UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Inteligencia computacional en la programación de la producción con recursos adicionales

Doctorado en Estadística y Optimización

Autor

Pedro Alfaro Fernández

Directoras

Eva Vallada Regalado
María Fulgencia Villa Juliá

Valencia, junio 2023

A mi madre

A mi hermana

A Jorge, Rodrigo y Ariadna.

Resumen

En esta Tesis Doctoral se aborda el problema del taller de flujo de permutación considerando recursos adicionales renovables, que es una versión más realista del clásico problema de taller de flujo de permutación, muy estudiado en la literatura. La inclusión de los recursos ayuda a acercar el mundo académico-científico al mundo real de la industria. Se ha realizado una completa revisión bibliográfica que no se ha limitado a problemas del taller de flujo, sino que han revisado problemas similares del ámbito de scheduling que consideren recursos. En esta revisión, no se han encontrado en la literatura artículos para el problema concreto que se estudia en esta tesis. Por ello, la aportación principal de esta Tesis Doctoral es el estudio por primera vez de este problema y la propuesta y adaptación de métodos para su resolución. Inicialmente, el problema se modeliza a través de un modelo de programación lineal entera mixta (MILP). Dada la complejidad del problema, el MILP es capaz de resolver instancias de un tamaño muy pequeño. Por ello, es necesario adaptar, diseñar e implementar heurísticas constructivas y metaheurísticas para obtener buenas soluciones en un tiempo de computación razonable. Para evaluar la eficacia y eficiencia de los métodos propuestos, se generan instancias de problemas partiendo de los conjuntos más utilizados en la literatura para el taller de flujo de permutación. Se utilizan estas instancias propuestas tanto para calibrar los distintos métodos como para evaluar su rendimiento a través de experimentos computacionales masivos. Los experimentos muestran que las heurísticas propuestas son métodos sencillos que consiguen soluciones factibles de una forma muy rápida. Para mejorar las soluciones obtenidas con las heurísticas y facilitar el movimiento a otros espacios de soluciones, se proponen tres metaheurísticas: un método basado en búsqueda local iterativa (ILS), un método voraz iterativo (IG) y un algoritmo genético con búsqueda local (HGA). Todos ellos utilizan las heurísticas propuestas más eficaces como solución o soluciones iniciales. Las metaheurísticas obtienen las mejores soluciones utilizando tiempos de computación razonables, incluso para las instancias de mayor tamaño. Todos los métodos han sido implementados dentro de la plataforma FACOP (Framework for Applied Combinatorial Optimization Problems). Dicha plataforma es capaz de incorporar nuevos algoritmos de optimización para problemas de investigación operativa relacionados con la toma de decisiones de las organizaciones y está diseñada para abordar casos reales en empresas. El incorporar en esta plataforma todas las metodologías propuestas en esta Tesis Doctoral, acerca el mundo académico al mundo empresarial.

Abstract

In this Doctoral Thesis, the permutation flowshop problem is addressed considering additional renewable resources, which is a more realistic version of the classic permutation flowshop problem, widely studied in the literature. The inclusion of resources helps to bring the academic-scientific world closer to the real world of industry. A complete bibliographic review has been carried out that has not been limited to flow shop problems, but has reviewed similar problems in the scheduling field that consider resources. In this review, no articles have been found in the literature for the specific problem studied in this thesis. Therefore, the main contribution of this Doctoral Thesis is the study for the first time of this problem and the proposal and adaptation of methods for its resolution. Initially, the problem is modeled through a mixed integer linear programming (MILP) model. Given the complexity of the problem, the MILP is capable of solving very small instances. Therefore, it is necessary to adapt, design and implement constructive heuristics and metaheuristics to obtain good solutions in a reasonable computation time. In order to evaluate the effectiveness and efficiency of the proposed methods, problem instances are generated starting from the sets most used in the literature for the permutation flowshop. These proposed instances are used both to calibrate the different methods and to evaluate their performance through massive computational experiments. Experiments show that proposed heuristics are simple methods that achieve feasible solutions very quickly. To improve the solutions obtained with the heuristics and facilitate movement to other solution spaces, three metaheuristics are proposed: a method based on iterated local search (ILS), an iterative greedy method (IG) and a hybrid genetic algorithm (HGA). All of them use the most effective proposed heuristics as initial solution or solutions. Metaheuristics get the best solutions using reasonable computation times, even for the largest instances. All the methods have been implemented within the FACOP platform (Framework for Applied Combinatorial Optimization Problems). Said platform is capable of incorporating new optimization algorithms for operational research problems related to decision-making in organizations and it is designed to address real cases in companies. Incorporating in this platform all the methodologies proposed in this Doctoral Thesis, brings the academic world closer to the business world.

Resum

En aquesta Tesi Doctoral s'aborda el problema del taller de flux de permutació considerant recursos addicionals renovables, que és una versió més realista del clàssic problema de taller de flux de permutació, molt estudiat a la literatura. La inclusió dels recursos ajuda a apropar el món acadèmic-científic al món real de la indústria. S'ha realitzat una revisió bibliogràfica completa que no s'ha limitat a problemes del taller de flux, sinó que ha revisat problemes similars de l'àmbit de scheduling que considerin recursos. En aquesta revisió, no s'ha trobat a la literatura articles per al problema concret que s'estudia en aquesta tesi. Per això, l'aportació principal d'aquesta Tesi Doctoral és l'estudi per primera vegada d'aquest problema i la proposta i l'adaptació de mètodes per resoldre'ls. Inicialment, el problema es modelitza mitjançant un model de programació lineal sencera mixta (MILP). Donada la complexitat del problema, el MILP és capaç de resoldre instàncies d'un tamany molt petita. Per això, cal adaptar, dissenyar i implementar heurístiques constructives i metaheurístiques per obtenir bones solucions en un temps de computació raonable. Per avaluar l'eficàcia i l'eficiència dels mètodes proposats, es generen instàncies de problemes partint dels conjunts més utilitzats a la literatura per al taller de flux de permutació. S'utilitzen aquestes instàncies proposades tant per calibrar els diferents mètodes com per avaluar-ne el rendiment a través d'experiments computacionals massius. Els experiments mostren que les heurístiques proposades són mètodes senzills que aconsegueixen solucions factibles de manera molt ràpida. Per millorar les solucions obtingudes amb les heurístiques i facilitar el moviment a altres espais de solucions, es proposen tres metaheurístiques: un mètode basat en cerca local iterativa (ILS), un mètode voraç iteratiu (IG) i un algorisme genètic híbrid (HGA). Tots ells utilitzen les heurístiques proposades més eficaces com a solució o solucions inicials. Les metaheurístiques obtenen les millors solucions utilitzant temps de computació raonables, fins i tot per a les instàncies més grans. Tots els mètodes han estat implementats dins de la plataforma FACOP (Framework for Applied Combinatorial Optimization Problems). Aquesta plataforma és capaç d'incorporar nous algorismes d'optimització per a problemes de recerca operativa relacionats amb la presa de decisions de les organitzacions i està dissenyada per abordar casos reals a empreses. El fet d'incorporar en aquesta plataforma totes les metodologies proposades en aquesta Tesi Doctoral, apropa el món acadèmic al món empresarial.

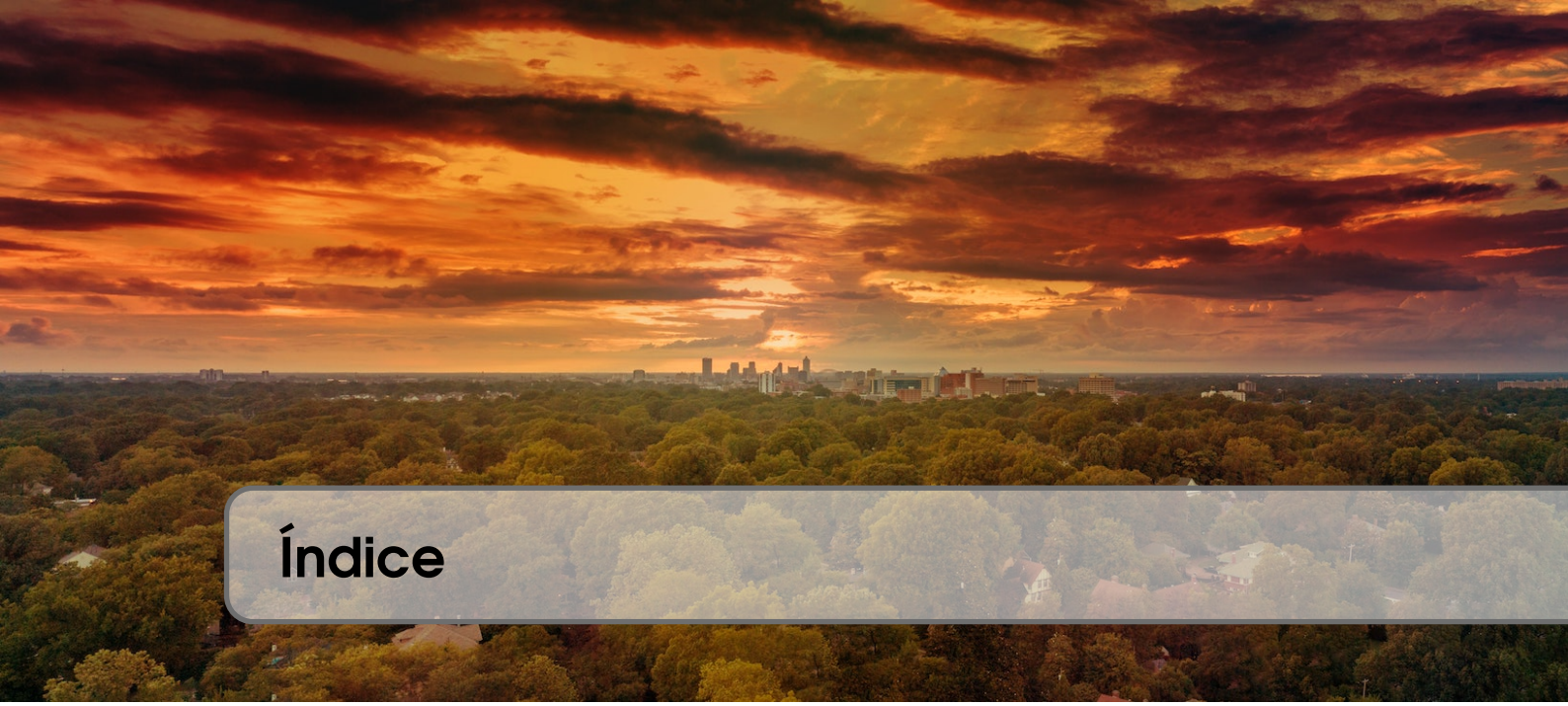
Agradecimientos

Son muchas las personas a las que quiero darle las gracias por el apoyo recibido a lo largo de estos años. Muchas personas han colaborado de forma directa o indirecta en el esfuerzo que finalmente ha culminado en esta tesis.

