPG está únicamente en la fase voraz, el resto de fases del algoritmo original (inicialización, selección, búsqueda local y reinicio) fueron adaptadas igual que en el algoritmo T-RIPG.

Al igual que con los otros dos algoritmos adaptados de la literatura, a éste también se le agrega la fase de temporalidad para hacer más equitativa la comparación.

5.3. Resultados computacionales

Para comparar los cuatro algoritmos explicados en este capítulo, utilizamos un grupo de instancias generadas de forma similar al grupo de instancias grandes de la Sección 4.6, solo con la diferencia de que en este capítulo, se cambia la forma de generar el consumo de recursos.

Al igual que en la Sección 4.6, las instancias son generadas variando el número de trabajos (n), el número de máquinas (m), los tiempos de ajuste (s_{ijk}) y el consumo de recursos (r_{ijk}). Estos valores varían de la siguiente manera:

- $n \in \{50, 100, 150, 200, 250\}$.
- $m \in \{10, 15, 20, 25, 30\}$.
- Los tiempos de ajuste s_{ijk} fueron generados con cuatro distribuciones uniformes discretas diferentes en los conjuntos $\{1, \ldots, 9\}$, $\{1, \ldots, 49\}$, $\{1, \ldots, 99\}$ y $\{1, \ldots, 124\}$.

Algoritmo 13: Pseudocódigo de la adaptación del algoritmo MOIGS.

```
1  P ← Generar dos soluciones iniciales;
2  para cada i ∈ P hacer
3      Quitar aleatoriamente d trabajos de la solución y agregarlos al conjunto D;
4      SolParcial ← Solución sin los d trabajos eliminados;
5      ConjuntoSolParciales ← SolParcial;
6      para cada j ∈ D hacer
7          ConjuntoSolParcialesTemp ← ∅;
8          para cada n ∈ ConjuntoSolParciales hacer
9              Insertar el trabajo j en todas las posiciones de ConjuntoSolParciales_n,
                generando en cada inserción una nueva solución parcial que se agrega a
                ConjuntoSolParcialesTemp;
10         fin
11         ConjuntoSolParciales ← ∅;
12         ConjuntoSolParciales ← ConjuntoSolParcialesTemp;
13         Evaluar ConjuntoSolParciales y eliminar soluciones dominadas;
14     fin
15     Aplicar fase de temporalidad a todas las soluciones de ConjuntoSolParciales;
16     Evaluar ConjuntoSolParciales y eliminar soluciones dominadas;
17     mientras TiempoEjecucion < LimiteTiempo hacer
18         para cada n ∈ ConjuntoSolParciales hacer
19             Quitar aleatoriamente d trabajos de la ConjuntoSolParciales_n y
                agregarlos al conjunto D;
20             mientras size(D) > 1 hacer
21                 para cada j ∈ D hacer
22                     Probar la inserción de j en cada posible posición k de cada
                        máquina i y calcular λ_j = mín_ik λ_ijk; //Mismo λ del algoritmo
                        T-RIPG
23                 fin
24                 Asignar el trabajo con el menor λ_j y eliminarlo de D;
25             fin
26             Insertar el último trabajo de D en todas las posibles posiciones de todas
                las máquinas y guardar cada inserción como solución final en
                ConjuntoSolParciales;
27             Aplicar fase de temporalidad a todas las soluciones de
                ConjuntoSolParciales;
28             Evaluar ConjuntoSolParciales y eliminar soluciones dominadas;
29         fin
30     fin
31 fin
```

- El consumo de recursos r_{ijk} fue generado con dos distribuciones uniformes discretas diferentes en los conjuntos $\{1, \ldots, m\}$ y $\{1, \ldots, 5m\}$.

- Los tiempos de proceso p_{ij} fueron generados con una distribución uniforme discreta entre 1 y 99.

Combinando los diferentes valores de n, m, s_{ijk} y r_{ijk} obtenemos $5 \times 5 \times 4 \times 2 = 200$ configuraciones de instancias. Generamos 5 réplicas de cada configuración, por lo que en total tenemos $200 \times 5 = 1000$ instancias de evaluación. Además, generamos una réplica adicional de cada configuración para tener un conjunto de 200 instancias diferentes para realizar la calibración de los algoritmos.

Las métricas utilizadas para medir la calidad de los algoritmos son el indicador de hipervolumen I_H y el indicador épsilon unario I_ϵ^1 explicados en la Sección 5.1.

Todos los experimentos computacionales fueron ejecutados en máquinas virtuales con dos procesadores virtuales, con 8GB de memoria RAM y con Windows 10 Enterprise de 64 Bits. Las máquinas fueron virtualizadas en un entorno OpenStack soportado por 12 blades, cada uno con procesadores de 12 núcleos AMD Opteron Abu Dhabi 6344 corriendo a 2.6GHz y 256GB de memoria RAM, para un total de 576 nucleos y 3 terabytes de memoria RAM.

5.3.1. Calibración del algoritmo T-RIPG

Como se mencionó en la Sección 5.2.1, el algoritmo T-RIPG propuesto tiene cuatro parámetros a calibrar:

- El parámetro d que es el número de trabajos que se quitan en la fase voraz. Probamos cuatro niveles para este parámetro: $d = 0.05 \times n$, $d = 0.1 \times n$, $d = 4$ y $d = 5$. Los niveles $d = 4$ y $d = 5$ se probaron porque son los valores que mejores resultados han mostrado en diferentes trabajos que usan algoritmos basados en procedimientos "*Iterated greedy*" para problemas de secuenciación máquinas (Ruiz et al., 2019, Zhang et al., 2020, Minella et al., 2011).

- El parámetro p que es la probabilidad con la que una solución es reparada en la fase de temporalidad. Probamos este parámetro con cinco niveles: $p = 0$, $p = 0.25$, $p = 0.5$, $p = 0.75$ y $p = 1$. Cuando $p = 0$ ninguna solución pasa por la fase de temporalidad, mientras que $p = 1$ quiere decir que todas las soluciones son reparadas en esta fase.

- El parámetro ℓ es el número de iteraciones sin mejora en el *conjunto de trabajo* antes de parar la búsqueda local. Probamos este parámetro con cuatro niveles: $\ell = 0$, $\ell = 50$, $\ell = n$ y $\ell = 2 \times n$. Cuando $\ell = 0$ quiere decir que no se aplica búsqueda local.

- El parámetro q es el número de iteraciones sin mejora en el *conjunto de trabajo* antes de reiniciar el algoritmo con la fase de reinicio. Probamos este parámetro con cuatro niveles: $q = 0$, $q = 50$, $q = n$ y $q = 2 \times n$. De nuevo, $q = 0$ quiere decir que no hay fase de reinicio en el algoritmo.

Para la calibración, proponemos un diseño de experimentos factorial completo con todas las combinaciones de factores y niveles posibles para evaluar las diferentes configuraciones del algoritmo. Combinando todo ello, tenemos $4 \times 5 \times 4 \times 4 = 320$ configuraciones posibles del T-RIPG. Cada configuración fue probada en las 200 instancias de calibración generadas, para un total de 64,000 aproximaciones de la frontera de Pareto. El criterio de parada del algoritmo para cada configuración es el mismo, el cual depende del tamaño de la instancia: $t = n$ segundos. En total, cada una de las 320 configuraciones requiere 30,000 segundos para ejecutar las 200 instancias de calibración, para un total de 9,600,000 segundos, equivalente a un poco más de 111 días de tiempo de CPU para toda la calibración del T-RIPG.

Para comparar estadísticamente las diferentes configuraciones del algoritmo, hacemos un análisis de varianza ($ANOVA$). Para el análisis se tienen en cuenta cuatro factores: d, p, ℓ, q. Las variables respuesta del $ANOVA$ son los indicadores de hipervolumen y épsilon unario. Analizando los resultados del $ANOVA$, observamos que los cuatro factores son estadísticamente significativos en ambos indicadores. Los resultados detallados de este $ANOVA$ se encuentran en el Apéndice B.

Las Figuras 5.5 y 5.6 muestran los intervalos HSD de Tukey al 95 % de confianza para los indicadores de hipervolumen y épsilon unario para los 4 factores estudiados. En las figuras se puede observar fácilmente que los factores que presentan diferencias más significativas en ambos indicadores son p y q. Los valores de $p = 0$ y $q = 0$ muestran resultados mucho peores en ambos indicadores, mostrando la gran importancia de las fases de temporalidad y de reinicio (recordar que con esos parámetros iguales a cero no se ejecutan ni la fase de temporalidad ni la de reinicio). Así mismo, podemos observar que las configuraciones con $\ell = 0$ muestran peores resultados en ambos indicadores, mostrando también la relevancia de la búsqueda local (recordar que $\ell = 0$ indica que no se lleva a cabo la búsqueda local). A continuación se analiza cada factor individualmente:

- En el factor d observamos que existen diferencias significativas entre los diferentes valores, siendo $d = 4$ el valor que mejores soluciones obtiene en los dos indicadores.
- En el factor p, podemos observar que el valor que mejores resultados obtiene en ambos indicadores es el de $p = 1$.
- En el factor ℓ nos damos cuenta que, a pesar de que las configuraciones con $\ell = 50$ son ligeramente mejores que las otras, no hay diferencias significativas entre los valores $\ell = 50$, $\ell = n$ y $\ell = 2n$ en ninguno de los dos indicadores.
- En el factor q observamos que no hay grandes diferencias entre $q = 50$, $q = n$ y $q = 2n$. Sin embargo, en el indicador de épsilon unario el valor $q = 50$ es significativamente mejor que los demás valores.

Siguiendo los resultados de los experimentos, los valores óptimos de cada factor sin considerar interacciones son: $d = 4$, $p = 1$, $\ell = 50$ y $q = 50$.

Todos los parámetros de los otros algoritmos también fueron calibrados y los resultados detallados se encuentran en el Apéndice C.

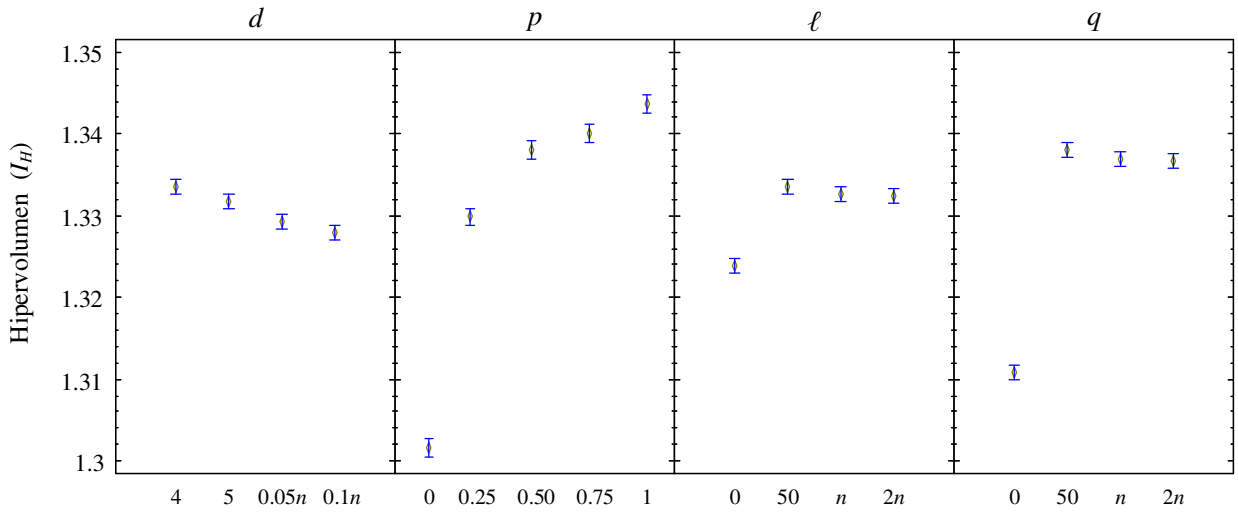

Figura 5.5: *Intervalos HSD de Tukey para el indicador de hipervolumen (I_H) para todos los factores calibrados del T-RIPG.*