En primer lugar, quiero agradecer a mis directoras, Eva Vallada Regalado y María Fulgencia Villa, por su gran labor y enorme apoyo durante estos últimos años, y también porque no, agradecerles su paciencia conmigo. Agradecerle a Rubén todo el trabajo realizado conmigo, gracias a ti he podido aprender mucho desde que comencé la andadura de esta tesis con él. También agradecerle al doctor Thomas Stützle su amabilidad y ayuda, así como a Federico Pagnozzi con quienes ha sido un placer trabajar.

Agradecerles a mis compañeros del equipo SOA y de SOIS por su compañerismo y amabilidad. Por darme un entorno colaborativo genial a pesar de tantas discusiones de paellas, tiburones o frutas... Especialmente agradecerle a Gerardo Minella todo el esfuerzo que ha puesto en nuestro trabajo estos últimos años en los que he tenido el privilegio de aprender mucho de él. Finalmente, a mi familia y amigos, especialmente a mi madre y mi hermana que siempre han estado ahí. A mis sobrinos que me traen la felicidad cada vez que los veo.

Muchas gracias a todos.



Índice

Índice de Figuras

Índice de Tablas

1. Introducción

En el actual contexto global, con ritmo acelerado de cambios en sociedades y empresas, estas últimas se han visto abocadas a grandes adaptaciones en los paradigmas empresariales. La tecnología digital, el desarrollo de capacidades computacionales cada vez más avanzadas, la personalización de los productos y una especialización cada vez más enfocada del trabajo, son solo algunos de la miríada de factores que empuja a las empresas a realizar una inversión constante en sistemas de producción que sean más eficientes, inmediatos y ágiles para mantenerse a la vanguardia y así poder competir en un mercado global que cada vez es más competitivo. En este sentido, una empresa se enfrenta a múltiples problemas en la toma de decisiones y en el ámbito de producción destacan dos retos críticos: la planificación y la programación de la producción.

La planificación de la producción es un proceso que determina de forma estratégica e informada, la producción futura, con el objetivo de optimizar las decisiones para balancear entre los objetivos económicos y otros no monetarios como la satisfacción del cliente (Pochet y Wolsey 2006). El producto final de este proceso es un plan de producción que establece: qué se quiere producir en el medio plazo, qué recursos serán necesarios y qué acciones llevarán a hacer posible esta producción de forma eficiente y/o económica. Un plan de producción equivocado puede generar interrupciones en la producción debido a la falta de materias primas, costes adicionales de almacenamiento y escasez de personal, entre otros factores. El siguiente paso en la toma de decisiones será la creación de un itinerario de producción futura con la programación de la producción.

1.1 Programación de la producción

La programación de la producción, conocida en inglés por *production scheduling* o *scheduling*, es el proceso crucial de la toma de decisiones que organiza la utilización de los recursos productivos a lo largo del tiempo para llevar a término la producción establecida en el plan de producción de la mejor manera posible, de acuerdo a los criterios establecidos (Graves 1981).

Scheduling se puede concebir como el conjunto de las decisiones tácticas a corto plazo, derivadas de unas decisiones estratégicas a medio plazo previamente establecidas a través del plan de producción. Estas decisiones se concretan en una programación de la producción que cronológicamente distribuye los recursos productivos en relación a los requisitos establecidos, presentada en un *programa* o *schedule*. El estudio de la programación de la producción ha experimentado un aumento de relevancia en el ámbito académico e industrial en las últimas décadas.

La investigación de operaciones, o investigación operativa (IO), es la disciplina que estudia la

aplicación de métodos analíticos avanzados asistir en la mejora de la toma de decisiones, se puede ver como una aproximación científica para resolver problemas ejecutivos de toma de decisiones en la gestión. Tiene su origen en la gestión de operaciones militares, pero actualmente se aplica a gestión de empresas, gobiernos, salud, etc. Entre los pasos comunes que se utilizan están: identificar el problema, generar un modelo del problema del mundo real, utilizar el modelo para encontrar soluciones al problema, comprobar estas soluciones y analizar su resultado, implementar la solución.

Una de muchas formas de aplicar investigación operativa a las decisiones en un entorno productivo es *scheduling*, donde el problema identificado es la planificación de la producción en un horizonte temporal. Por lo tanto, se "modeliza" la realidad en un problema concreto y discreto de investigación operativa. En este proceso de modelización se han de establecer los tiempos y las restricciones del sistema. Además, es crucial identificar correctamente cuales son los recursos clave y cuales no, ya que en situaciones de producción reales existen una gran cantidad de factores que deben ser considerados. Por lo tanto, es necesario encontrar un equilibrio adecuado en la modelización y determinar las restricciones más críticas. Por un lado, una modelización excesivamente simple puede generar soluciones irreales o incluso inviables, mientras que una modelización que tenga en cuenta un excesivo número de factores puede aumentar la complejidad de la solución del problema, haciéndolo casi intratable e incluyendo factores prácticamente irrelevantes para la toma de decisiones finales.

1.2 Motivación

Aunque en la actualidad la programación de la producción se considera crucial, rara vez se estudia teniendo en cuenta toda su amplitud. Por una parte, las empresas pequeñas no suelen disponer de personal o herramientas adecuadas y las empresas grandes suelen tener sistemas muy rígidos. Como consecuencia muchas veces las aproximaciones utilizadas son casi tradicionales, pues operan como solían hacerlo en ocasiones anteriores, confiando en los conocimientos de un experto que decide como se realiza la programación de la producción, y lo hace basándose en su experiencia particular y el conocimiento adquirido en el día a día. Esta metodología no es reactiva a los cambios, la calidad de los resultados obtenidos suele ser pobre y es propensa a producir errores humanos. Por otra parte, en el mundo académico muchos trabajos se centran en problemas teóricos, de pequeñas dimensiones o extremadamente simplificados. De estos, es más difícil extraer conocimiento práctico, pues son problemas muy alejados de la realidad industrial. La distancia que separa la investigación científica de la realidad industrial es lo que se conoce como *GAP* (J. M. Framinan, Leisten y Ruiz García 2014). Actualmente, se dedican muchos esfuerzos por parte de investigadores y empresas en acortar esta brecha, los primeros trabajando en problemas más realistas que se ajusten mejor a los problemas de las empresas, mientras que estas dedican cada vez más recursos a resolver estos problemas con software especializado.

En este marco de la investigación académica actual, en esta tesis se busca la resolución de problemas más realistas de *scheduling*, que nunca o rara vez han sido considerados, se ha buscado la intersección entre talleres más realistas, problemas con tamaños más ajustados a la realidad con recursos adicionales. Por lo tanto, en esta tesis se ha elegido enfrentar el problema del taller de flujo con permutación y recursos adicionales renovables, un problema que no ha sido estudiado hasta el momento y se optimizará intentando minimizar el tiempo de finalización máximo. Simultáneamente, en el esfuerzo de generar soluciones más eficientes, se ha trabajado en el diseño e implantación de software especializado para la experimentación a gran escala, la reutilización de código y la modularización algorítmica.

1.3 Objetivos

Esta tesis doctoral se plantea con el propósito de acortar la brecha entre ciencia y empresa, planteando y resolviendo problemáticas más cercanas a la realidad. El objetivo principal es proporcionar soluciones competitivas para el problema del taller de flujo con permutación y recursos adicionales renovables, con la finalidad de acabar todos los procesos del taller lo antes posible.

Existen múltiples formas de modelar los recursos adicionales ya que estos pueden representar diferentes bienes o activos. No se deben modelar igual casos tan diferentes como: tipo de personal, energía disponible, puntos de montaje, computadoras, etc. Por ello, se comenzará por definir, explicar y clasificar los diferentes tipos de los recursos adicionales, con especial énfasis en los recursos adicionales renovables. Este tipo de recursos modelan casos tan comunes como personal o energía disponible entre otros, a pesar de ser tan habituales en el entorno productivo, rara vez se estudian en problemas realistas o de gran tamaño.

Con la intención de abordar este problema de manera eficiente, primero se estudiarán las diferentes propuestas que se han planteado en los últimos años para problemas similares y se propondrán métodos algorítmicos de resolución que han probado ser competitivos en problemas parecidos pero que nunca han sido adaptados ni estudiados en este caso. Para mejorar la eficiencia se propondrán soluciones algorítmicas simples y heurísticas. Asimismo, para mejorar la eficacia, se presentarán metaheurísticas, métodos más complejos que requerirán un proceso de calibración previo. Por último, se evaluará la eficacia y eficiencia de los métodos propuestos a través de una extensa experimentación computacional.

Como objetivos secundarios de esta tesis se incluyen:

- El desarrollo de software especializado para el diseño e implementación de algoritmos de optimización.
- Creación de herramientas de asistencia que permitan simplificar la experimentación a gran escala, faciliten la reutilización de código y la mejora en la modularidad algorítmica.
- A través de los dos puntos anteriores facilitar el estudio de otros problemas realistas.

1.4 Esquema general de esta tesis

Para alcanzar los objetivos señalados, la presente tesis se ha estructurado como sigue:

- En el capítulo 2, se explica con apoyo de la bibliografía, en qué consisten los problemas de *scheduling*: cuales son las diferentes estructuras que tienen, diferentes características de los trabajos, varias funciones objetivo y por último se explican los diferentes tipos de recursos adicionales.
- En el capítulo 3 se muestra una revisión bibliográfica de los diferentes trabajos que se relacionan con los problemas y las soluciones que se van a proponer en esta tesis doctoral.
- El capítulo 4 se centra en el problema del taller de flujo de permutación para el objetivo makespan con recursos adicionales. Primero se explica y define el problema, utilizando un modelo matemático para ayudar en la explicación. Después se plantean dos formas de evaluar cuando se tienen recursos adicionales. Seguidamente se proponen y explican diferentes algoritmos constructivos basados en reglas deterministas y heurísticas para el problema. Además, se incluye la experimentación realizada para comparar la eficacia y la eficiencia de los métodos propuestos. En este capítulo también se incluye la explicación de las instancias que se usarán en esta tesis doctoral y finalmente se exponen las conclusiones del capítulo.
- En el capítulo 5 se explican y se proponen metaheurísticas para el mismo problema, cada una de ellas es calibrada a través de un proceso de experimentación que se expone y se explica. Finalmente, se realiza una segunda experimentación entre los métodos ya calibrados, se analizan los resultados y se plantean las conclusiones del capítulo.
- El capítulo 6 presenta el *framework* FACOP, se explican sus diferentes capacidades y fun-

cionalidades. Además, se exponen todos los esfuerzos realizados en este frente para reducir el GAP entre la ciencia y la industria, aportando ejemplos concretos.

- El capítulo 7 presenta las conclusiones de toda la tesis, y se explican otras aportaciones de la misma así como lineas futuras.
- Finalmente, en los apéndices encontramos materiales adicionales que han sido movidos del grueso de la tesis para mejorar la organización y reducir el contenido en tablas y figuras para facilitar la lectura.

2. Programación de la producción realista y recursos adicionales

En este capítulo, se exponen de manera resumida los principales tipos de problemas de programación de la producción que se pueden encontrar. Se presentará la notación especifica más utilizada y se mostrarán algunas formas de clasificar estos problemas, notación y clasificación que posteriormente se usarán en esta tesis.

La programación de la producción define cómo se utilizan los recursos productivos a lo largo del tiempo para obtener la producción deseada cumpliendo con ciertas restricciones y criterios establecidos. Ahora bien, los recursos productivos varían enormemente de una empresa a otra, incluso cuando las empresas trabajan en áreas o productos similares. Por esta razón, existen gran cantidad de problemas diferentes en el área de programación de la producción. Para plantearlos, es necesario utilizar una notación o nomenclatura específica que nos permita expresar y explicar sus diferencias.

En la literatura especializada de investigación operativa, para la transformación de un problema real en un modelo formal, es necesario establecer unos nombres en base a la funcionalidad que tiene cada elemento en el entorno productivo. De esta manera, se utilizan algunos nombres clave para abstraernos del problema real y tener un lenguaje común. Estos cambios son una convención de nombres y se realizan sin pérdida de generalidad en el problema. Así, se denomina "máquina" al recurso o conjunto de recursos que procesan los bienes o productos. Se denomina "trabajo" al bien o producto en sí mismo y se denomina "taller" al entorno productivo en su conjunto cuando los trabajos necesitan más de un procesado. Se les conoce por estos nombres independientemente de su naturaleza real. Por ejemplo, un punto para lijado a mano de las piezas sensibles de un mueble en una fábrica de muebles se identificará como una "máquina", independientemente del hecho de que lo sea o no.

Existen muchas clasificaciones y notaciones, en Graham, Lawler, Lenstra y Kan (1979), se planteó la notación $\alpha/\beta/\gamma$ para catalogar diferentes características que definen problemas de *scheduling* utilizando 3 aspectos importantes del problema:

1. El primer término es α, define el entorno productivo. El término representa por una parte cómo se distribuyen o cómo están colocados los recursos conocidos como máquinas (en serie, en paralelo, mixto). Por otra parte, define cómo se "desplazan" los trabajos respecto a las máquinas para realizar su procesado. Por último, registra el número de recursos de procesado (máquinas) disponibles.
2. El segundo término es β, que representa un conjunto de restricciones que afectan al funcionamiento del taller, muchas veces incluyendo características especiales o limitaciones de los trabajos o las máquinas.

3. Finalmente, γ representa el o los criterios que se buscan mejorar. Es *de facto* el objetivo del problema e igual que los anteriores puede cambiar radicalmente de un caso a otro.

A continuación, se definen estos términos con más detalle y se señalan muchas de las opciones posibles, poniendo especial énfasis en las más relevantes para esta tesis.

2.1 Clasificación en base a la distribución de los recursos del entorno productivo (α)

En la terminología de Graham, (α) es una o varias letras clave para señalar cómo se colocan las máquinas y cómo se desplazan los trabajos en el entorno productivo. Las diferentes letras posibles las explicamos a continuación. En algunos casos, se añade un número para indicar cuántas máquinas hay disponibles en el entorno productivo. Si no aparece este número, significa que el número de máquinas no está fijado y puede variar.

En un problema de *scheduling*, se deben procesar n trabajos del conjunto $N = (1,2,\ldots,n)$, y se dispone de m máquinas del conjunto $M = (1,2,\ldots,m)$. Para identificar un elemento, se utilizan los subíndices j e i, respectivamente.

2.1.1 Problemas de máquinas

Son problemas en los que una o más máquinas procesan los trabajos, y cada trabajo ha de ser procesado solo una vez.

Una sola máquina (1)

En este caso especial, donde solamente existe una máquina, esta tiene que procesar todos los trabajos una sola vez. Por lo que se estudia solo la secuenciación de los trabajos, es decir, en que orden se procesan, ya que no se pueden asignar trabajos a otras máquinas. Este problema ha sido ampliamente estudiado.

Máquinas paralelas idénticas (P)

Las distintas máquinas se distribuyen en paralelo de tal forma que los trabajos disponibles para ser procesados, pueden entrar en cualquiera de ellas, siempre que la máquina no esté procesando otro trabajo. En este caso las máquinas son idénticas, por lo que el tiempo de procesado de un trabajo es el mismo en todas las máquinas. El tiempo de proceso es el vector $P_j = (p_1, p_2, \ldots, p_n)$.

Máquinas paralelas uniformes (Q)

Estas máquinas, de nuevo en paralelo, tienen velocidades diferentes entre ellas; sin embargo, los tiempos de proceso son proporcionales. Los tiempos de procesado de cada trabajo se calculan multiplicando la velocidad de la máquina $V_i = (v_1, v_2, \ldots, v_m)$ por el tiempo de procesado base $P_j = (p_1, p_2, \ldots, p_n)$. En este problema, la asignación es relevante, ya que el tiempo de finalización varía según qué máquina se utilice.

Máquinas paralelas no relacionadas (R)

Por diversas razones en un entorno real de máquinas paralelas, las máquinas acaban siendo diferentes entre si, debido a factores como el desgaste, la especialización en ciertos trabajos, etc. Como resultado encontramos un caso muy general, que es aquel en el que cada trabajo tiene un tiempo de procesado diferente según en qué máquina se procese. Se necesita una matriz de valores que son los tiempos de proceso de cada trabajo en cada máquina, este es $P_{ij} = (p_{1,1}, p_{1,2}, \ldots, p_{m,n})$.

2.1.2 Problemas de talleres o *shops*

Se llaman talleres o *shops* aquellos problemas en los que el bien procesado, al que llamamos trabajo, tiene que pasar por diferentes procesos antes de considerarse finalizado. Dependiendo de si

existe una ruta o rutas prefijadas y cómo son estas encontramos uno u otro tipo de taller.

Taller de flujo o *Flowshop* (F)

El taller de flujo o *Flowshop* es aquel en el que los productos o trabajo han de ser procesados por un conjunto de máquinas en una secuencia u orden prefijado que se mantiene constante para todos ellos. En este problema cada uno de los n trabajos tiene m tareas de procesado y cada una de estas tareas se procesa en una máquina distinta con un tiempo de procesado diferente, además el orden de procesamiento en las máquinas es fijo y siempre es el mismo. Algunas líneas de montaje o ensamblado se pueden modelar como un taller de flujo, el ensamblado va proceso a proceso y siempre sigue el mismo orden.

Taller de trabajos o *Jobshop* (J)

A diferencia del *Flowshop*, en un taller de trabajos o *Jobshop*, cada trabajo tiene un orden o ruta de procesado propio, por ejemplo, una ruta podría ser (m_1, m_3, m_4, m_2). En el caso mas teórico de la investigación de *jobshops*, los trabajos visitan todas las máquinas exactamente una vez, sin saltarse ninguna (J. M. Framinan, Leisten y Ruiz García 2014). Otras muchas veces está permitido por lo que es habitual confirmarlo de forma explicita con los términos *recrc*, *reproc* o *skip* en el campo β, estos casos se explican junto con los demás casos de β en la siguiente sección. Algunos talleres mecánicos de reparaciones se podrían modelar como un taller de trabajos, no todas las reparaciones han de seguir el mismo flujo de procesado, pero si suelen hacerlo. Dependiendo de las características específicas del trabajo que se realiza, el orden de las máquinas variará, pero como norma general necesitará varios procesados.

Taller abierto u *Openshop* (O)

En este taller no existen restricciones de distribución ni existe orden prefijado de tareas para un trabajo dado. Así, dos productos iguales pueden seguir rutas distintas de procesado y acabar siendo iguales.

Talleres híbridos o *Hybrid shops* (HF, HJ, HO)

A diferencia de los casos anteriores, en los casos híbridos en lugar de tener que procesar los trabajos en máquinas, las tareas se procesan en una etapa. Cada etapa contiene una máquina o un conjunto de máquinas paralelas. Para ser considerado un taller híbrido es necesario que al menos una etapa sea de máquinas paralelas. En esta etapa o etapas, las máquinas paralelas pueden ser idénticas, uniformes o no relacionadas. Los trabajos han de pasar por las etapas de procesado cumpliendo las restricciones propias de cada tipo de taller y denotándose como HF, HJ y HO (Hybrid Flowshop, Hybrid Jobshop y Hybrid Openshop).

Taller de flujo híbrido flexible *Hybrid Flexible Flowshop* (HFF)

Es el caso especial de un taller de flujo híbrido donde no todos los trabajos realizan todas las tareas. Para que un trabajo sea procesado completamente no se necesita realizar una tarea en cada etapa, en algunos trabajos se procesarán solo en un subconjunto de las etapas, respetando el orden. Pueden ser necesarios re-procesados.

2.2 Restricciones (β)

En el término β se engloban muchas posibles restricciones que se refieren a características de los trabajos. A continuación, se explican de forma sucinta algunas de las más importantes.

Pre-emption (*pmtn*): cuando se permite la interrupción o corte de un trabajo que más adelante podrá continuar su procesado.

Recursos adicionales (*res*): en los problemas de *scheduling* siempre se tiene un tipo de recurso para procesar trabajos al que conocemos como máquina. Cuando existe más de un tipo de recurso

vinculado al procesado de los trabajos se les conoce como recursos adicionales, o *additional resources*. Estos pueden representar: trabajadores, máquinas, materias primas, energía, etc.; y siempre son limitados de una u otra forma. Estos tipos de recursos adicionales se explican con más detalle en la sección 2.4 ya que forman parte del problema que se aborda en esta tesis.