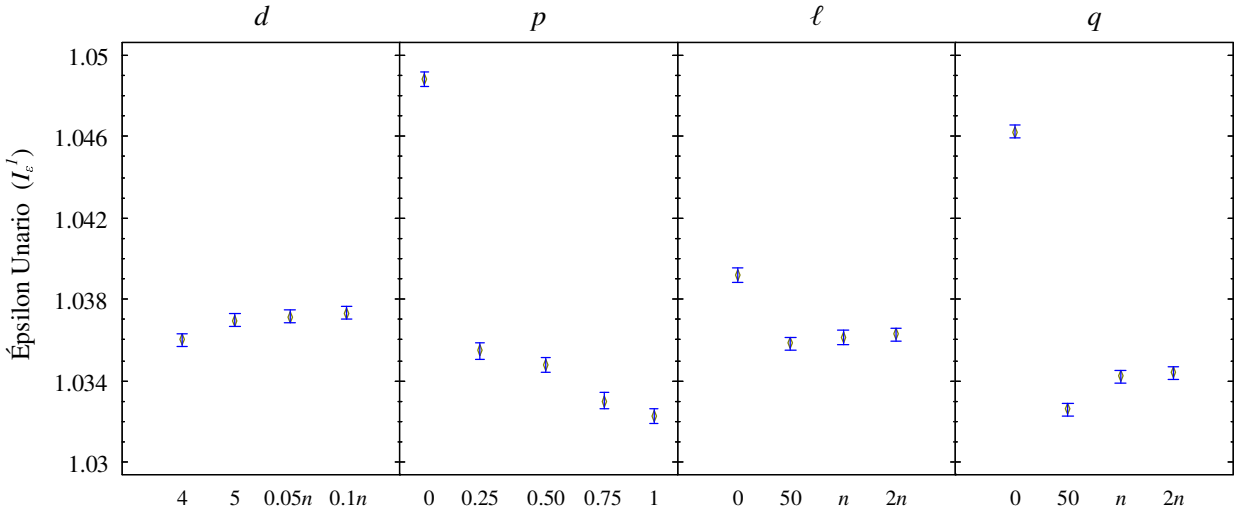

Figura 5.6: *Intervalos HSD de Tukey para el indicador épsilon unario (I_ϵ^1) para todos los factores calibrados del T-RIPG.*

5.3.2. Comparación computacional entre algoritmos

En esta sección comparamos el rendimiento de los cuatro algoritmos en las 1,000 instancias de evaluación. Al igual que en la calibración, todos los algoritmos tienen el mismo tiempo de ejecución que es $t = n$ segundos. La Tabla 5.1 muestra los valores promedio de I_H y de I_ϵ^1 para cada algoritmo en cada tamaño de instancias. En negrita tenemos los mejores resultados promedio para cada indicador y para cada tamaño de instancias. En esa tabla podemos ver que, en promedio, el algoritmo T-RIPG propuesto obtiene mejores resultados que los demás algoritmos en ambos indicadores. Además, se observa que el algoritmo T-RIPG obtiene mejores resultados en ambos indicadores en casi todos los tamaños de instancias, menos en los tamaños 150 $\times$ 20 y 250 $\times$ 25, donde el algoritmo RIPG obtiene resultados ligeramente mejores en el indicador de épsilon unario.

También observamos que el segundo mejor algoritmo es el algoritmo RIPG, ya que comparado con los algoritmos NSGA-II y MOIGS, tiene mejores resultados en promedio. Así mismo, es interesante ver que el algoritmo MOIGS es peor que los demás en ambos indicadores en instancias con menos trabajos, sin embrago, cuando el tamaño de las instancias aumenta, tiene mejores resultados que los algoritmos RIPG y NSGA-II en el indicador de hipervolumen.

Las Figuras 5.7 y 5.8 muestran el promedio de los indicadores de hipervolumen y épsilon unario en función del tamaño de las instancias. En esas figuras podemos ver que el algoritmo T-RIPG propuesto tiene mejores resultados (más altos) que los demás algoritmos en todos los tamaños de instancias en el indicador de hipervolumen. En el indicador épsilon unario, el algoritmo T-RIPG también presenta mejores resultados (más bajos) que los algoritmos MOIGS y NSGA-II en todos los tamaños de las instancias. Sin embargo, como se vio en la Tabla 5.1, el algoritmo RIPG parece funcionar un poco mejor en los tamaños 150 $\times$ 20 y 250 $\times$ 25. En esos tamaños de instancias, vemos que tenemos resultados de I_H y de I_ϵ^1 contradictorios, lo que indica que en esos dos tamaños de instancias, entre los algoritmos RIPG y T-RIPG, ninguno domina al otro.

	T-RIPG		RIPG		MOIGS		NSGA-II	
Tamaño	I_H	I_ϵ^1	I_H	I_ϵ^1	I_H	I_ϵ^1	I_H	I_ϵ^1
50 x 10	**1.2251**	**1.0291**	1.1056	1.0376	0.8479	1.2178	1.0678	1.1327
50 x 15	**1.2446**	**1.0230**	1.1050	1.0361	0.9096	1.1868	1.1176	1.0792
50 x 20	**1.2774**	**1.0226**	1.1818	1.0317	0.9678	1.1738	1.1711	1.0808
50 x 25	**1.2733**	**1.0271**	1.1539	1.0338	0.9149	1.1894	1.1618	1.0788
50 x 30	**1.2914**	**1.0325**	1.1693	1.0376	0.9444	1.2206	1.1993	1.0616
100 x 10	**1.2000**	**1.0405**	0.9812	1.0432	0.9765	1.1514	0.8938	1.1142
100 x 15	**1.2651**	**1.0266**	1.1095	1.0361	1.0319	1.1393	1.0727	1.0800
100 x 20	**1.2595**	**1.0206**	1.1388	1.0263	1.0820	1.0972	1.1019	1.0775
100 x 25	**1.2731**	**1.0192**	1.1355	1.0234	1.0667	1.0903	1.1316	1.0652
100 x 30	**1.2702**	**1.0151**	1.1113	1.0277	1.0467	1.1011	1.1200	1.0610
150 x 10	**1.2473**	**1.0251**	1.0163	1.0415	1.0703	1.1094	0.7789	1.1389
150 x 15	**1.2660**	**1.0139**	1.0352	1.0352	1.1133	1.1059	0.8600	1.1052
150 x 20	**1.2911**	1.0214	1.1360	**1.0119**	1.1312	1.0786	1.0175	1.0728
150 x 25	**1.3125**	**1.0105**	1.1578	1.0264	1.1864	1.0763	1.0915	1.0628
150 x 30	**1.2775**	**1.0121**	1.1074	1.0155	1.1285	1.0741	1.0270	1.0670
200 x 10	**1.3039**	**1.0121**	1.0429	1.0438	1.1305	1.1189	0.6771	1.2097
200 x 15	**1.3247**	**1.0167**	1.1410	1.0210	1.1816	1.0895	0.8482	1.1403
200 x 20	**1.3105**	**1.0103**	1.1070	1.0227	1.1728	1.0787	0.8531	1.1352
200 x 25	**1.3083**	**1.0133**	1.1501	1.0142	1.1842	1.0640	0.9275	1.0880
200 x 30	**1.3226**	**1.0091**	1.1338	1.0173	1.2035	1.0580	0.9521	1.0946
250 x 10	**1.3044**	**1.0162**	1.0216	1.0419	1.1819	1.1144	0.5984	1.2562
250 x 15	**1.3191**	**1.0157**	1.1191	1.0235	1.2097	1.0837	0.7590	1.1796
250 x 20	**1.2959**	**1.0167**	1.1250	1.0204	1.1842	1.0827	0.8059	1.1481
250 x 25	**1.3196**	1.0136	1.1707	**1.0122**	1.2546	1.0528	0.9227	1.1041
250 x 30	**1.3448**	**1.0103**	1.1722	1.0127	1.2249	1.0742	0.9404	1.1073
Prom.	**1.2851**	**1.0189**	1.1131	1.0277	1.0938	1.1132	0.9639	1.1096

Tabla 5.1: *Resultados computacionales para todos los algoritmos.*

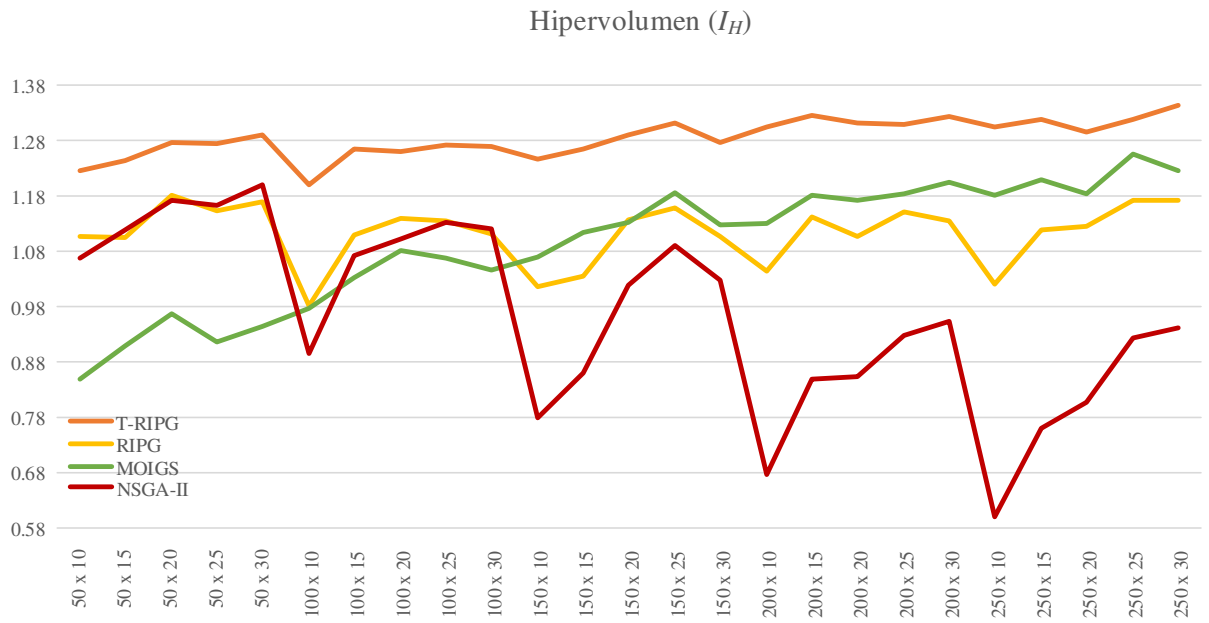

Figura 5.7: *Hipervolumen (I_H) promedio en función del tamaño de las instancias para todos los algoritmos.*

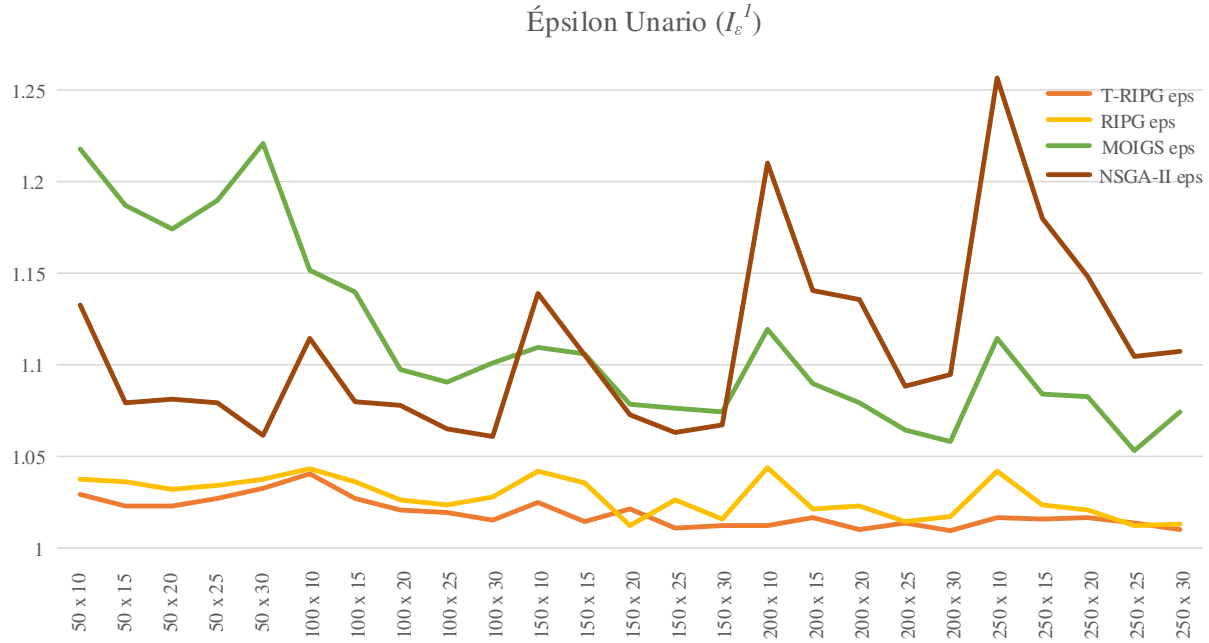

Figura 5.8: *Épsilon Unario (I_ϵ^1) promedio en función del tamaño de las instancias para todos los algoritmos.*

Para comprobar si el tamaño de las instancias influye en los resultados de los dos mejores algoritmos para los indicadores de hipervolumen y épsilon unario, hacemos un ANOVA multifactorial teniendo en cuenta tres factores: 1) El número de trabajos a procesar, 2) el número de máquinas y 3) el algoritmo utilizado. Las Figuras 5.9 y 5.10 muestran los gráficos de interacciones entre número de trabajos y algoritmo y entre el número de máquinas y algoritmo para ambos indicadores. En ambas figuras podemos observar que el número de máquinas parece afectar al algoritmo RIPG, ya que funciona mucho peor cuando las instancias a resolver tienen 10 máquinas. Así mismo, si observamos la interacción entre número de trabajos a procesar y algoritmo, parece que el RIPG funciona un poco peor para el indicador de épsilon unario en las instancias con 50 trabajos. En cuanto al algoritmo T-RIPG, parece que su rendimiento no se ve muy afectado por las variaciones en el número de trabajos a procesar o en el número de máquinas en ninguno de los dos indicadores.

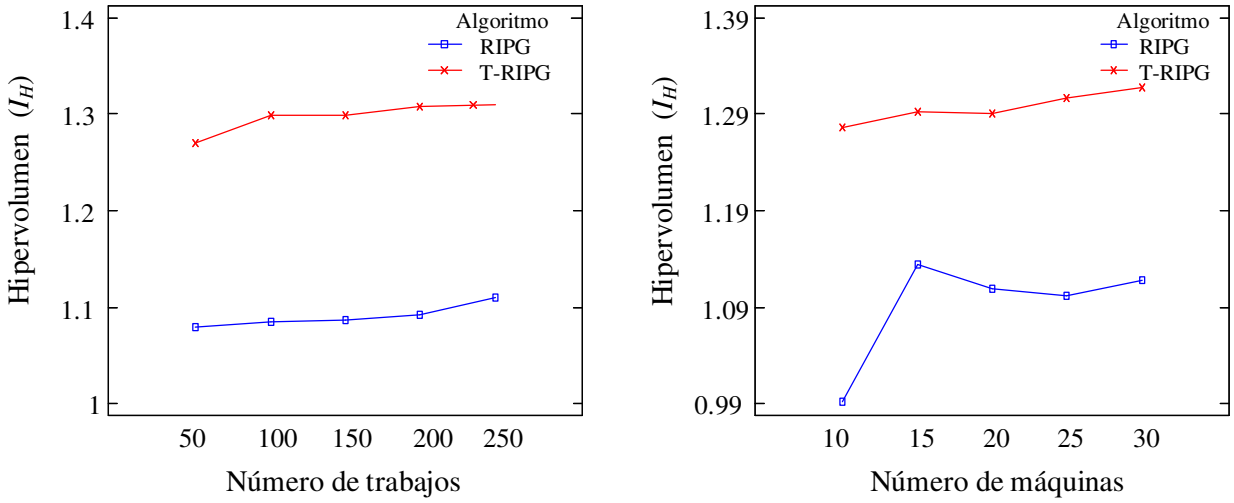

Figura 5.9: *Gráficos de interacciones entre número de trabajos y algoritmo y entre número de máquinas y algoritmo para el indicador de hipervolumen.*

Finalmente, para validar si las diferencias en I_H y en I_ϵ^1 entre los algoritmos son estadísticamente significativas, realizamos un $ANOVA$ con los dos indicadores como variable respuesta y el tipo de algoritmo como único factor. La Figura 5.11 muestra los intervalos HSD de Tukey al 95 % de confianza para los indicadores de hipervolumen y épsilon unario

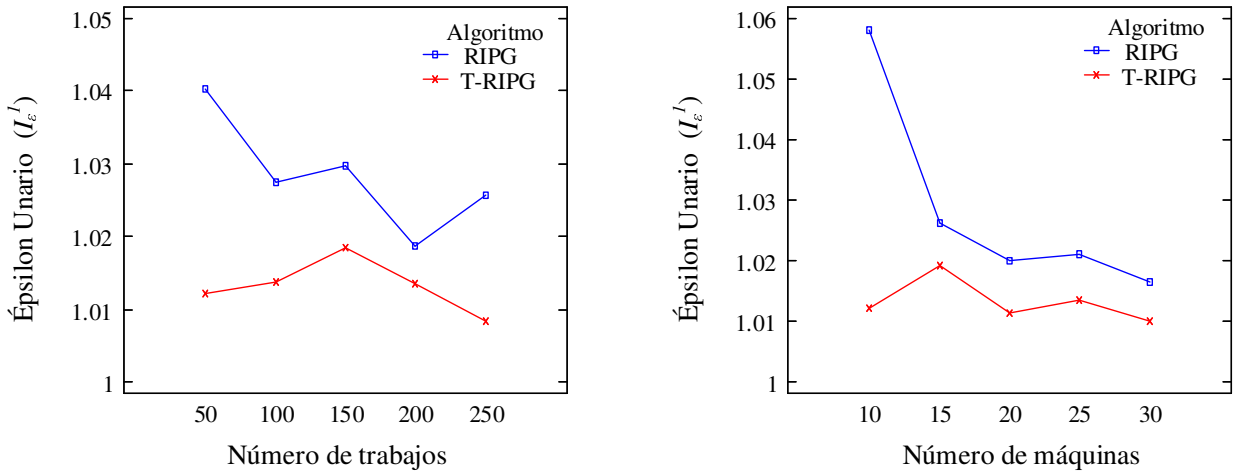

Figura 5.10: *Gráficos de interacciones entre número de trabajos y algoritmo y entre número de máquinas y algoritmo para el indicador de épsilon unario.*

para los diferentes algoritmos. En la figura podemos observar que el algoritmo T-RIPG es significativamente mejor que los demás algoritmos en los dos indicadores. El segundo mejor algoritmo en ambos indicadores es el algoritmo RIPG, mostrando mayores diferencias con respecto a los algoritmos NSGA-II y MOIGS en indicador épsilon unario. En cuanto a los algoritmos MOIGS y NSGA-II vemos que el algoritmo MOIGS es significativamente mejor en el indicador de hipervolumen, mientras que, aunque no hay diferencias significativas, el algoritmo NSGA-II es ligeramente mejor en el indicador épsilon unario.

Teniendo en cuenta que ambos indicadores han sido normalizados, las diferencias que vemos, especialmente en el indicador de hipervolumen, no solo son estadísticamente significativas, sino que muestran claramente que el algoritmo T-RIPG es el método más eficiente.

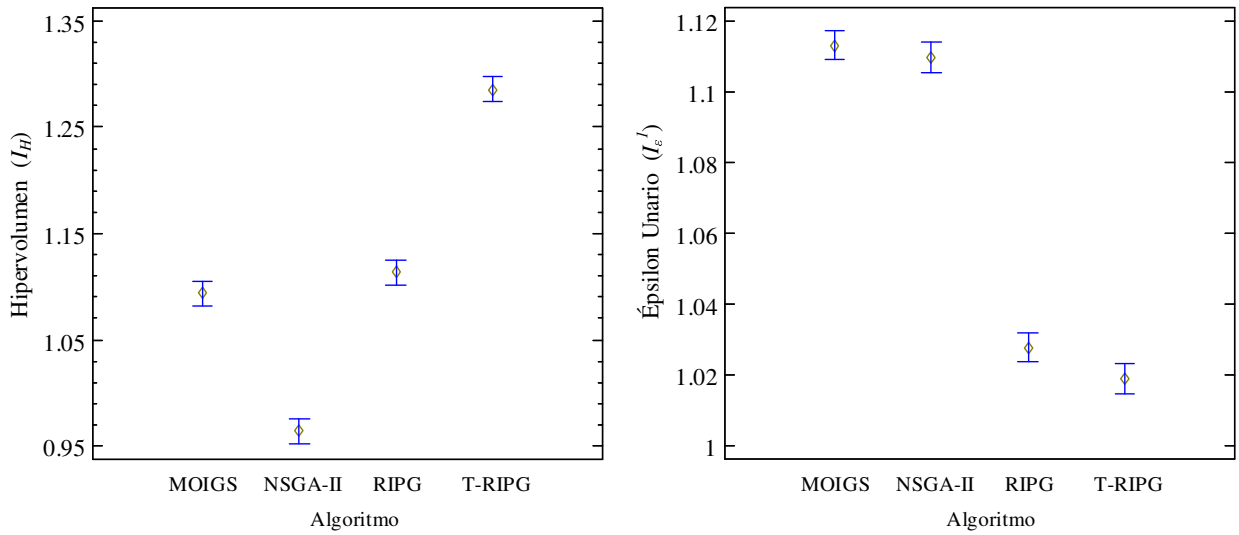

Figura 5.11: *Intervalos HSD de Tukey para I_H y I_ϵ^1 para los diferentes algoritmos.*

Capítulo 6

Conclusiones y líneas futuras

En esta tesis doctoral introducimos y trabajamos dos variaciones del problema de máquinas paralelas no relacionadas con necesidad de ajustes y recursos adicionales asignados a los ajustes:

1. El problema con el objetivo de minimizar el *makespan*, llamado UPMSR-S (*Unrelated Parallel Machine scheduling problem with Setup times and additional limited Resources in the Setups*).

2. El problema multi-objetivo buscando minimizar simultáneamente el *makespan* y el consumo de recursos, llamado MO-UPMSR-S (*Multi-Objective Unrelated Parallel Machine scheduling problem with Setup times and additional limited Resources in the Setups*).

Con el estudio de estos problemas, consideramos que logramos acercarnos un poco más a escenarios reales, ya que en muchos entornos productivos, el procesamiento de tareas por parte de las máquinas es automático y, sin embargo, el ajuste de estas máquinas debe ser realizado por un recurso adicional.

En el Capítulo 2 hicimos una revisión bibliográfica de los trabajos relacionados más relevantes, separando los trabajos multi-objetivo de los trabajos que solo tratan un objetivo. Con esta revisión, vemos que, a pesar del interés que han despertado los problemas de secuenciación de máquinas en las últimas décadas, hay muy pocos trabajos relacionados que

tengan en cuenta recursos adicionales (menos aún si los recursos son asignados a los ajustes de las máquinas). Así mismo, se encuentran muy pocos estudios de máquinas paralelas multi-objetivo.

Antes de explicar los métodos de resolución utilizados, en el Capítulo 3 explicamos los problemas de secuenciación de máquinas de forma general, empezando por el problema básico de máquinas paralelas idénticas, hasta llegar al estudiado en esta tesis que tiene en cuenta ajustes y recursos adicionales.

Para resolver el primer problema planteado, con el único objetivo de minimizar el *makespan*, en el Capítulo 4 propusimos un modelo de programación lineal entera mixta y dos enfoques de resolución heurística, ambos de tipo voraz o *greedy*. El modelo matemático propuesto pudo resolver únicamente instancias muy pequeñas, lo que justifica el uso de algoritmos heurísticos y metaheurísticos en instancias más grandes y, por lo tanto, más cercanas a entornos industriales realistas. Un aporte importante de esta tesis es la inclusión de la fase de reparación de soluciones en los algoritmos heurísticos y metaheurísticos, la cual permite obtener soluciones factibles en tiempos de cómputo muy cortos.

Al comparar los resultados de los algoritmos heurísticos con las soluciones óptimas obtenidas por el modelo matemático en las instancias en las que el modelo obtenía soluciones óptimas, observamos que, en promedio, los resultados de los algoritmos heurísticos propuestos están lejos de los óptimos. Al mejorar los algoritmos aplicando metaheurísticas (más concretamente proponemos algoritmos tipo GRASP), observamos que sus resultados son en promedio muy cercanos a las soluciones óptimas obtenidas por el modelo matemático.