Precedencia (*prec*): existe una precedencia entre los trabajos existentes, uno o más trabajos necesitan que otro u otros trabajos hayan sido procesados con anterioridad. Según la cantidad de predecesores y sucesores que puede tener un trabajo se utilizan los términos: *chain, tree, intree u outree*.

Fechas de entrada (r_j): en muchos entornos productivos reales, un trabajo no puede empezar hasta que las materias primas o componentes están disponibles. Por esta razón, en muchos modelos encontramos fechas de entrada o *release dates* para cada trabajo. Todos los trabajos del *schedule* resultante empezarán después de su fecha de entrada. Cuando esta restricción no aparece se sobreentiende que los trabajos están disponibles en el instante inicial.

Tiempos de proceso (p_{ij} ó $\underline{p} \leq p_{ij} \leq \overline{p}$): cuando los tiempos de proceso cumplen algunas características se puede especificar en este campo. Por ejemplo, $p_{ij} = 1$ especifica que el tiempo de procesado de todas las tareas es uno. Cuando existen cotas inferiores y superiores también se puede reflejar con la notación $\underline{p} \leq p_{ij} \leq \overline{p}$.

Permutación (*prmu*): en los talleres de flujo encontramos este caso especial donde la secuencia o permutación de trabajos procesados es constante e idéntica en todas las máquinas. Está característica de los trabajos se profundiza más adelante por ser parte del problema que se estudia en esta tesis.

Tiempos de cambio (S_{sd} o S_{nsd}): el tiempo de cambio o *setup* es un sobre-coste en el tiempo que se necesita para preparar una máquina o un trabajo antes o después del tiempo de procesado en si mismo. Está causado por múltiples causas reales, puede ser por configurar las máquinas, enfriamiento, limpieza, etc. Dependiendo de la naturaleza del problema estos tiempos de cambio pueden ser anticipativos o no anticipativos, según si ese tiempo está ligado a hacerse justo antes del siguiente procesado o no. También existe la distinción de tiempos de cambio dependientes de la secuencia (S_{sd}) o independientes (S_{nsd}) de ella. Son tiempos de cambio dependientes de la secuencia cuando el sobre-coste de tiempo varía dependiendo de qué trabajo se acaba de procesar en la máquina y qué trabajo se va a procesar a continuación. En este caso tendremos tiempos de cambio diferentes según la tarea que se ha de realizar.

Elegibilidad de máquina (M_j): algunos trabajos no pueden ser procesados por todas las máquinas. Estos deben tener una lista de posibles máquinas para ser procesados.

Disponibilidad de máquinas y roturas (*brkdwn*): se refiere a los casos donde ya sea por roturas o mantenimiento, las máquinas sufren paradas durante las cuales no pueden procesar trabajos.

Re-circulado, re-procesado y salto (*recrc*, *reproc* y *skip*): son los casos especiales donde un trabajo puede tener que volver a ser procesado por la misma máquina (re-circulado), cuando algunas máquinas pueden necesitar varios procesados (re-procesado) debido a no haber acabado satisfactoriamente el primer procesado y cuando algunos trabajos pueden saltar un procesado o etapa que no necesitan realizar (salto).

Máquinas no inactivas ($no-idle$): en algunos entornos productivos las máquinas no pueden entrar en reposo o inactividad una vez han procesado su primer trabajo. La programación de la producción deberá entonces asegurar que una vez esa máquina se activa, procesa todos los trabajos que tiene que procesar sin insertar tiempos ociosos entre los trabajos.

Máquinas de procesado por lotes ($batch$): ya sea en serie o en paralelo, son máquinas que procesan múltiples trabajos, como un lote, bien por necesidad o para ser más eficientes.

Fechas de entrega (d_j o $\overline{d}_j$): representa el momento de entrega máximo para un trabajo, supone la necesidad de que el trabajo esté completamente procesado antes de ese instante o fecha. Las fechas de entrega o *due dates* pueden ser completamente restrictivas ($\overline{d}_j$) en cuyo caso se llaman *dead lines* y en caso de incumplirse, la solución no sería factible. Las fechas de entrega que no son completamente restrictivas (d_j) pueden tener un impacto en el beneficio de la empresa ya que

acabar antes de la fecha de entrega el procesado puede acarrear costes extra de almacenaje, sin embargo, acabar demasiado tarde también puede tener penalizaciones contractuales o de otro tipo. La evaluación de los sobre-costes relacionados con las fechas de entrega se explicará con más detalle en la sección 2.3. Un caso particular más realista son las ventanas de entrega o *due date windows* donde la fecha de entrega es en realidad un periodo de tiempo durante el cual acabar el procesado se considera correcto.

Sin espera ($no-wait$): en algunos problemas, los trabajos han de ser procesados sin tiempos de espera entre el procesado en una máquina y el procesado en la siguiente.

Restricciones de almacenamiento ($buffer$ o $block$): en algunos casos existe una limitación al almacenamiento ($buffer$), este puede ser un valor idéntico para todas las máquinas o relativo a la máquina i (b o b_i). Cuando no existe capacidad intermedia, es decir, no se puede almacenar el producto se conocen como bloqueos ($block$), la capacidad intermedia es 0.

2.3 Criterios u objetivos (γ)

El último término (γ) define la función objetivo u objetivos de optimización. A continuación, se plantean algunas notaciones que se utilizan para definir este término (J. M. Framinan, Leisten y Ruiz García 2014; French 1982). Las medidas de rendimiento más estudiadas en la literatura para este tipo de problemas son las siguientes:

2.3.1 — Medidas de rendimiento.

• Completion time (C_j): Tiempo de finalización del trabajo j en el taller.
• Flow time (F_j): Tiempo del que ha dispuesto un trabajo desde su fecha de liberación hasta su fecha de finalización.

$$F_j = C_j - r_j$$

• Lateness (L_j): Tiempo de holgura de un trabajo. Es el tiempo que transcurre entre la fecha de finalización hasta la fecha de entrega, puede ser un valor negativo.

$$L_j = C_j - d_j$$

• Tardiness (T_j): Tiempo de retraso de un trabajo. Para los trabajos que finalizan después de la fecha de entrega, es el tiempo que transcurre desde la fecha de entrega hasta la fecha de finalización.

$$T_j = \max\{0, d_j - C_j\}$$

• Earliness (E_j): Tiempo de adelanto de un trabajo. Para los trabajos que finalizan antes de la fecha de entrega, es el tiempo que transcurre desde la fecha de finalización hasta la fecha de entrega.

$$E_j = \max\{0, C_j - d_j\}$$

• Earliness-Tardiness (ET_j): Es la suma de los tiempos de adelanto y retraso para cada trabajo. También es equivalente al valor absoluto del tiempo de holgura.

$$ET_j = E_j + T_j$$

• Late job (U_j): Trabajo retrasado, será uno siempre que el trabajo se retrase respecto a su fecha de entrega.

$$U_j = \begin{cases} 1 & L_j > \ 0 \\ 0 & L_j <= \ 0 \end{cases}$$

• Early job (V_j): Trabajo adelantado, será uno siempre que el trabajo se adelante respecto a su fecha de entrega.

$$V_j = \begin{cases} 1 & L_j < \ 0 \\ 0 & L_j >= \ 0 \end{cases}$$

• Just in time (JIT_j): Trabajo *just-in-time*, será uno siempre que el trabajo se adelante o retrase respecto a su fecha de entrega.

$$JIT_j = \begin{cases} 1 & L_j = 0 \\ 0 & L_j \neq 0 \end{cases}$$

A partir de estas medidas se generan diferentes objetivos de optimización. Muchos de los objetivos posibles están representados en la tabla 2.1, en esta tabla se facilita de cada objetivo su nombre, la denominación que se utiliza y en algunos casos con siglas, y en la última columna se presenta el cálculo en fórmula. En la tabla se observa como estos objetivos pueden basarse en minimizar el máximo valor de esa métrica en todo el taller, en cuyo caso se representa con el subíndice *max* de máximo. También pueden ser minimizar el valor de la suma total de la métrica para todos los trabajos y se representaría como el sumatorio de todas las métricas de j, desde 1 a n, todos los trabajos. Una tercera opción puede ser minimizar el promedio de esa métrica para todos los trabajos, lo que se representa con una linea sobre la métrica que siempre significa promedio.

Nombre	Notación	Fórmula
Tiempo máximo de finalización o makespan	C_{max}	$\max C_j = \max\{C_1, C_2 \dots C_n\}$
Tiempo de flujo máximo	F_{max}	$\max F_j = \max\{F_1, F_2 \dots F_n\}$
Tiempo de holgura máxima	L_{max}	$\max L_j = \max\{L_1, L_2 \dots L_n\}$
Tiempo de retraso máximo	T_{max}	$\max T_j = \max\{T_1, T_2 \dots T_n\}$
Tiempo de adelanto máximo	E_{max}	$\max E_j = \max\{E_1, E_2 \dots E_n\}$
Tiempo total de finalización (TCT)	$\sum C_j$	$\sum_{1..n} C_j$
Tiempo total de flujo (TFT)	$\sum F_j$	$\sum_{1..n} F_j$
Tiempo total de holgura (TL)	$\sum L_j$	$\sum_{1..n} L_j$
Tiempo total de retraso (TT)	$\sum T_j$	$\sum_{1..n} T_j$
Tiempo total de adelantos (TE)	$\sum E_j$	$\sum_{1..n} E_j$
Tiempo total de adelantos y retrasos (TET)	$\sum ET_j$	$\sum_{1..n} ET_j$

Continua en la página siguiente

Table 2.1 – continua desde la página anterior

Nombre	Denominación	Fórmula
Número de trabajos retrasados	$\sum U_j$	$\sum_{1..n} U_j$
Número de trabajos adelantados	$\sum V_j$	$\sum_{1..n} V_j$
Número de trabajos just-in-time (JIT)	$\sum JIT_j$	$\sum_{1..n} JIT_j$
Tiempo de finalización promedio	$\overline{C}$	$\overline{C} = \frac{1}{n} \times \sum_{1..n} C_j$
Tiempo de flujo promedio	$\overline{F}$	$\overline{F} = \frac{1}{n} \times \sum_{1..n} F_j$
Tiempo de holgura promedio	$\overline{L}$	$\overline{L} = \frac{1}{n} \times \sum_{1..n} L_j$
Tiempo de retraso promedio	$\overline{T}$	$\overline{T} = \frac{1}{n} \times \sum_{1..n} T_j$
Tiempo de adelanto promedio	$\overline{E}$	$\overline{E} = \frac{1}{n} \times \sum_{1..n} E_j$

Tabla 2.1: Tabla con los objetivos de minimización más utilizados

Por último, a estas medidas de rendimiento se les puede añadir pesos, también conocidos como ponderaciones o prioridades. Los pesos se representan por w_j y se multiplican por las medidas de rendimiento dando mayor relevancia a unos trabajos respecto a otros. Una situación común en un sistema de producción se da cuando el coste de entregar con retraso un producto sea diferente y dependa del producto o del cliente. Por ejemplo, la notación para representar el tiempo total de retraso ponderado (*Total Weighted Tardiness - TWT*) será como el que se muestra en la ecuación 2.1. De forma idéntica se pueden añadir pesos a otros objetivos. En el caso excepcional del tiempo total de adelantos y retrasos (*Total Weighted Earliness Tardiness - TWET*), se tienen dos valores diferentes ya que el sobre-coste de almacenaje y el sobre-coste de un retraso no suelen estar vinculados. La fórmula 2.2 como los pesos tienen diferentes valores en este caso especial.

$$TWT_{\text{total}} = \sum_{1}^{n} w_j T_j \tag{2.1}$$

$$TWET_{\text{total}} = \sum_{1}^{n} (w_j E_j + w'_j T_j) \tag{2.2}$$

2.4 Recursos Adicionales (*res*)

En el trabajo de Błażewicz, Lenstra y Kan (1983) se propuso una extensión a la notación de Graham. Esta notación denominada $\lambda/\sigma/\delta$ proporciona mayor detalle sobre el funcionamiento de los recursos adicionales. Cada uno de los términos es un número entero positivo, pero admite un valor indeterminado, denotado por el punto "·".

- Lambda (λ): cantidad de tipos de recursos adicionales a considerar; por ejemplo, 1 representaría un tipo de recurso adicional.
- Sigma (σ): todos los tamaños de recursos requeridos son igual a σ. Si λ es mayor que 1, tendremos una lista de valores para σ. Este término solo se usa cuando los tamaños de recursos están bloqueados (son como paquetes).

• Delta (δ): los recursos requeridos máximos de todas las tareas tienen una cota superior conocida.

El valor indeterminado "·" indica que no se conoce de antemano el valor concreto que toma el término y no es un valor constante. En este caso se daría un valor de entrada a cada problema concreto. Por ejemplo, $res(\lambda = 1/\sigma = \cdot/\delta = 9)$, representaría un problema con un solo recurso adicional, donde los recursos no tienen que agruparse y los recursos requeridos son 9 o menos para todos los trabajos.

2.4.1 Características de los recursos

En Błażewicz, Lenstra y Kan (1983), Edis, Oguz y Ozkarahan (2013), Edis y Ozkarahan (2012) y Kellerer y Strusevich (2008), se identifican diversas características de los recursos adicionales que influyen de manera significativa en la forma en que estos se modelan. A continuación, se describen algunas de estas diferencias que, al mismo tiempo, nos permiten clasificarlos.

Dependiendo de su comportamiento con respecto al tiempo de procesado, los recursos pueden ser:
• Recursos de aceleración ($speeding-up$): aquellos que al ser asignados reducen el tiempo de procesado.
• Recursos fijos (fix): son necesarios pero no alteran el tiempo de procesado.

Una forma de clasificar los recursos en problemas de máquinas paralelas es en función de la forma en la que se produce la asignación de recursos (Daniels, Hoopes y Mazzola 1996).
• Recursos estáticos (Stc): son aquellos que se preasignan a una máquina, y no pueden cambiar de máquina en todo el horizonte de la planificación.
• Recursos dinámicos ($Dynamic$): son los recursos adicionales que pueden asignarse a cualquier máquina.

La notación de Błażewicz, Lenstra y Kan (1983) fue ampliada por Kellerer y Strusevich (2008) para problemas de máquinas paralelas con recursos adicionales dinámicos ($DPMFRS$) con recursos de aceleración. Estos se clasifican dependiendo de como interactúa el recurso adicional en la reducción del tiempo de procesado.
• Recursos binarios (Bi): un recurso puede ser asignado o no asignado. Si está asignado, se reduce el tiempo de procesado en una constante dependiente de la máquina, conocida como π_i.
• Recursos enteros (Int): asignando una cantidad entera de recursos adicionales τ se reduce el tiempo de procesado según una tabla $p_{i\tau}$.
• Recursos lineales (Lin): el tiempo de procesado se reduce de forma lineal. Se asigna una cantidad de recursos τ y esto reduce el tiempo de procesado en τ veces un valor de aceleración dependiente de la máquina que se conoce previamente, π_i. $p_{i\tau} = p_i - \tau \times \pi_i$.

Dependiendo del momento en el que se necesitan estos recursos, se pueden encontrar recursos de procesado y recursos de entrada o salida (Blazewicz, Brauner y Finke 2004; Edis, Oguz y Ozkarahan 2013):
• Recursos de procesado: los recursos de procesado son requeridos durante el procesamiento del trabajo.
• Recursos de entrada o salida: los recursos de entrada son requeridos puntualmente en el instante que entra el trabajo en la máquina. Cuando los recursos son requeridos en el momento de finalizar el procesado de un trabajo en una máquina, se conocen como recursos de salida.

En base a la restrictividad de los recursos, estos pueden ser renovables, no renovables y doblemente restringidos (Blazewicz et al. 2019; Słowiński 1980):

• Recursos renovables: los recursos renovables son aquellos que no se consumen en el procesado de los trabajos y vuelven a estar disponibles una vez el procesado ha terminado. Un ejemplo puede ser el personal que trabaja haciendo una tarea y cuando esta termina, al trabajador se le asigna otra tarea.
• Recursos no renovables: son aquellos recursos que por su naturaleza dejan de estar disponibles una vez se ha terminado el trabajo. Las materias primas son un ejemplo de este tipo de recurso ya que una vez se emplea pasa a ser parte del producto y deja de estar disponible, se agota.
• Recursos doblemente restrictivos: cuando existen recursos de ambos tipos al mismo tiempo en el entorno productivo.

En base a la divisibilidad se pueden catalogar como:
• Recursos discretos: cuando se utilizan en los trabajos como unidades discretas con un set finito de posibles formas de asignación.
• Recursos continuos: cuando se pueden utilizar cantidades arbitrarias en un intervalo.

Además, se pueden clasificar en función de la cantidad de tipos de recursos diferentes:
- Se dispone de solo un tipo de recurso (*one*).
- Se dispone de uno o varios tipos de recursos (*multi*).

En los problemas de máquinas paralelas con recursos adicionales se estudian un nuevo tipo de problemas de máquinas paralelas (Edis, Oguz y Ozkarahan 2013). Los problemas de máquinas dedicadas (o especificadas), en inglés *dedicated or specified*, son problemas de máquinas paralelas con una simplificación en la que los trabajos están asignados a máquina a priori. Las máquinas no especificadas, en inglés *unspecified*, serían los problemas de máquinas que se han explicado en la sección 2.1 con sus diferencias, donde los trabajos han de ser asignados a máquina.

2.4.2 Notaciones y características aplicadas al problema de esta tesis

A continuación, se utilizan las clasificaciones y explicaciones previamente definidas en el problema del taller de flujo de permutación con recursos adicionales renovables (PFSPR). Se utilizan las notaciones y clasificaciones para explicar el problema y sus características, aunque la definición formal se aporta en el capítulo. La notación de Graham, Lawler, Lenstra y Kan (1979) para el problema PFSPR, aparece en la fórmula 2.3. Respecto a esta, el primer término (α) corresponde al taller de flujo que se representará con F. Si es un problema específico en el que se conozca el número de máquinas, este se añadirá. El segundo término (β) contiene las palabras clave de permutación (*prmu*) y la palabra clave de recursos (*res*). En el tercer término el objetivo es que el taller acabe lo antes posible de procesar todos los trabajos, optimizando el *makespan*, es decir, la minimización del tiempo máximo de finalización ($minimize\ C_{max}$).

$$Fm\ /\ prmu,\ res\ /\ minimize\ C_{max} \tag{2.3}$$
$$Fm\ /\ prmu,\ res(1,\cdot,9)\ /\ minimize\ C_{max} \tag{2.4}$$

También se presenta la notación extendida de Błażewicz, Lenstra y Kan (1983) ("$\lambda/\ \sigma/\ \delta$") en la fórmula 2.4. En ella λ es uno, indicando existe un solo tipo de recurso extra. El segundo término σ, se representa por un punto, mostrando que los recursos requeridos son un valor de entrada y es por unidades. El tercer término δ es 9, y representa el número máximo de recursos requeridos por cualquier tarea.

A pesar de estar usando ambas notaciones sigue siendo necesario añadir mucha más información. En base a la clasificación descrita anteriormente, los recursos del problema que tratamos son los siguientes: son recursos fijos (*fix*) ya que son necesarios para el procesado de los trabajos, pero no aceleran su velocidad de procesado. Por lo que la extensión de Kellerer y Strusevich (2008) no

se aplica. Son recursos renovables, ya que una vez se han utilizado en una tarea, y esta ha acabado, vuelven a estar disponibles para ser utilizados nuevamente en otra. Se dispone de un solo recurso adicional, de un solo tipo (*one*).

Otras características de los recursos adicionales pueden parecer más forzadas, esto se debe a que reflejan situaciones excepcionales. Estos casos como podrían ser los recursos especificados y estáticos (*specified* y *static*), donde los recursos vienen asignados a una máquina a priori o tienen que ser asignados a cada máquina a priori. Como ninguna de las dos situaciones se dan y se puede asignar cualquier recurso a cualquier máquina, tenemos recursos no especificados y dinámicos (unspecified and dynamic). Por último, se puede afirmar que son recursos de procesado, puesto que se necesitan durante todo el tiempo de procesamiento de la tarea y no en un momento concreto. Esto implica que, significa que mientras la tarea esté siendo procesada los recursos están bloqueados y no se pueden ser usados en otras tareas.

En el presente capítulo se ha podido evidenciar que es prácticamente imposible enumerar, clasificar y ordenar todas las situaciones que se pueden dar en los procesos productivos del mundo real. Con el fin de poder definir y comprender dichos fenómenos, en el transcurso del presente capítulo se ha recurrido a notaciones y clasificaciones muy utilizadas que facilitan la comprensión de estos problemas de *scheduling* y simultáneamente proporcionen un lenguaje común que permite la comunicación efectiva en el campo de estudio.