Después de los experimentos computacionales en el total de instancias, observamos que el enfoque nuevo de resolución propuesto es muy superior a las adaptaciones de los mejores algoritmos tomados de la literatura del problema que no tienen en cuenta recursos adicionales en su fase constructiva. Esto muestra la importancia de explorar nuevos algoritmos y nuevos métodos de resolución a un problema que buscamos acercar un poco más a entornos industriales reales.

Adicionalmente, ya que el problema propuesto no había sido estudiado previamente en la literatura, no se contaba con un grupo de instancias para evaluar los métodos de resolución. Por esto propusimos una modificación de las instancias existentes para el problema sin recursos adicionales (conocido como *Unrelated Parallel Machine Scheduling problem*, o UPMS por sus siglas en Inglés), agregando el consumo de recursos y el máximo permitido de éstos para cada instancia. Este grupo de instancias servirá para futuros estudios del UPMSR-S.

La formulación del problema UPMSR-S y los resultados obtenidos fueron presentados en tres congresos:

- Yepes Borrero, J. C., Perea Rojas-Marcos, F., y Villa, F. (2018). Heurística para un problema de secuenciación de máquinas paralelas con necesidad de recursos en los ajustes. En *XXXVII Congreso Nacional de Estadística e Investigación Operativa*.

- Yepes Borrero, J. C., Perea Rojas-Marcos, F., y Villa, F. (2018). Scheduling with additional resources in the setups. En *The 29th European Conference on Operational Research (EURO2018)*.

- Yepes Borrero, J. C., Villa, F., y Perea Rojas-Marcos, F. (2019). GRASP algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. En *The 2nd Spanish Young Statisticians and Operational Researchers Meeting (SYSORM2019)*.

Además, los resultados obtenidos fueron publicados en un artículo en la revista *Expert Systems with Applications*:

- Yepes-Borrero, J. C., Villa, F., Perea, F., and Caballero-Villalobos, J. P. (2020). GRASP algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. *Expert Systems with Application*, 141:112959.

Finalmente, en el Capítulo 5 propusimos una variación multi-objetivo al problema, el cual denotamos como MO-UPMSR-S. Para este problema propusimos un nuevo algoritmo

llamado T-RIPG, basado en el algoritmo RIPG de Minella et al. (2011) que ha mostrado muy buenos resultados en diferentes problemas de secuenciación de máquinas multi-objetivo. Al no encontrar estudios sobre el MO-UPMSR-S, adaptamos otros algoritmos multi-objetivo para evaluar los resultados del nuevo algoritmo propuesto.

Para este problema, también propusimos un nuevo grupo de instancias para la evaluación de los algoritmos. Estas instancias servirán para futuros estudios del MO-UPMSR-S o problemas similares. Además, propusimos otro grupo de instancias de calibración, con las cuales calibramos todos los parámetros de los algoritmos estudiados.

En la calibración del algoritmo propuesto, vimos que todos los parámetros del algoritmo eran significativos, especialmente en la fase de temporalidad (similar a la fase de reparación del Capítulo 4) y la de reinicio. Debido a la relevancia de la fase de temporalidad, tal y como se mostró en la calibración, decidimos incluir esta fase a todos los algoritmos adaptados para hacer justa la comparación entre todos los algoritmos.

Al analizar la eficiencia de los algoritmos, y ser probados en el grupo de instancias de evaluación, el algoritmo T-RIPG propuesto en esta tesis, mostró ser mucho más eficiente que los algoritmos adaptados de otros estudios de problemas multi-objetivo.

Siguiendo todos los resultado obtenidos para el MO-UPMSR-S, podemos ver la gran importancia que tiene la fase de temporalidad, siendo la fase que más mejoraba los resultados de todos los algoritmos probados. Así mismo, podemos concluir que la modificación propuesta a la fase voraz del algoritmo RIPG original es muy efectiva, ya que la única diferencia entre el T-RIPG y el RIPG se encuentra en esta fase y el algoritmo T-RIPG obtuvo resultados significativamente mejores a los del RIPG. Estos resultados confirman que el nuevo método de construcción de soluciones propuesto en esta tesis, funciona mejor para este problema en comparación con otros algoritmos propuestos anteriormente para problemas similares.

La metodología propuesta para abordar la versión multi-objetivo ha sido enviada a la revista European Journal of Operational Research el día 27 de abril de 2020 y actualmente

se encuentra en fase de revisión del mismo tras haber recibido sugerencias de cambios por parte de los revisores.

En cuanto a líneas futuras de investigación, estamos trabajando en el problema mono-objetivo con tiempos de ajuste estocásticos. Consideramos que este problema es una aproximación realista a entornos variables, donde puede ser interesante considerar métodos alternativos buscando soluciones robustas ante la variabilidad. Este problema fue estudiado durante una estancia doctoral de seis semanas en la Universidad Javeriana de Bogotá, Colombia. El planteamiento general del problema se encuentra en el Apéndice D. Adicionalmente, creemos que los problemas de secuenciación de máquinas con necesidad de recursos adicionales es un campo de investigación aún poco estudiado, por lo que sería lógico continuar con el estudio de problemas similares, por ejemplo, con diferentes objetivos a optimizar, teniendo en cuenta uno o más objetivos simultáneamente, o extendiendo la necesidad de recursos a otro tipo de problemas como el del taller de flujo.

Bibliografía

Afzalirad, M. y Rezaeian, J. (2016). Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. *Computers & Industrial Engineering*, 98:40 – 52.

Aiex, R., Binato, S., y Resende, M. (2003). Parallel grasp with path-relinking for job shop scheduling. *Parallel Computing*, 29:393–430.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345 – 378.

Arbaoui, T. y Yalaoui, F. (2018). Solving the unrelated parallel machine scheduling problem with additional resources using constraint programming. In *Intelligent Information and Database Systems*, pages 716–725. Springer International Publishing.

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

Arnaout, J.-P., Musa, R., y Rabadi, G. (2014). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines - part ii: Enhancements and experimentations. *Journal of Intelligent Manufacturing*, 25:43–53.

Arnaout, J.-P., Rabadi, G., y Musa, R. (2009). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 21:693–701.

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

Avalos-Rosales, O., Angel-Bello, F., y Alvarez, A. (2014). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 76(9):1705–1718.

Bandyopadhyay, S. y Bhattacharya, R. (2013). Solving multi-objective parallel machine scheduling problem by a modified nsga-ii. *Applied Mathematical Modelling*, 37(10):6718 – 6729.

Barnes, J. W. y Vanston, L. K. (1981). Scheduling jobs with linear delay penalties and sequence dependent setup costs. *Operations Research*, 29(1):146–160.

Binato, S., Hery, W. J., Loewenstern, D. M., y Resende, M. G. C. (2002). A grasp for job shop scheduling. In *Essays and Surveys in Metaheuristics*, pages 59–79, Boston, MA. Springer US.

Bitar, A., Dauzère-Pérès, S., Yugma, G. C., y Roussel, R. (2014). A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. *Journal of Scheduling*, (19):367–376.

Chen, J.-F. (2005). Unrelated parallel machine scheduling with secondary resource constraints. *International Journal of Advanced Manufacturing Technology*, 26(3):285–292.

Cheng, T. y Sin, C. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3):271 – 292.

Choobineh, F. F., Mohebbi, E., y Khoo, H. (2006). A multi-objective tabu search for a single-machine scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 175(1):318 – 337.

Ciavotta, M., Minella, G., y Ruiz, R. (2013). Multi-objective sequence dependent setup times

flowshop scheduling: a new algorithm and a comprehensive study. *European Journal of Operational Research*, 227(2):301–313.

Cochran, J. K., Horng, S.-M., y Fowler, J. W. (2003). A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines. *Computers & Operations Research*, 30(7):1087 – 1102.

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

De Castro, L. y Von Zuben, F. (2002). Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6:239 – 251.

Deb, K., Pratap, A., Agarwal, S., y Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182 – 197.

Diana, R. O. M., de França Filho, M. F., de Souza, S. R., y de Almeida Vitor, J. F. (2015). An immune-inspired algorithm for an unrelated parallel machines' scheduling problem with sequence and machine dependent setup-times for makespan minimisation. *Neurocomputing*, 163:94–105.

Edis, E. y Ozkarahan, I. (2012). Solution approaches for a real-life resource-constrained parallel machine scheduling problem. *International Journal of Advanced Manufacturing Technology*, 58:1141–1153.

Edis, E. B. y Oguz, C. (2012). Parallel machine scheduling with flexible resources. *Computers & Industrial Engineering*, 63(2):433 – 447.

Edis, E. B., Oguz, C., y Ozkarahan, I. (2013). Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *European Journal of Operational Research*, 230(3):449 – 463.

Fanjul-Peyró, L. (2010). *Nuevos algoritmos para el problema de secuenciación en máquinas paralelas no relacionadas y generalizaciones*. PhD thesis, Universitat Politècnica de València.

Fanjul-Peyró, L. (2020). Models and an exact method for the unrelated parallel machine scheduling problem with setups and resources. *Expert Systems with Applications: X*, 5:100022.

Fanjul-Peyró, L., Perea, F., y Ruiz, R. (2017). Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operational Research*, 260(2):482–493.

Fanjul-Peyró, L., Ruiz, R., y Perea, F. (2019). Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *Computers & Operations Research*, 101:173 – 182.

Feo, T., Venkatraman, K., y Bard, J. (1991). A grasp for a difficult single machine scheduling problem. *Computers & Operations Research*, 18:635–643.

Feo, T. A. y Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67 – 71.

Feo, T. A., Sarathy, K., y McGahan, J. (1996). A grasp for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Computers & Operations Research*, 23(9):881 – 895.

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

Garey, M. R. y Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA.

Graham, R., Lawler, E., Lenstra, J., y Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In Hammer, P., Johnson, E., y Korte, B., editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier.

Hansen, M. (2000). Tabu search for multiobjective combinatorial optimization: Tamoco. *Control and Cybernetics*, 29:799–815.

Helal, M., Rabadi, G., y Al-Salem, A. (2006). A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research*, 3:182–192.

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, Part C (Applications and Reviews)*, 28(3):392–403.

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

Jones, D., Mirrazavi, S., 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.

Juan, A. A., Barrios, B. B., Vallada, E., Riera, D., y Jorba, J. (2014). A simheuristic algorithm for solving the permutation flow shop problem with stochastic processing times. *Simulation Modelling Practice and Theory*, 46:101 – 117. Simulation-Optimization of Complex Systems: Methods and Applications.

Juan, A. A., Faulin, J., Grasman, S. E., Rabe, M., y Figueira, G. (2015). A review of sim-

heuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. *Operations Research Perspectives*, 2:62 – 72.

Kim, D.-W., Kim, K.-H., Jang, W., y Chen, F. F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, 18(3):223 – 231. 11th International Conference on Flexible Automation and Intelligent Manufacturing.

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

Kurz, M. E. y Askin, R. G. (2001). Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, 39(16):3747–3769.

Laguna, M., Barnes, J., y Glover, F. (1991). Tabu search methods for a single machine scheduling problem. *Journal of Intelligent Manufacturing*, 2:63–73.

Laguna, M. y Glover, F. (1993). Integrating target analysis and tabu search for improved scheduling systems. *Expert Systems with Applications*, 6(3):287 – 297.

Lee, H. y Guignard, M. (1996). A hybrid bounding procedure for the workload allocation problem on parallel unrelated machines with setups. *The Journal of the Operational Research Society*, 47:1247–1261.

Lee, Y. H. y Pinedo, M. (1997). Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, 100(3):464 – 474.

Lenstra, J., Kan, A. R., y Brucker, P. (1977). Complexity of machine scheduling problems. In Hammer, P., Johnson, E., Korte, B., y Nemhauser, G., editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343 – 362. Elsevier.

Liu, C.-H. y Tsai, W.-N. (2015). Multi-objective parallel machine scheduling problems by considering controllable processing times. *Journal of the Operational Research Society*, 67:654–663.

Manupati, V., Rajyalakshmi, G., Chan, F. T. S., y Thakkar, J. J. (2017). A hybrid multi-objective evolutionary algorithm approachfor handling sequence- and machine-dependent set-up timesin unrelated paralle l machine scheduling problem. *Sadhana*, 42:391–403.

Minella, G. (2014). *Optimización multi-objetivo para la programación de la producción*. PhD thesis, Universitat Politècnica de València.