3. Revisión bibliográfica

Los problemas de programación de la producción han sido objeto de numerosos estudios en la literatura académica. El primer trabajo sobre el problema del taller de flujo con permutación se encuentra en S. Johnson (1954). Desde entonces, las investigaciones en el ámbito del programación de la producción (scheduling) han ido adquiriendo cada vez más importancia en la literatura académica. Tal como se señaló en el capítulo 2, el ámbito del *scheduling* es muy amplio y los problemas reales pueden variar significativamente según las características específicas. Las metodologías planteadas para resolver problemas de *scheduling* se centraron inicialmente en reglas y métodos exactos. Posteriormente, las heurísticas cobran más protagonismo, pero desde antes del año 2000 las metaheurísticas empiezan a ser las herramientas más utilizadas, especialmente en los casos más realistas (Kellerer y Strusevich 2008; Ribas, Leisten y Framiñan 2010; Rossit, Tohmé y Frutos 2018; Ruiz y Vázquez-Rodríguez 2010).

Los problemas realistas, tales como el problema del taller de flujo de permutación con recursos adicionales abordado en esta tesis, han recibido menos atención. Esto se explica, en parte debido a su mayor complejidad y en parte porque un trabajo de carácter más general puede llegar a más audiencia, aunque no sea muy aplicable a casos reales.

Sin embargo, esta situación ha experimentado un cambio en los últimos años, con un creciente número de trabajos sobre problemas más realistas como los que involucran recursos adicionales.

Dada esta situación, la revisión bibliográfica no debe centrarse únicamente en trabajos sobre el problema a resolver: el taller de flujo de permutación con recursos renovables adicionales para optimizar el makespan. Por lo tanto, esta revisión bibliográfica será amplia y se subdividirá en cuatro apartados dependiendo de cual es la aportación principal de cada trabajo. En el primer apartado se encuentran: las revisiones bibliográficas, los trabajos que proponen notaciones y los que se centran en la complejidad de los problemas. Los tres siguientes apartados se subdividirán según el tipo de solución propuesta: métodos exactos, algoritmos heurísticos y algoritmos metaheurísticos.

3.1 Notación, complejidad y revisiones

Las notaciones que se han considerado más importantes en este área de estudio ya han sido explicadas en detalle en el capítulo anterior, por lo que ahora solo se mencionarán.

En Graham, Lawler, Lenstra y Kan (1979) los autores realizan una revisión sobre problemas de *scheduling* y añaden una notación conocida como $\alpha\beta\gamma$. En este completo trabajo también se estudia la complejidad de muchos problemas de *scheduling* y se muestran los árboles de jerarquía de complejidad para los campos α y γ. Estos árboles también están explicados en múltiples

fuentes (J. M. Framinan, Leisten y Ruiz García 2014; Graham, Lawler, Lenstra y Kan 1979; Pinedo 2016). En Pinedo (2016) se muestra el siguiente ejemplo de reducción en complejidad: $1||\sum C_j \propto 1||\sum w_j C_j \propto Pm||\sum C_j \propto Qm|prec|\sum C_j$. En este ejemplo se diría que $1||\sum C_j$ reduce a $1||\sum w_j C_j$. Esta jerarquía permite entender que si un problema A reduce a otro problema B, la complejidad del problema B será mayor o igual a la complejidad del problema A. Garey, D. S. Johnson y Sethi (1976) estudiaron la complejidad de muchos problemas del taller de flujo y del taller de trabajos, demostrando que el problema del taller de flujo es $\mathscr{NP}$-Completo en sentido estricto cuando existen tres o más máquinas. Los autores extienden ese estudio en el libro de Garey y D. S. Johnson (1979). El problema con la restricción de permutación (PFSP) se demostró que también era $\mathscr{NP}$-Completo en sentido estricto en Rinnooy Kan (1976). En Graves (1981) se realiza una revisión del estado del arte para los problemas de producción y se expone el conocido GAP o distancia que existe entre los estudios académicos y la realidad industrial. Se extiende la notación $\alpha\beta\gamma$ en Błażewicz, Lenstra y Kan (1983), añadiendo una restricción detallada en $\beta = \lambda\sigma\delta$. Además se demuestra la complejidad en muchos problemas con recursos como "$F2|res111|C_{\max}$" $\mathscr{NP}$-Difícil en sentido estricto. En Błażewicz, Kubiak, Röck y Szwarcfiter (1987) el problema de minimizar el tiempo medio de flujo con procesos paralelos y recursos adicionales se demostró $\mathscr{NP}$-Difícil en sentido estricto. Se realiza una revisión de los problemas del taller de flujo con el objetivo de minimizar el *makespan*, junto con métodos propuestos, en Reza Hejazi y Saghafian (2005). Otra revisión sobre problemas de *scheduling* con recursos adicionales, especialmente en recursos dinámicos o recursos con tiempos de liberación, se muestra en A. Janiak, W. Janiak y Lichtenstein (2007). En Kellerer y Strusevich (2008) se amplía la notación de Błażewicz, Lenstra y Kan (1983) ($\lambda\sigma\delta$) con términos específicos para problemas de máquinas paralelas con recursos adicionales dinámicos (*DPMFRS*), estos recursos son aquellos que reducen el tiempo de procesado según el número de recursos utilizados. En Ribas, Leisten y Framiñan (2010) y Ruiz y Vázquez-Rodríguez (2010) se presentan dos revisiones del estado del arte en los problemas del taller de flujo híbrido con clasificaciones en base al tipo de solución, tipo de objetivo, etc. E.B. Edis junto con otros autores propusieron varios modelos y heurísticas en varios trabajos consecutivos (Edis y Oguz 2012; Edis y Ozkarahan 2011; Edis y Ozkarahan 2012), en ellos se estudian varios problemas de máquinas paralelas con recursos adicionales. En el artículo Edis, Oguz y Ozkarahan (2013) en concreto, se centran en notaciones y clasificaciones para los problemas de máquinas paralelas con recursos adicionales y hacen una revisión de como se clasifican los diferentes tipos de recursos en base a su funcionamiento y restricciones. En Rossit, Tohmé y Frutos (2018) también se presenta una revisión bibliográfica del problema taller de flujo sin permutación (FSP).

3.2 Métodos exactos

En 1954 se publicó el primer trabajo en el taller de flujo con permutación (PFSP - Permutation Flowshop Scheduling Problem). En este trabajo, S. Johnson (1954), optimiza el makespan para un problema con solo dos máquinas donde se consigue obtener el óptimo aplicando el Algoritmo de Johnson. Se resuelve el problema del taller de flujo con permutación a optimalidad para dos máquinas y el objetivo makespan ($F2 \mid prmu \mid minimize\ C_{\max}$). Existen principalmente dos aproximaciones exactas para resolver este tipo de problemas.

La primera aproximación que se documenta a continuación, es la búsqueda de resolución de problemas con modelos matemáticos. En Wagner (1959) se presenta un modelo de programación lineal entera (ILP - *Integer Linear-Programming*) para el problema del taller de flujo con el objetivo makespan. Sobre este modelo en Baker (1974) se proponen mejoras, que será modificado a su vez en Stafford (1988) usando ahora un modelo mixto (MILP - *Mixed Integer Linear-Programming*). En Srikar y Ghosh (1986) también se utiliza un modelo MILP para el problema de flowshop con permutación con tiempos de cambio que fue depurado en Stafford Jr y Tseng (1990) con varios modelos MILP. En C.-H. Pan (1997) se analizan varios de los modelos presentados hasta la fecha

y se catalogan de mejor a peor en base al número de variables. En Wilson (1989) se adapta al problema de flowshop de permutación un modelo creado originalmente para el taller de trabajos propuesto en Manne (1960), el modelo resultante se mejora de nuevo en Tseng, Stafford Jr y Gupta (2004). En este último trabajo, se comparan varios de los modelos estudiados hasta el momento y se catalogan en base al tiempo de CPU requerido para resolver estos problemas. Estos resultados indican que el orden de mejor a peor es: Wagner, Wilson y Manne. Esta forma de evaluar la calidad de un modelo MILP es claramente superior a la propuesta por Pan en C.-H. Pan (1997). Tres problemas del taller de flujo con restricciones adicionales se resuelven con modelos MILP en Sadjadi, Aryanezhad y Ziaee (2008). En Naderi y Ruiz (2010) se proponen adaptaciones de los modelos del problema del taller de flujo al taller de flujo distribuido con permutación (DPFSP - *Distributed Permutation Flowshop Scheduling Problem*), en total 6 modelos MILP.

En Edis y Ozkarahan (2011) se estudia el problema con recursos renovables fijos de máquinas paralelas idénticas con elegibilidad de máquina (Resource-Constrained Parallel Machine Scheduling Problem - RCPMSP). Se propone un modelo de programación de restricciones (CP), otro de programación entera (IP) y uno mixto (IP/CP) para el objetivo de minimizar el makespan. Se generan o adaptan varios modelos IP, CP y mixtos IP/CP en Edis y Oguz (2012) para los problemas UPMFRS y DPMFRS. A este problema PMFRS *Parallel machine flexible resource scheduling* se añade la U cuando es un problema con recursos *unspecified* y D cuando el problema es *dynamic*. Ambos problemas se resuelven para el objetivo makespan. Un trabajo de entorno real se analiza en Edis y Ozkarahan (2012), para el problema RCPMSP con elegibilidad de máquina se prueba un modelo IP que resulta ser insuficiente por lo que se subdivide el problema en dos fases y se proponen modelos mixtos para resolverlo IP/CP y IP/IP. Sobre los trabajos anteriores se añaden notaciones y clasificaciones en Edis, Oguz y Ozkarahan (2013), además se realiza una revisión bibliográfica de todos los problemas de máquinas paralelas con recursos adicionales. Se proponen dos modelos y se incluye una clasificación de los distintos tipos de recursos adicionales de las que se ha explicado en el capitulo anterior.

En Fanjul-Peyro, Perea y Ruiz (2017) los autores proponen dos modelos MILP para el problemas de máquinas paralelas no relacionadas con recursos renovables adicionales, este problema lo denotan con las siglas UPMR. Sus dos modelos son: UPMR-S, que está basado en literatura de scheduling y UPMR-P, que está basado en la literatura de empaquetado como son los problemas de *bin-packing y strip-packing*. En este trabajo se incluyen también tres matheuristicas y siempre se estudia el objetivo de minimizar el makespan. El problema *Unrelated Parallel Machine scheduling problem with Setups and Resources (UPMSR)* en el que hay recursos para el procesado de las tareas pero también para los tiempos de cambio, que a su vez son dependientes de la secuencia se estudió en Fanjul-Peyro (2020). Los recursos adicionales son renovables y se analizan de tres formas distintas: por separado para el tiempo de proceso, por separado para setups y generales para ambos casos. Para resolver el problema se propone un MILP y un método exacto llamado algoritmo de tres fases (Three Phase Algorithm - TPhA). Se utiliza una descomposición de Benders en Bektaş, Hamzadayı y Ruiz (2020) para resolver el problema del taller de flujo con permutación y "no-idle mixto". En este problema algunas máquinas no permiten tiempos ociosos y otras si. Se obtienen buenos resultados comparados con metaheurísticas de la literatura siempre que las instancias no sean muy grandes. Una comparación de modelos de programación de restricciones (Constraint Programming - CP) con modelos MIP se presenta en Naderi, Ruiz y Roshanaei (2023). Los autores exponen como las soluciones con constraint programming solían resultar claramente peores en tiempo de cálculo así como en el tamaño de problemas que resolvían , pero consiguen mejorar los modelos MIP. Muchos de los trabajos citados de métodos exactos especialmente los resueltos con modelos se pueden ver ordenados de forma cronológica en la tabla 3.1 para el objetivo de minimizar el makespan, aun siendo solo unos pocos trabajos, podemos apreciar que con el tiempo se estudian problemas más realistas, algunos de ellos con recursos adicionales.

Autores / Año	Tipo de Problema	Recursos
S. Johnson (1954)	PFSP	No
Wagner (1959)	FSP	No
Baker (1974)	PFSP	No
Stafford (1988)	PFSP	No
Srikar y Ghosh (1986)	PFSPS	No
Stafford Jr y Tseng (1990)	PFSPS	No
C.-H. Pan (1997)	-	No
Manne (1960)	JSP	No
Wilson (1989)	PFSP	No
Tseng, Stafford Jr y Gupta (2004)	PFSP	No
Sadjadi, Aryanezhad y Ziaee (2008)	PFSP*	No
Naderi y Ruiz (2010)	DPFSP	No
Edis y Ozkarahan (2011)	RCPMSP	De aceleración
Edis y Oguz (2012)	UPMFRS DPMFRS	De aceleración*
Edis y Ozkarahan (2012)	RCPMSP	Si
Edis, Oguz y Ozkarahan (2013)	*	*
Fanjul-Peyro, Perea y Ruiz (2017)	UPMR	Renovables
Fanjul-Peyro (2020)	UPMSR	Renovables
Bektaş, Hamzadayı y Ruiz (2020)	NI-PFSP	No
Naderi, Ruiz y Roshanaei (2023)	*	No

Tabla 3.1: Trabajos en problemas cercanos a PFSPR resueltos con modelos.

Otra aproximación para resolver problemas de *scheduling* son los algoritmos tipo ramificación y acotación, en inglés, *Branch and Bound* (B&B). Los primeros métodos B&B para problemas PFSP son los propuestos en Ignall y Schrage (1965) y Lomnicki (1965) y durante décadas se han planteado nuevos métodos o modificaciones para mejorar este tipo de algoritmos. Stinson, Davis y Khumawala (1978) presentaron un algoritmo B&B para problemas con recursos adicionales en problemas de talleres y procesos. En A. Janiak (1988) se plantea un algoritmo basado en *the disjunctive graph theory* y B&B para problemas con recursos adicionales que son de aceleración. La solución propuesta sirve para resolver problemas de taller de flujo sin permutación. En Daniels y Mazzola (1994) extienden un trabajo anterior sobre el problema del taller de flujo con recursos flexibles (flexible-resource flowshop scheduling (FRFS) problem). Ahora utilizando métodos heurísticos y exactos, y añadiendo además un análisis de como la disponibilidad de recursos afecta al objetivo makespan. Para el problema PFSP con tiempos de cambio dependientes de la secuencia se presenta un algoritmo de B&B en Rios-Mercado y Bard (1999). En Ronconi y Armentano (2001) se proponen nuevas cotas para el problema FSP con bloqueos para objetivos de tardiness y de makespan. Para los últimos trabajos de branch and bound se incluye una guía así como una revisión exhaustiva en Tomazella y Nagano (2020). En este trabajo se puede observar como muchos trabajos se centran en problemas de dos máquinas dejando de lado casos más reales de la industria (Tomazella y Nagano 2020). Sin embargo, como demuestra el reciente artículo, Gmys, Mezmaz, Melab y Tuyttens (2020), los trabajos en este tipo de métodos exactos siguen dando buenos frutos. Los autores proponen un complejo B&B que ha conseguido obtener la solución óptima en dos instancias de Taillard. También explican cómo debido a la complejidad del problema PFSP los esfuerzos se han centrado en metaheurísticas, ya que el tiempo de ejecución de los algoritmos tipo B&B es impredecible y exponencial en el peor de los casos.

3.3 Métodos heurísticos

Continuando con los primeros trabajos en talleres de flujo pero ahora en el campo de las heurísticas, el primer trabajo importante es el de Page (1961) en el se presentan tres heurísticas de reglas de ordenación para resolver el problema del taller de flujo de permutación ($F \mid prmu \mid minimize\, C_{\max}$). Usando conceptos similares a los del Algoritmo de Johnson en Dudek y Teuton Jr (1964) se proponen reglas para minimizar los tiempos ociosos (*idle-times*) de la última máquina. En Palmer (1965) se crea la heurística de pendiente, (*Slope Heuristic*), aunque en muchas ocasiones es referida como el Algoritmo de Palmer. Esta heurística da prioridad a los trabajos con poco tiempo de proceso en sus primeras tareas y mucho tiempo de proceso en sus últimas tareas, para lo que genera un índice de la pendiente o *slope index*. Este algoritmo es muy relevante y muy utilizado para resolver problemas del taller de flujo, por ello se explica en profundidad en la sección 4.2. Posteriormente Campbell, Dudek y Smith (1970) presentaron el algoritmo CDS cuyo nombre se refiere a las iniciales de sus autores, este algoritmo extiende los principios del algoritmo de Johnson para utilizarlo con m máquinas dejando de ser un método exacto para ser un método heurístico. Se proponen métodos heurísticos para problemas de *flowshop* con objetivos del tiempo de finalización, pero también con el tiempo de flujo en Gupta (1971). Nuevos métodos basados en la regla de Palmer se plantean en Bonney y Gundry (1976).

En Nawaz, Enscore Jr y Ham (1983) se presenta el algoritmo NEH cuyo nombre se refiere a las iniciales de sus autores, donde esa heurística constructiva genera una nueva solución desde cero. Partiendo de una solución vacía y un orden de trabajos, inserta cada trabajo en la solución por orden dado, necesitando n iteraciones. En una iteración comprueba en que posición encaja mejor el trabajo de todas las posiciones donde se puede insertar, para finalmente insertar el trabajo en la mejor posición posible de las comprobadas. Un punto crucial para mejorar el uso de esta heurística es la ordenación inicial. Esta heurística ha sido ampliamente utilizada, especialmente en problemas del taller de flujo. Se pueden encontrar más detalles en la sección 4.2. En Taillard (1990) para resolver el problema PFSP con el objetivo *makespan* se propuso una metaheurística, búsqueda tabú (tabu search - TS), con la heurística NEH como método constructivo inicial. En este artículo se incluye la técnica de aceleraciones de Taillard para el método NEH, que consiste en utilizar una matriz de valores para los tiempos de fin de proceso de cada tarea, así durante una iteración NEH en la que se prueban todas las posibles inserciones de un trabajo, los cálculos repetidos no se realizan. Con las aceleraciones de Taillard el orden de complejidad del algoritmo NEH pasa de tener orden $O(n^3m)$ a $O(n^2m)$. Estas aceleraciones se diseñaron para el problema PFSP con el objetivo del *makespan* y no son fácilmente adaptables a todos los problemas ni objetivos, dado que requieren que el camino crítico hasta el punto de inserción no varíe desde el comienzo del *schedule*, ni desde el final del *schedule* a pesar de la inserción realizada. En Taillard (1993) se introduce un set de instancias del problema o *benchmark*, conocido como "las instancias de Taillard", fueron diseñadas originalmente para los problemas base del taller de flujo, taller de trabajos o taller abierto. No obstante, también se han utilizado en otros problemas, requiriendo en muchos casos más información para adaptarse a cada caso concreto. Estas instancias están disponibles en `http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html` y junto a las instancias se almacenan algunos datos como el mejor valor conocido para algunos objetivos (especialmente makespan) y las mejores cotas superiores e inferiores. En la sección 4.5.1 se explican estas y otras instancias.

En J. Framinan, Leisten y Rajendran (2003) se diseñan muchas formas de ordenación para la heurística NEH (Nawaz, Enscore Jr y Ham 1983) para resolver problemas PFSP. Se estudian los objetivos de optimizar el makespan, los tiempos ociosos de las máquinas o el tiempo de flujo, de forma independiente. En Kalczynski y Kamburowski (2008) se establece un completo análisis sobre métodos de ordenación en el algoritmo NEH, siguiendo a otros trabajos anteriores como J. Framinan, Leisten y Rajendran (2003). Los autores propusieron una nueva ordenación, a esta

NEH con ordenación especial basada en ideas del algoritmo de Johnson la llamaron NEHKK1 (o NEH_{kk1}) y mejora a todas las propuestas anteriores. En este trabajo también se argumenta que no creen que sea posible encontrar otra heurística sencilla y relativamente rápida que pueda competir con la NEH para el problema mencionado. Se concluye que NEH es la heurística más popular y potente para ser utilizada como solución inicial en metaheurísticas.