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

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

Mokotoff, E. (2001). Parallel machine scheduling problems: A survey. *Asia-Pacific Journal of Operational Research*, 18:193–242.

Montgomery, D. C. (2006). *Design and Analysis of Experiments*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

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.

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

Ovacik, I. M. y Uzhoy, R. (1993). Worst-case error bounds for parallel machine scheduling problems with bounded sequence-dependent setup times. *Operations Research Letters*, 14(5):251 – 256.

Paula, M., Ravetti, M., Mateus, G., y Pardalos, P. (2007). Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics*, 18:101–115.

Perea, F., Ruiz, R., y Fanjul-Peyró, L. (2016). Mip models for the scheduling of unrelated parallel machines with sequence dependent setup times and a scarce resource. Congreso SEIO. Toledo (España).

Rabadi, G., Moraga, R., y Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17:85–97.

Rajkumar, M., Asokan, P., Anilkumar, N., y Page, T. (2011). A grasp algorithm for flexible job-shop scheduling problem with limited resource constraints. *International Journal of Production Research*, 49:2409–2423.

Rostami, M., Pilerood, A. E., y Mazdeh, M. M. (2015). Multi-objective parallel machine scheduling problem with job deterioration and learning effect under fuzzy environment. *Computers & Industrial Engineering*, 85:206 – 215.

Ruiz, R. y Andrés-Romano, C. (2011). Scheduling unrelated parallel machines with resource-assignable sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 57(5):777–794.

Ruiz, R., Pan, Q.-K., y Naderi, B. (2019). Iterated greedy methods for the distributed permutation flowshop scheduling problem. *Omega*, 83:213–222.

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-Torres, A. J., López, F. J., y Ho, J. C. (2007). Scheduling uniform parallel machines subject to a secondary resource to minimize the number of tardy jobs. *European Journal of Operational Research*, 179(2):302 – 315.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278 – 285.

T'kindt, V. y Billaut, J.-C. (2001). Multicriteria scheduling problems: A survey. *RAIRO - Operations Research*, 35:143 – 163.

Torabi, S., Sahebjamnia, N., Mansouri, S., y Bajestani, M. A. (2013). A particle swarm optimization for a fuzzy multi-objective unrelated parallel machines scheduling problem. *Applied Soft Computing*, 13(12):4750 – 4762.

Vallada, E. y Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):612–622.

Vallada, E., Villa, F., y Fanjul-Peyró, L. (2019). Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem. *Computers & Operations Research*, 111:415 – 424.

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. Multicriteria Scheduling.

Villa, F., Vallada, E., y Fanjul-Peyró, L. (2018). Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource. *Expert Systems with Applications*, 93:28 – 38.

Wang, S. y Liu, M. (2015). Multi-objective optimization of parallel machine scheduling integrated with multi-resources preventive maintenance planning. *Journal of Manufacturing Systems*, 37:182 – 192.

Yenisey, M. M. y Yagmahan, B. (2014). Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega*, 45:119 – 135.

Yepes-Borrero, J. C., Villa, F., Perea, F., y Caballero-Villalobos, J. P. (2020). Grasp algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. *Expert Systems with Applications*, 141:112959.

Zhang, Z., Tang, Q., Ruiz, R., y Zhang, L. (2020). Ergonomic risk and cycle time minimization for the u-shaped worker assignment assembly line balancing problem: A multi-objective approach. *Computers & Operations Research*, 118:104905.

Zheng, X. y Wang, L. (2016). A two-stage adaptive fruit fly optimization algorithm for unrelated parallel machine scheduling problem with additional resource constraints. *Expert Systems with Applications*, 65:28 – 39.

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., y Fonseca, V. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7:117 – 132.

Apéndice A

Resultados completos para los algoritmos GRASP

En este apéndice mostramos los resultados completos de los algoritmos GRASP introducidos en la Sección 4.6. Todas las instancias fueron ejecutadas tres veces por cada algoritmo GRASP con tres diferentes valores de α: $\alpha = 0.25$, $\alpha = 0.5$ y $\alpha = 0.75$.

A continuación se muestran los resultados completos de cada uno de los algoritmos GRASP en todas las instancias.

A.1. GRASP 1

A.1.1. Resultados completos en el grupo de instancias pequeñas

La Tabla A.1 muestra el RPD promedio para los tres valores de α probados en el algoritmo GRASP 1 en las instancias pequeñas. En la tabla se observa que el mejor valor de α para este algoritmo en este grupo de instancias es el de 0.75.

A.1.2. Resultados completos en el grupo de instancias grandes

La Tabla A.2 muestra el RPD promedio para los tres valores de α probados en el algoritmo GRASP 1 en las instancias grandes. En la tabla se observa que el mejor valor de α para este algoritmo en este grupo de instancias es el de 0.25.

Tamaño	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$
6x2	7.32	5.78	4.76
6x3	12.08	7.64	6.91
6x4	15.32	10.73	7.74
6x5	17.83	9.20	10.53
8x2	7.21	4.44	3.73
8x3	13.37	9.65	8.63
8x4	19.23	12.57	11.77
8x5	6.64	5.55	3.99
10x2	6.64	5.55	3.99
10x3	12.28	9.36	7.29
10x4	13.38	11.44	7.00
10x5	19.49	12.72	10.97
12x2	5.89	3.98	3.55
12x3	8.62	6.69	5.51
12x4	12.73	9.51	7.16
12x5	17.01	15.56	11.51
Prom.	12.19	8.77	7.19

Tabla A.1: *RPD promedio para del algoritmo GRASP 1 para todos los valores de α en las instancias pequeñas.*

A.2. GRASP 2

A.2.1. Resultados completos en el grupo de instancias pequeñas

La Tabla A.3 muestra el RPD promedio para los tres valores de α probados en el algoritmo GRASP 2 en las instancias pequeñas. En la tabla se observa que el mejor valor de α para este algoritmo en este grupo de instancias es el de 0.75.

A.2.2. Resultados completos en el grupo de instancias grandes

La Tabla A.4 muestra el RPD promedio para los tres valores de α probados en el algoritmo GRASP 2 en las instancias grandes. En la tabla se observa que el mejor valor de α para este algoritmo en este grupo de instancias es el de 0.25.

Tamaño	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$
50x10	16.81	21.67	32.81
50x15	37.09	44.62	50.67
50x20	37.12	47.70	65.06
50x25	43.06	56.78	78.76
50x30	32.54	57.67	88.57
100x10	31.10	28.46	42.27
100x15	45.36	45.29	65.65
100x20	52.56	51.37	76.33
100x25	54.66	56.08	86.27
100x30	57.46	55.90	90.54
150x10	19.85	26.15	32.20
150x15	34.90	42.65	49.82
150x20	41.71	48.01	57.41
150x25	42.26	51.79	62.05
150x30	40.46	50.12	62.34
200x10	20.29	27.01	33.98
200x15	35.46	42.55	55.23
200x20	37.34	44.51	56.87
200x25	41.78	50.75	63.60
200x30	46.51	57.51	74.97
250x10	22.32	27.67	38.89
250x15	35.97	43.54	55.64
250x20	39.93	51.05	66.01
250x25	42.85	51.80	70.78
250x30	43.88	54.82	77.56
Prom.	38.13	45.42	61.37

Tabla A.2: *RPD promedio para del algoritmo GRASP 1 para todos los valores de α en las instancias grandes.*

Tamaño	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$
6x2	9.57	2.79	1.16
6x3	10.37	6.12	4.45
6x4	11.76	5.56	2.35
6x5	6.27	4.69	2.71
8x2	10.22	5.32	2.55
8x3	17.13	5.46	2.53
8x4	18.12	9.00	3.41
8x5	5.53	2.99	1.97
10x2	5.53	2.99	1.97
10x3	11.76	6.50	4.30
10x4	15.66	9.79	5.44
10x5	12.79	8.20	6.23
12x2	5.47	4.26	3.18
12x3	9.61	7.59	4.46
12x4	13.59	5.76	4.18
12x5	10.32	9.32	4.09
Prom.	10.86	6.02	3.44

Tabla A.3: *RPD promedio para del algoritmo GRASP 2 para todos los valores de α en las instancias pequeñas.*

A.3. GRASP 3

A.3.1. Resultados completos en el grupo de instancias pequeñas

La Tabla A.5 muestra el RPD promedio para los tres valores de α probados en el algoritmo GRASP 3 en las instancias pequeñas. En la tabla se observa que el mejor valor de α para este algoritmo en este grupo de instancias es el de 0.25.

A.3.2. Resultados completos en el grupo de instancias grandes

La Tabla A.6 muestra el RPD promedio para los tres valores de α probados en el algoritmo GRASP 3 en las instancias grandes. En la tabla se observa que el mejor valor de α para este algoritmo en este grupo de instancias es el de 0.25.

Tamaño	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$
50x10	21.14	21.15	19.19
50x15	36.33	32.22	34.03
50x20	40.71	36.33	39.99
50x25	33.88	34.55	36.85
50x30	28.58	22.30	19.71
100x10	27.23	25.50	31.10
100x15	40.61	38.23	45.36
100x20	48.52	50.75	52.56
100x25	54.03	53.03	54.66
100x30	48.12	50.56	57.46
150x10	27.26	29.06	30.31
150x15	43.83	44.09	45.04
150x20	51.74	51.76	57.31
150x25	51.44	55.45	59.72
150x30	53.36	57.66	55.72
200x10	26.22	29.67	28.80
200x15	41.22	42.92	44.24
200x20	44.70	44.89	46.88
200x25	48.61	50.28	51.46
200x30	58.54	56.40	58.47
250x10	28.44	28.10	30.20
250x15	39.98	39.79	43.21
250x20	48.69	49.06	50.41
250x25	51.93	53.88	52.52
250x30	55.88	55.84	59.00
Prom.	42.04	43.74	44.17

Tabla A.4: *RPD promedio para del algoritmo GRASP 2 para todos los valores de α en las instancias grandes.*

Tamaño	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$
6x2	3.23	2.07	2.30
6x3	2.83	2.58	3.90
6x4	2.82	5.10	8.93
6x5	1.56	3.05	4.14
8x2	2.13	1.23	2.50
8x3	4.98	4.07	7.84
8x4	4.41	6.98	10.69
8x5	5.35	3.25	3.78
10x2	2.29	3.58	4.11
10x3	4.60	4.54	7.56
10x4	3.15	5.50	11.15
10x5	6.16	7.85	10.98
12x2	3.36	4.02	4.93
12x3	4.36	6.85	9.86
12x4	6.23	11.80	12.96
12x5	6.08	15.36	14.94
Prom.	3.97	5.49	7.54

Tabla A.5: *RPD promedio para del algoritmo GRASP 3 para todos los valores de α en las instancias pequeñas.*

A.4. GRASP 4

A.4.1. Resultados completos en el grupo de instancias pequeñas

La Tabla A.7 muestra el RPD promedio para los tres valores de α probados en el algoritmo GRASP 4 en las instancias pequeñas. En la tabla se observa que el mejor valor de α para este algoritmo en este grupo de instancias es el de 0.25.

A.4.2. Resultados completos en el grupo de instancias grandes

La Tabla A.8 muestra el RPD promedio para los tres valores de α probados en el algoritmo GRASP 4 en las instancias grandes. En la tabla se observa que el mejor valor de α para este algoritmo en este grupo de instancias es el de 0.25.

Tamaño	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$
50x10	14.06	18.92	29.95
50x15	14.80	17.54	33.03
50x20	11.66	29.00	42.81
50x25	2.94	26.65	42.76
50x30	10.89	33.60	55.83
100x10	18.47	20.88	36.52
100x15	14.14	27.16	42.69
100x20	8.45	26.82	42.27
100x25	10.29	14.21	31.38
100x30	17.95	27.89	48.49
150x10	13.90	23.53	35.31
150x15	4.57	19.31	32.66
150x20	9.40	17.60	32.91
150x25	18.97	19.63	34.25
150x30	7.53	22.97	38.91
200x10	15.16	8.74	22.82
200x15	8.02	20.70	38.92
200x20	1.20	16.93	36.29
200x25	15.14	22.69	44.24
200x30	11.53	26.42	51.89
250x10	0.24	18.55	34.79
250x15	13.62	19.26	41.22
250x20	14.21	14.69	42.46
250x25	17.38	18.65	49.09
250x30	16.50	18.01	56.04
Prom.	11.64	21.22	39.90

Tabla A.6: *RPD promedio para del algoritmo GRASP 3 para todos los valores de α en las instancias grandes.*

Tamaño	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$
6x2	2.65	1.35	1.58
6x3	2.43	2.05	3.37
6x4	2.23	4.36	8.19
6x5	1.51	2.27	3.37
8x2	1.19	1.14	2.42
8x3	4.05	3.95	7.72
8x4	3.88	6.36	10.06
8x5	2.08	2.82	3.34
10x2	2.08	2.82	3.34
10x3	3.70	4.54	7.55
10x4	2.18	5.31	10.96
10x5	5.41	7.08	10.21
12x2	3.06	4.02	4.92
12x3	3.43	6.77	9.77
12x4	6.23	11.13	12.29
12x5	5.28	14.79	14.37
Prom.	3.41	5.05	7.09

Tabla A.7: *RPD promedio para del algoritmo GRASP 4 para todos los valores de α en las instancias pequeñas.*

A.5. GRASP 5

A.5.1. Resultados completos en el grupo de instancias pequeñas

La Tabla A.9 muestra el *RPD* promedio para los tres valores de α probados en el algoritmo GRASP 5 en las instancias pequeñas. En la tabla se observa que el mejor valor de α para este algoritmo en este grupo de instancias es el de 0.25.

A.5.2. Resultados completos en el grupo de instancias grandes

La Tabla A.10 muestra el *RPD* promedio para los tres valores de α probados en el algoritmo GRASP 5 en las instancias grandes. En la tabla se observa que el mejor valor de α para este algoritmo en este grupo de instancias es el de 0.25.

Tamaño	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$
50x10	2.87	8.78	19.81
50x15	1.33	9.65	25.15
50x20	0.64	14.34	28.14
50x25	0.00	18.66	34.77
50x30	3.57	17.89	40.12
100x10	0.55	9.47	25.10
100x15	0.21	11.18	26.71
100x20	0.00	13.88	29.33
100x25	0.18	13.53	30.69
100x30	0.72	13.74	34.33
150x10	0.23	7.21	18.99
150x15	0.34	8.52	21.87
150x20	0.69	9.18	24.49
150x25	0.31	12.59	27.20
150x30	0.07	11.09	27.03
200x10	0.35	8.03	22.11
200x15	0.17	8.96	27.18
200x20	0.05	9.37	28.73
200x25	0.06	11.11	32.67
200x30	0.17	11.71	37.18
250x10	0.03	7.89	24.13
250x15	0.11	9.32	31.28
250x20	0.11	11.20	38.97
250x25	0.08	12.32	42.76
250x30	0.21	13.38	51.41
Prom.	0.52	11.32	30.01

Tabla A.8: *RPD promedio para del algoritmo GRASP 4 para todos los valores de α en las instancias grandes.*

Tamaño	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$
6x2	3.24	2.05	2.29
6x3	2.69	3.42	4.74
6x4	2.29	5.37	9.20
6x5	2.30	3.48	4.58
8x2	1.36	2.08	3.36
8x3	4.98	4.99	8.77
8x4	3.88	6.77	10.48
8x5	6.12	3.53	4.06
10x2	2.90	4.27	4.79
10x3	3.92	4.62	7.63
10x4	2.73	6.60	12.25
10x5	6.31	8.34	11.47
12x2	3.27	4.46	5.37
12x3	4.04	8.05	11.05
12x4	6.27	11.90	13.06
12x5	5.96	16.00	15.58
Prom.	3.89	6.00	8.04

Tabla A.9: *RPD promedio para del algoritmo GRASP 5 para todos los valores de α en las instancias pequeñas.*

Tamaño	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$
50x10	11.63	10.58	21.61
50x15	7.39	10.73	26.23
50x20	7.46	22.34	36.15
50x25	5.56	18.88	34.99
50x30	14.88	22.31	44.54
100x10	1.89	12.33	27.97
100x15	7.90	14.09	29.62
100x20	3.62	14.16	29.61
100x25	3.62	23.72	40.89
100x30	4.45	18.95	39.54
150x10	9.82	11.75	23.53
150x15	5.27	10.94	24.29
150x20	15.93	14.71	30.02
150x25	11.00	24.41	39.03
150x30	0.09	18.79	34.73
200x10	14.86	16.88	30.97
200x15	9.96	20.09	38.31
200x20	13.93	17.83	37.19
200x25	2.10	11.22	32.77
200x30	15.77	20.32	45.79
250x10	2.44	16.50	32.74
250x15	12.52	16.26	38.22
250x20	2.60	18.23	46.00
250x25	14.72	21.70	52.14
250x30	13.49	15.88	53.90
Prom.	8.52	16.94	35.63

Tabla A.10: *RPD promedio para del algoritmo GRASP 5 para todos los valores de α en las instancias grandes.*

Apéndice B

Resultados completos ANOVA T-RIPG

En este apéndice mostramos los resultados completos del ANOVA realizado para calibrar el algoritmo T-RIPG del Capítulo 5. Las Tablas B.1 y B.2 muestran los resultados del ANOVA para los indicadores de hipervolumen y épsilon unario. En un ANOVA se calcula la razón-F para cada factor o interacción a estudiar. A partir de la razón-F, se calcula el valor-P que indica si el factor estudiado es significativo o no. Un valor-P menor que 0.05 indica que el factor es significativo, es decir, existen diferencias significativas entre los diferentes niveles que toma el factor estudiado. Por otro lado, un valor-P mayor que 0.05 indica que no hay diferencias entre los diferentes niveles del factor. Observando el valor-p en las tablas, nos damos cuenta que todos los parámetros son significativos en ambos indicadores. También podemos observar que las interacciones entre los parámetros p y q y los parámetros ℓ y q son significativas en ambos indicadores, mientras que la interacción triple entre p, ℓ y q, es significativa únicamente en el indicador de épsilon unario. Además, la interacción doble entre p y q también es significativa únicamente en el indicador de épsilon unario. El resto de interacciones no son significativas en ningún indicador.

Fuente	Suma de Cuadrados	GL	Cuadrado Medio	Razón-F	Valor-P
A:d	0.308815	3	0.102938	25.73	**0**
B:p	14.8007	4	3.70018	924.86	**0**
C:ℓ	0.994743	3	0.331581	82.88	**0**
D:q	8.39688	3	2.79896	699.6	**0**
Interacciones					
AB	0.0577548	12	0.0048129	1.2	0.2738
AC	0.022332	9	0.00248134	0.62	0.7809
AD	0.00837313	9	0.00093035	0.23	0.9899
BC	0.0226653	12	0.00188877	0.47	0.932
BD	0.120354	12	0.0100295	2.51	**0.0027**
CD	0.214782	9	0.0238647	5.96	**0**
ABC	0.0291631	36	0.00081009	0.2	1
ABD	0.0150786	36	0.00041885	0.1	1
ACD	0.00848446	27	0.00031424	0.08	1
BCD	0.169537	36	0.00470937	1.18	0.2152
Residuo	255.204	63788	0.00400082		
Total	280.374	63999			

Tabla B.1: *Tabla ANOVA para el indicador de hipervolumen en la calibración del T-RIPG.*

Fuente	Suma de Cuadrados	GL	Cuadrado Medio	Razón-F	Valor-P
A:d	0.0158047	3	0.00526824	10.59	**0**
B:p	2.37942	4	0.594854	1196.2	**0**
C:ℓ	0.117373	3	0.0391242	78.68	**0**
D:q	1.90467	3	0.634892	1276.71	**0**
Interacciones					
AB	0.00576095	12	0.00048008	0.97	0.4796
AC	0.00497368	9	0.00055263	1.11	0.3504
AD	0.00201489	9	0.00022388	0.45	0.908
BC	0.013865	12	0.00115541	2.32	**0.0058**
BD	0.0439826	12	0.00366522	7.37	**0**
CD	0.161944	9	0.0179938	36.18	**0**
ABC	0.00831645	36	0.00023101	0.46	0.9975
ABD	0.00739198	36	0.00020533	0.41	0.9993
ACD	0.0054936	27	0.00020347	0.41	0.9971
BCD	0.10539	36	0.0029275	5.89	**0**
Residuo	31.721	63788	0.00049729		
Total	36.4974	63999			

Tabla B.2: *Tabla ANOVA para el indicador de épsilon unario en la calibración del T-RIPG.*

Apéndice C

Calibración de los algoritmos multi-objetivo adaptados de la literatura

Para hacer una comparación justa de los algoritmos, los parámetros de los tres algoritmos adaptados de la literatura también fueron calibrados en las 200 instancias de calibración generadas. En el resto del apéndice se detallan tales resultados para cada algoritmo.

C.1. Calibración NSGA-II

Los parámetros a calibrar del algoritmo NSGA-II son los siguientes:

- El parámetro s que es el tamaño de la población inicial generada antes del primer cruce. El parámetro s también denota el número de soluciones generadas cuando se aplica el operador de cruce, así como el número de soluciones que se eligen para pasar a la siguiente iteración. Probamos este parámetro con tres niveles: $s = 20$, $s = 30$ y $s = 50$

- El parámetro p que es la probabilidad con la que una solución es reparada en la fase de temporalidad. Probamos este parámetro con cinco niveles: $p = 0$, $p = 0.25$, $p = 0.5$, $p = 0.75$ y $p = 1$. Al igual que en la calibración del algoritmo T-RIPG, cuando $p = 0$ quiere decir que ninguna solución pasa por la fase de temporalidad, mientras que $p = 1$ quiere decir que todas las soluciones son reparadas en esta fase.

- El parámetro c que es el número de trabajos copiados del padre 1 al primer descendiente en cada máquina. Se copia desde la posición 1 hasta la posición c de cada máquina. Probamos este parámetro con dos niveles: c elegido aleatoriamente en cada máquina, y c =número de trabajos en la máquina i dividido por 2.

Para comparar estadísticamente las diferentes configuraciones del algoritmo, hacemos un análisis de varianza (ANOVA) teniendo como variables respuesta los indicadores de hipervolumen y épsilon unario. Las Tablas C.1 y C.2 muestran los resultados completos del ANOVA para los dos indicadores. En esas tablas podemos observar que todos los parámetros calibrados son significativos en ambos indicadores. Además, se observa que la interacción entre p y c es significativa en el indicador de hipervolumen. El resto de interacciones no son significativas en ninguno de los dos indicadores.

Fuente	Suma de Cuadrados	GL	Cuadrado Medio	Razón-F	Valor-P
A:s	1.07823	2	0.539117	58.07	**0**
B:p	14.0856	4	3.5214	37.28	**0**
C:c	0.811175	1	0.811175	87.37	**0**
Interacciones					
AB	0.0254387	8	0.00317983	0.34	0.9496
AC	0.00380937	2	0.00190468	0.21	0.8145
BC	0.128743	4	0.0321858	3.47	**0.0078**
ABC	0.0194765	8	0.00243456	0.26	0.9778
Residuo	55.4283	5970	0.00928447		
Total	71.5808	5999			

Tabla C.1: *Tabla ANOVA para el indicador de hipervolumen en la calibración del NSGA-II.*

Las Figuras C.1 y C.2 muestran los intervalos HSD de Tukey al 95 % de confianza para los indicadores de hipervolumen y épsilon unario para los tres factores estudiados. Observando los resultados, concluímos que los mejores valores para cada parámetro son los siguientes: $p = 1$, $s = 50$ y $c =$ aleatorio

C.2. Calibración MOIGS

Los parámetros a calibrar del algoritmo MOIGS son los siguientes:

Fuente	Suma de Cuadrados	GL	Cuadrado Medio	Razón-F	Valor-P
A:s	0.181821	2	0.0909104	30.36	**0**
B:p	7.76223	4	1.94056	648.14	**0**
C:c	0.0308547	1	0.0308547	10.31	**0.0013**
Interacciones					
AB	0.0112795	8	0.00140994	0.47	0.8774
AC	0.00352535	2	0.00176268	0.59	0.5551
BC	0.0137333	4	0.00343333	1.15	0.3325
ABC	0.00663873	8	0.00082984	0.28	0.9736
Residuo	17.8745	5970	0.00299406		
Total	25.8846	5999			

Tabla C.2: *Tabla ANOVA para el indicador de épsilon unario en la calibración del NSGA-II.*

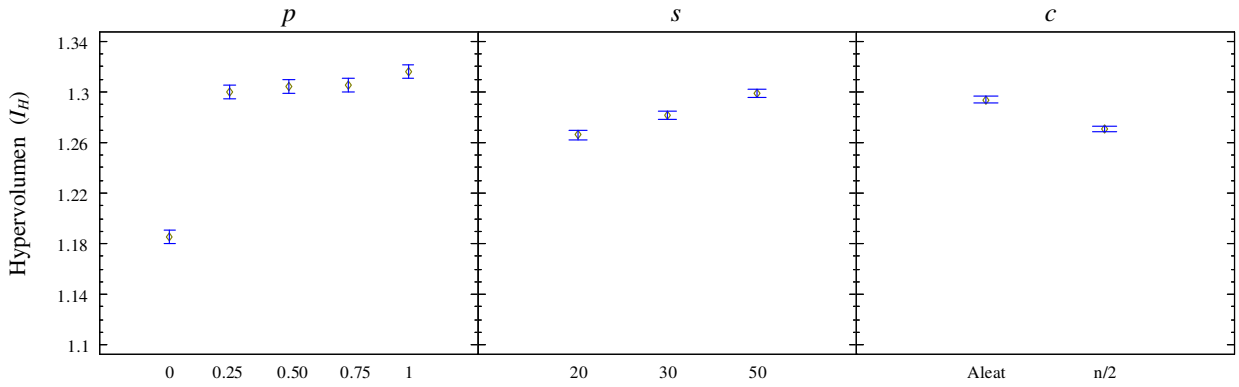

Figura C.1: *Intervalos HSD de Tukey para el indicador de hipervolumen (I_H) para todos los factores calibrados del NSGA-II.*

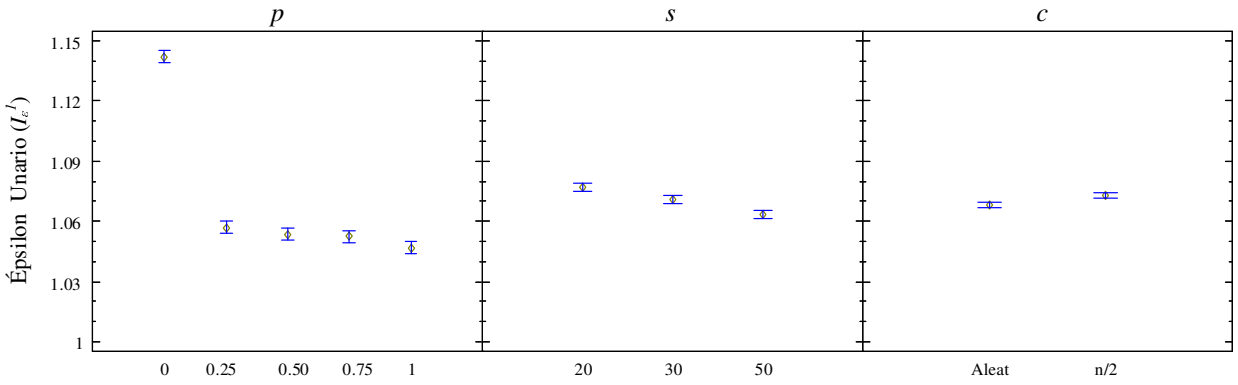

Figura C.2: *Intervalos HSD de Tukey para el indicador épsilon unario (I_ϵ^1) para todos los factores calibrados del NSGA-II.*

- El parámetro d que es el número de trabajos que se quitan en la fase voraz. Probamos cuatro niveles para este parámetro: $d = 0.05 \times n$, $d = 0.1 \times n$, $d = 4$ y $d = 5$.

- El parámetro p que es la probabilidad con la que una solución puede ser reparada en la fase de temporalidad. Probamos este parámetro con cinco niveles: $p = 0$, $p = 0.25$, $p = 0.5$, $p = 0.75$ y $p = 1$.

Hacemos el ANOVA para comparar estadísticamente las diferentes configuraciones del algoritmo, teniendo como variables respuesta los indicadores de hipervolumen y épsilon unario.

Las Tablas C.3 y C.4 muestran los resultados completos del ANOVA para los dos indicadores. En esas tablas podemos observar que los dos parámetros calibrados son significativos en ambos indicadores. En esas tablas también podemos observar que la interacción entre los dos parámetros no es significativa en ninguno de los dos indicadores.

Fuente	Suma de Cuadrados	GL	Cuadrado Medio	Razón-F	Valor-P
A:d	0.119973	3	0.0399909	7.66	**0**
B:p	1.96733	4	0.491832	94.22	**0**
Interacciones					
AB	0.00887529	12	0.00073961	0.14	0.9997
Residuo	20.7767	3980	0.00522028		
Total	22.8729	3999			

Tabla C.3: *Tabla ANOVA para el indicador de hipervolumen en la calibración del MOIGS.*

Fuente	Suma de Cuadrados	GL	Cuadrado Medio	Razón-F	Valor-P
A:d	0.0507211	3	0.016907	19.1	**0**
B:p	0.49795	4	0.124488	140.65	**0**
Interacciones					
AB	0.00426776	12	0.00035565	0.4	0.9636
Residuo	3.52267	3980	0.00088509		
Total	4.07561	3999			

Tabla C.4: *Tabla ANOVA para el indicador de épsilon unario en la calibración del MOIGS.*

Las Figuras C.3 y C.4 muestran los intervalos HSD de Tukey al 95 % de confianza para los indicadores de hipervolumen y épsilon unario para los dos factores estudiados. Observando

los resultados, concluimos que los mejores valores para cada parámetro son los siguientes: $d = 4$, y $p = 1$.

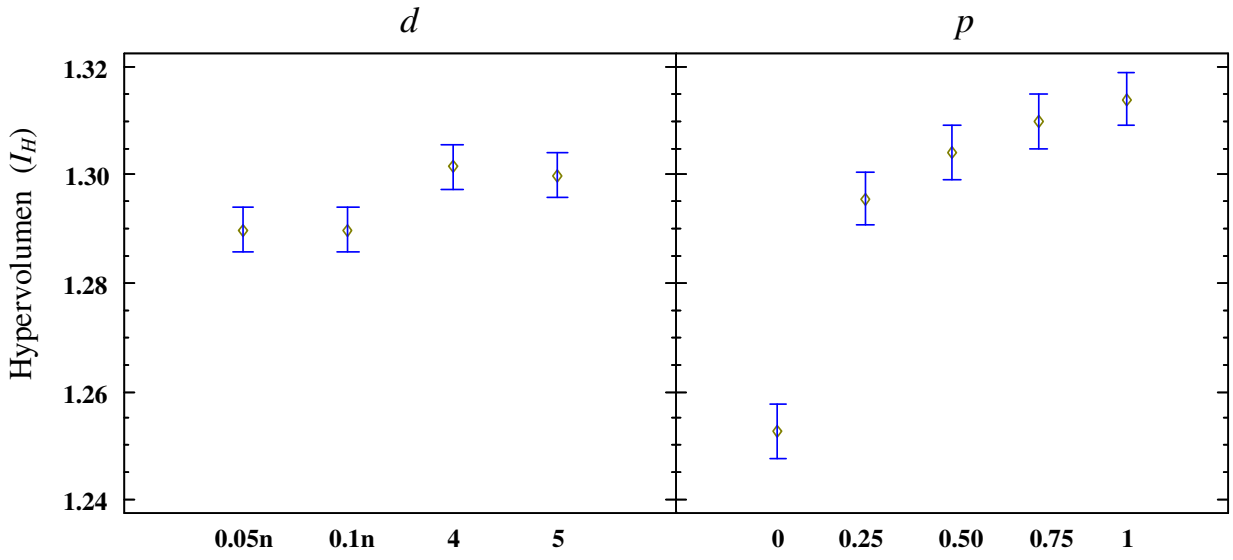

Figura C.3: *Intervalos HSD de Tukey para el indicador de hipervolumen (I_H) para todos los factores calibrados del MOIGS.*

C.3. Calibración RIPG

Finalmente, el algoritmo RIPG tiene los mismos parámetros a calibrar que el algoritmo T-RIPG explicado en la Sección 5.3.1. Al igual que con todos los demás algoritmos, hacemos el ANOVA para comparar estadísticamente las diferentes configuraciones del algoritmo, teniendo como variables respuesta los indicadores de hipervolumen y épsilon unario.

Las Tablas C.1 y C.2 muestran los resultados completos del ANOVA para los cuatro indicadores. En esas tablas podemos observar que todos los parámetros calibrados son significativos en ambos indicadores. Además, podemos observar que las interacciones dobles entre p y q y entre ℓ y q son significativas en ambos indicadores. Del mismo modo, la interacción triple entre p, ℓ y q es significativa en los dos indicadores. Por otro lado, la interacción entre p y ℓ es significativa solo en el indicador de épsilon unario.

Las Figuras C.5 y C.6 muestran los intervalos HSD de Tukey al 95 % de confianza

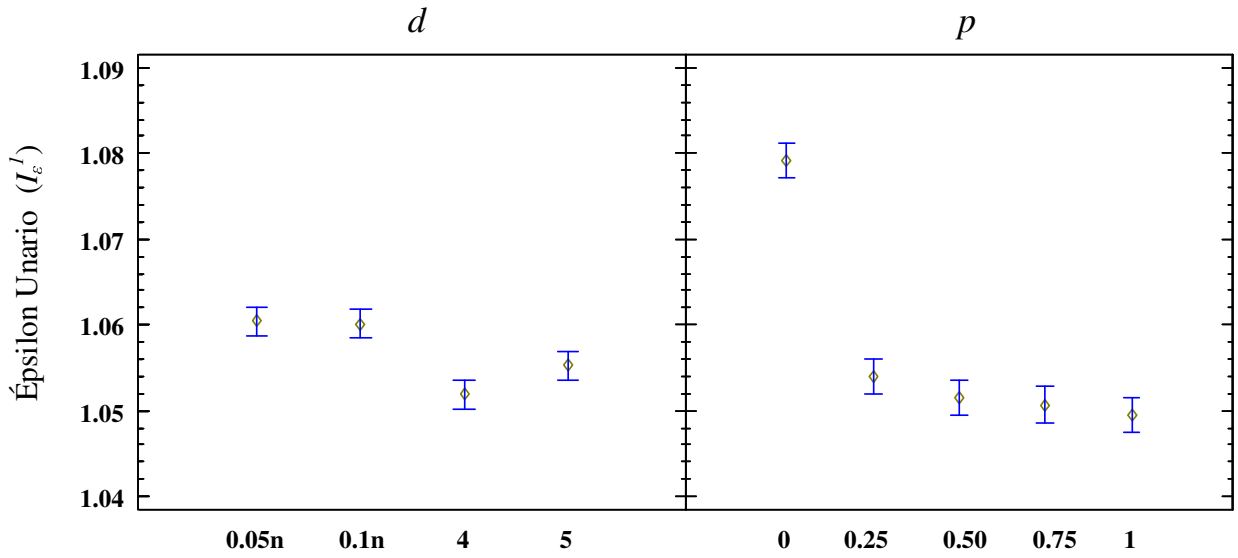

Figura C.4: *Intervalos HSD de Tukey para el indicador épsilon unario (I_ϵ^1) para todos los factores calibrados del MOIGS.*

Fuente	Suma de Cuadrados	GL	Cuadrado Medio	Razón-F	Valor-P
A:d	0.163478	3	0.0544926	12.86	**0**
B:p	8.57965	4	2.14491	506.36	**0**
C:ℓ	0.701311	2	0.350655	82.78	**0**
D:q	3.5824	2	1.7912	422.85	**0**
Interacciones					
AB	0.0308065	12	0.00256721	0.61	0.8391
AC	0.0108271	6	0.00180452	0.43	0.8621
AD	0.0114595	6	0.00190992	0.45	0.8448
BC	0.0240497	8	0.00300622	0.71	0.6833
BD	0.135507	8	0.0169384	4	**0.0001**
CD	0.131432	4	0.0328581	7.76	**0**
ABC	0.0178121	24	0.00074217	0.18	1
ABD	0.0109823	24	0.0004576	0.11	1
ACD	0.00418889	12	0.00034907	0.08	1
BCD	0.124748	16	0.00779674	1.84	**0.0211**
Residuo	151.936	35868	0.00423597		
Total	165.464	35999			

Tabla C.5: *Tabla ANOVA para el indicador de hipervolumen en la calibración del RIPG.*

Fuente	Suma de Cuadrados	GL	Cuadrado Medio	Razón-F	Valor-P
A:d	0.00711405	3	0.00237135	4.45	**0.004**
B:p	1.38672	4	0.346679	649.95	**0**
C:ℓ	0.086443	2	0.0432215	81.03	**0**
D:q	0.704048	2	0.352024	659.97	**0**
Interacciones					
AB	0.00151454	12	0.00012621	0.24	0.9966
AC	0.00287753	6	0.00047959	0.9	0.4943
AD	0.00144433	6	0.00024072	0.45	0.8445
BC	0.0157404	8	0.00196755	3.69	**0.0003**
BD	0.0458105	8	0.00572631	10.74	**0**
CD	0.0744739	4	0.0186185	34.91	**0**
ABC	0.0044467	24	0.00018528	0.35	0.9987
ABD	0.00361063	24	0.00015044	0.28	0.9998
ACD	0.00416487	12	0.00034707	0.65	0.7999
BCD	0.0728043	16	0.00455027	8.53	**0**
Residuo	19.1318	35868	0.0005334		
Total	21.543	35999			

Tabla C.6: *Tabla ANOVA para el indicador de épsilon unario en la calibración del RIPG.*

para los indicadores de hipervolumen y épsilon unario para los cuatro factores estudiados. Observando los resultados, concluimos que los mejores valores para cada parámetro son los siguientes: $d = 4$, $p = 1$, $\ell = n$ y $q = 50$.

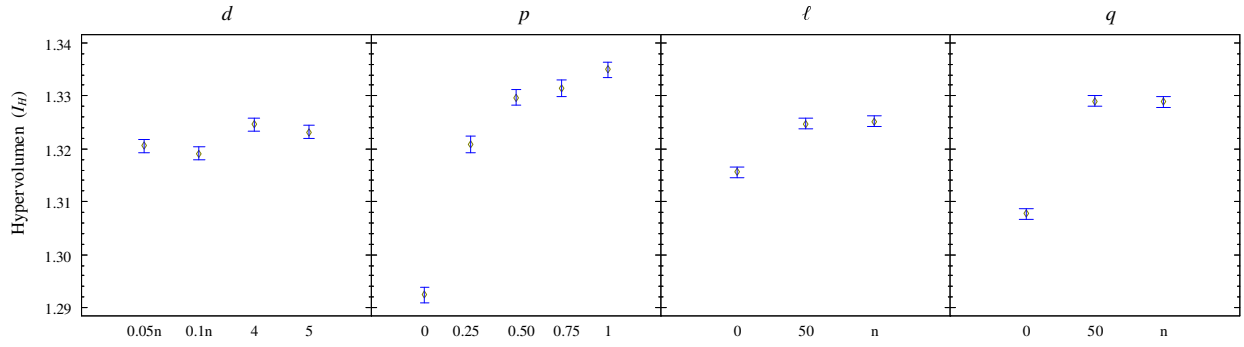

Figura C.5: *Intervalos HSD de Tukey para el indicador de hipervolumen (I_H) para todos los factores calibrados del RIPG.*

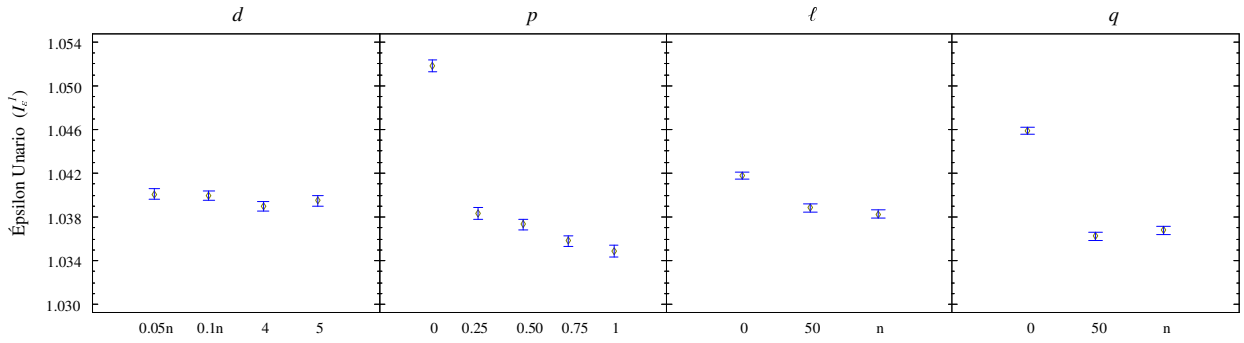

Figura C.6: *Intervalos HSD de Tukey para el indicador épsilon unario (I_ϵ^1) para todos los factores calibrados del RIPG.*

Apéndice D

Problema de secuenciación de máquinas paralelas con tiempos de ajuste estocásticos y necesidad de recursos en los ajustes

Una variación al problema estudiado a lo largo de esta tesis doctoral surge al considerar los tiempos de ajuste de las máquinas como estocásticos en lugar de asumirlos deterministas. Al igual que en el problema UPMSR-S explicado en el Capítulo 4, el objetivo en este problema es el de minimizar el *makespan*. Esta es una variación realista del problema, ya que puede modelar un sistema en el que, al ser realizados por recursos adicionales, los ajustes en las máquinas pueden estar sujetos a variaciones en sus tiempos.

Para este problema, estamos trabajado con el enfoque planteado en Juan et al. (2014) y en Juan et al. (2015). En este primer acercamiento al problema, consideramos que los tiempos de ajuste se distribuyen con una distribución de probabilidad con dominio positivo, es decir, $x \in (0, +\infty)$. En este caso, al igual que en Juan et al. (2014), estamos utilizando una distribución Log-Normal, pero se podría utilizar otra distribución de probabilidad no negativa, como la distribución Weibull, exponencial, Gamma, etc.

Cada uno de los tiempos de ajuste de las matrices s_{ijk} (explicadas en la Sección 3.4) sigue una distribución Log-Normal con media μ_{ijk} y desviación típica σ_{ijk}. La idea principal es resolver el problema de forma determinista en una instancia, donde los tiempos de ajuste son

iguales a la media de la distribución Log-Normal que sigue cada uno de estos tiempos (μ_{ijk}). Una vez resuelta esta instancia, se toma la solución obtenida y se hacen s simulaciones con los tiempos estocásticos. Después de este proceso, se obtienen s soluciones, con las cuales se calculará el *makespan* promedio de la solución simulando un entorno variable (*makespan* estocástico). Además, se pueden evaluar otros estadísticos interesantes como la varianza del *makespan*.

En la Figura D.1 se muestra el procedimiento general que estamos utilizando para resolver el problema. Este procedimiento está basado en el utilizado en Juan et al. (2014) para resolver un problema de taller de flujo, donde los tiempos de proceso son estocásticos. Como se observa en la figura, inicialmente se resuelve el problema determinista para la instancia con los tiempos de ajuste iguales a μ_{ijk}. En este caso, estamos utilizando el algoritmo GRASP 4 del Capítulo 4 que fue el que mejores resultados dio para el problema determinista. Una vez resuelto el problema determinista, se aplica una simulación "pequeña" para calcular el *makespan* estocástico de la mejor solución determinista. Posteriormente, aplicamos una perturbación a la solución buscando mejorarla, si la nueva solución es mejor que la anterior en el problema determinista, se actualiza la mejor solución determinista y se repite el proceso de la simulación "pequeña" con esta nueva solución. Por otro lado, si la nueva solución no es mejor que la mejor solución determinista, se evalúa si pasa un criterio de aceptación. Este criterio de aceptación consiste en que el *makespan* de la nueva solución sea menor al *makespan* de la mejor solución determinista más un valor X a calibrar. Si la nueva solución pasa este criterio, esta sería la nueva solución con la que se trabajaría y se repite la simulación "pequeña".

Cada vez que se realiza una simulación, se evalúa el *makespan* estocástico de la nueva solución. Si este *makespan* estocástico es mejor que el mejor *makespan* estocástico conocido, se actualiza la mejor solución estocástica conocida. Este proceso se repite hasta que se cumple el tiempo de ejecución previsto para la instancia ejecutada. Finalmente, al terminar el tiempo de ejecución, se realiza una simulación "grande" con la mejor solución estocástica

y la mejor solución determinista que resultaron al final de la ejecución del algoritmo. Con los resultados de estas simulaciones, se calcula el *makespan* estocástico y la varianza del *makespan* estocástico de ambas soluciones. Actualmente, estamos trabajando en la calibración del algoritmo y los experimentos computacionales.

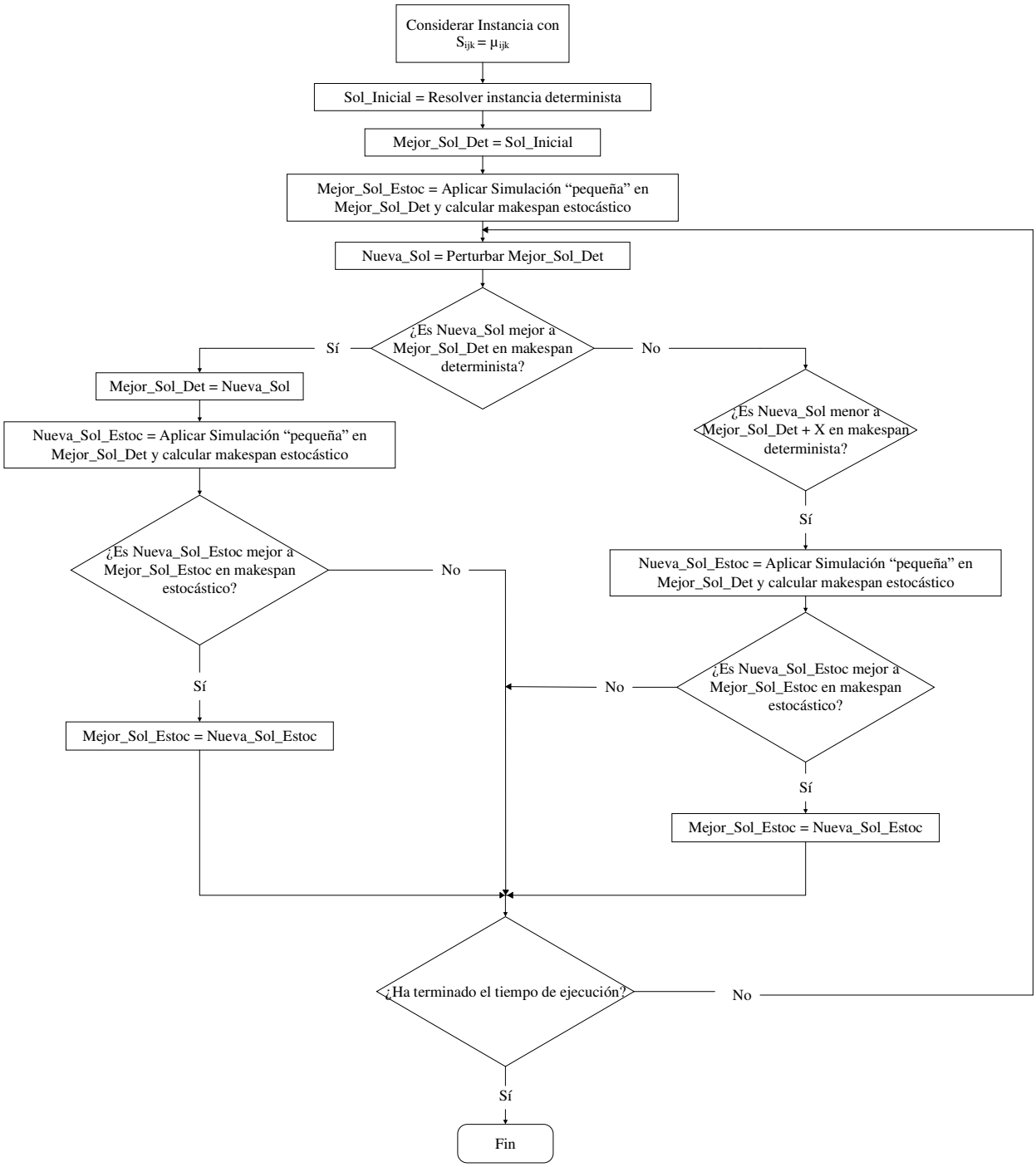

Figura D.1: *Procedimiento general de la simheurística planteada.*