Autores / Año	Tipo de Problema	Recursos	Objetivo
Page (1961)	PFSP	No	C_{max}
Dudek y Teuton Jr (1964)	PFSP	No	C_{max}
Palmer (1965)	PFSP	No	C_{max}
Campbell, Dudek y Smith (1970)	PFSP	No	C_{max}
Gupta (1971)	PFSP	No	C_{max}
Bonney y Gundry (1976)	PFSP	No	C_{max}
Nawaz, Enscore Jr y Ham (1983)	PFSP	No	C_{max}
Taillard (1990)	PFSP	No	C_{max}
Taillard (1993)	-	No	-
J. Framinan, Leisten y Rajendran (2003)	PFSP	No	C_{max} $\sum F_j$ *M idle-times*
Kalczynski y Kamburowski (2008)	PFSP	No	C_{max}
Figielska, Ewa (2008)	HFSP	Renovables 0-1	C_{max}
Figielska, Ewa (2010)	HFSP *pre-empt*	Renovables 0-1	C_{max}
Villa, Vallada y Fanjul-Peyro (2018)	UPMR	Renovables	C_{max}
Z. Liang et al. (2022)	FSP	No	C_{max}

Tabla 3.2: Trabajos con las heurísticas de problemas cercanos a PFSPR

En Figielska, Ewa (2008) se trabaja sobre el problema de taller de flujo híbrido, con dos etapas que tienen máquinas paralelas no relacionadas y recursos renovables de tipo 0-1 necesarios para el procesado, se define el problema y se plantea una heurística como solución. En este método se utiliza un modelo LP con un Solver para resolver el primer problema devolviendo el *schedule* óptimo para la primera máquina en caso de ser un problema independiente, y partiendo de esta solución como fija se completa la solución con una heurística. En Figielska, Ewa (2010) se trata el problema HFSP con recursos 0-1, se extiende añadiendo pre-emption y se resuelve usando heurísticas. Se utilizan las mismas ideas que se definen en Figielska, Ewa (2008) de separar el problema en dos fases y utilizar modelos LP para resolver el problema inicial completando el *schedule* con heurísticas para la etapa de una sola máquina.

Se presentan soluciones heurísticas para el problema UPMR con el objetivo de minimizar el makespan en Villa, Vallada y Fanjul-Peyro (2018). Estas soluciones heurísticas mejoran los resultados de los modelos y las matheurísticas propuestas en Fanjul-Peyro, Perea y Ruiz (2017). Se proponen dos aproximaciones, una trabajando con soluciones con recursos no factibles y reparaciones y otra trabajando con soluciones factibles directamente. Se plantean varias reglas de ordenación, se proponen dos constructores, uno basado en NEH y otro basado en swap, además de una búsqueda local. Finalmente estos métodos se combinan en heurísticas multi-pasada, lo que permite medir el rendimiento que tienen diferentes combinaciones de los métodos propuestos. En Z. Liang et al. (2022) se proponen algoritmos genéticos y formas de generar poblaciones basándose en el algoritmo NEH trabajando en problemas del taller de flujo. En este trabajo se afirma que pese a las mejoras el algoritmo genético propuesto se queda aún distante de las mejores soluciones

conocidas y obtenidas con otros métodos. Al mismo tiempo en este trabajo se reconoce que el algoritmo NEH sigue siendo la heurística predominante para iniciar metaheurísticas, incluso para las iniciaciones con poblaciones.

En la tabla 3.2 se puede ver de forma resumida algunos de los trabajos mencionados que intentar resolver problemas de *scheduling* cercanos al problema PFSPR utilizando como método de resolución principal las heurísticas. Ejemplos actuales como: Laribi, Yalaoui y Sari (2019), Z. Liang et al. (2022), Vallada, Villa y Fanjul-Peyro (2019) y Villa, Vallada y Fanjul-Peyro (2018); demuestran que el método NEH sigue siendo la heurística constructiva más utilizada para obtener la solución o soluciones iniciales en algoritmos metaheurísticos para resolver problemas cercanos al taller de flujo de permutación.

3.4 Métodos metaheurísticos

Una de las primeras metaheurísticas para resolver el problema PFSP fue propuesta en Osman y Potts (1989), en concreto el algoritmo de recocido simulado (Simmulated Annealing - SA). Este algoritmo utiliza una regla primero en llegar primero en servir (First Come First Served - FSFC) para obtener la solución inicial. También usa búsquedas locales de inserción e intercambio para mejorar las soluciones y un criterio de aceptación dependiente de la temperatura que le permite aceptar peores soluciones cuando ha dedicado muchos esfuerzos en un área del espacio de soluciones sin éxito. Al mismo tiempo se propone en Widmer y Hertz (1989) la primera búsqueda tabú, a la que siguieron dos versiones más en Reeves (1993) y Taillard (1990).

Daniels y Mazzola (1993) exponen un algoritmo TS para enfrentar el problema del taller de flujo con recursos flexibles (Flexible-Resource Flowshop Scheduling (FRFS) problem), que son recursos de aceleración. En Nowicki y Smutnicki (1996b) se presenta otra versión de la búsqueda tabú para el problema del taller de flujo con permutación. Los mismos autores adaptaron este algoritmo para los problemas: el taller de trabajos (Nowicki y Smutnicki 1996a) y el taller de flujo híbrido (Nowicki y Smutnicki 1998). Estas búsquedas tabú se apoyan en búsquedas locales para mejorar la solución. La particularidad de las búsqueda tabú es utilizar un listado de soluciones visitadas que se llama lista tabú utilizada para evitar que se vuelva a estudiar una solución que ya ha sido estudiada. Esto permite que el algoritmo no dé vueltas en círculos estudiando de nuevo las mismas soluciones. En Reeves (1995) se diseña una adaptación de algoritmo genético (Genetic Algorithm - GA) para el problema del taller de flujo con permutación para optimizar el makespan que demuestra ser mejor que el algoritmo de recocido simulado (Osman y Potts 1989). Simultáneamente en C.-L. Chen, Vempati y Aljaber (1995) se propone otro algoritmo genético de similares características para el problema del taller de flujo. Se presenta un algoritmo genético híbrido para el mismo problema en Reeves y Yamada (1998) donde se incluye la técnica de re-encadenado de trayectorias, con las siglas PR del inglés path relinking que consigue hacer su nuevo algoritmo genético más competitivo que el anterior. En Murata, Ishibuchi y Tanaka (1996a) se diseña otro algoritmo genético y dos versiones híbridas que contienen búsquedas locales: una normal y otra basada en el recocido simulado. También se propuso una versión de algoritmo genético para multi-objetivo (minimizar el makespan y minimizar la tardanza total) en Murata, Ishibuchi y Tanaka (1996b), en este GA se trabaja con conjuntos de soluciones formando fronteras de Pareto.
En Stützle (1998) se presenta un algoritmo de búsqueda local iterada (Iterated Local Search - ILS) para resolver el problema PFSP y se realiza una extensa calibración de diferentes parámetros. Uno de los parámetros usados es el criterio de aceptación donde se utiliza la misma versión que en SA de usar temperatura constante (Osman y Potts 1989). En Leu y Hwang (2002) se propone un algoritmo genético para el problema del taller de flujo con recursos en una planta mixta de prefabricados, estos son recursos renovables de aceleración. También incluye un análisis de la relación entre los recursos disponibles y el valor objetivo del makespan de forma similar a como

hacen en Daniels y Mazzola (1994). También se plantean algoritmos de colonias de hormigas (Ant Colony Optimization - ACO), el primero fue Rajendran y Ziegler (2004) que propuso dos ACO diferentes y los utilizó para el objetivo de minimizar el makespan y también para el objetivo de minimizar el tiempo total de flujo.

Un algoritmo genético híbrido se propuso en Ruiz, Maroto y Alcaraz (2006) para el problema PFSP con el objetivo de minimizar el makespan. Onwubolu y Davendra (2006) diseñaron un algoritmo de evolución diferencial (Differential Evolution exploration algorithm - DE) para el problema del taller de flujo con permutación enfrentando diferentes objetivos (de forma independiente): minimizar el makespan, minimizar el tiempo de flujo promedio y reducir la tardanza total. Su algoritmo lo comparan con algoritmos genéticos obteniendo mejoras estadísticamente significativas en instancias pequeñas. Otro algoritmo que se presenta para el problema PFSP es el de la optimización por enjambre de partículas de Tasgetiren, Y.-C. Liang, Sevkli y Gencyilmaz (2007) (PSO, del inglés, Particle Swarm Optimization). En este trabajo también se usa como búsqueda local la búsqueda en vecindarios variables (Variable Neighbourhood Search - VND). El problema se estudia para los objetivos de minimizar el makespan y tiempo total de flujo. Para el problema PFSP con objetivo de la tardanza total se realiza un análisis de heurísticas y metaheurísticas que muestra que las metaheurísticas SA dan los mejores resultados (Vallada, Ruiz y Minella 2008).

Se introduce el algoritmo iterativo voraz (Iterated Greedy - IG) en Ruiz y Stützle (2007) para el problema del taller de flujo con permutación y minimización del makespan. El IG es un algoritmo de trayectoria que adopta un mecanismo de destrucción y construcción basado en la búsqueda NEH. Este algoritmo, usando y sin usar búsqueda local, se compara con los mejores algoritmos del momento para el problema PFSP y obtiene los mejores resultados hasta el momento. La versión que integra una búsqueda local resultó mejor. Saravanan, Noorul Haq, Vivekraj y Prasad (2008) propuso un algoritmo de (Scatter Search - SS) para resolver el problema PFSP con el objetivo de makespan, sus resultados mejoran a otros algoritmos en el benchmark de Taillard (Taillard 1993). Ruiz-Torres y Centeno (2008) se enfrentan al problema PFSPR con el objetivo de minimizar el número de trabajos retrasados. Los recursos adicionales reducen el tiempo de procesado y se fijan para todo el horizonte del *schedule*, es decir, son recursos de aceleración estáticos. Para resolver el problema se proponen varias heurísticas, búsquedas locales y una metaheurística (SA). En Figielska, Ewa (2009) se estudia el problema del taller de flujo híbrido con recursos renovables enteros. El taller tiene máquinas paralelas no relacionadas en una etapa y una sola máquina en la segunda. Se proponen metaheurísticas: un *Simulated Annealing* y un *Genetic Algorithm*. La evaluación de la solución se realiza en dos etapas, primero resuelve el problema UPMR de la primera etapa y después coloca los trabajos en la última máquina intentando reducir tiempos ociosos.

Una aproximación diferente a la de aplicar una metaheurística es ejecutar algoritmos en paralelo. Esto se hace en Vallada y Ruiz (2009) con un sistema cooperativo de islas de tal forma que varios hilos de ejecución de la misma metaheurística están funcionando al mismo tiempo, uno en cada isla, de forma que están funcionando en paralelo. Cuando se llega a un punto de la ejecución, estas islas comienzan a permitir pasarse información de soluciones que son buenas mediante mecanismos de inmigración y emigración. Esta técnica se prueba con varios algoritmos genéticos, con un iterativo voraz y un iterativo voraz cooperativo para resolver el problema PFSP con los objetivos de la tardanza total y minimización del makespan. En Minella, Ruiz y Ciavotta (2011) para resolver el problema PFSP multi-objetivo se presenta el algoritmo Pareto iterativo voraz con reinicio multi-objetivo (Restarted Iterated Pareto Greedy algorithm - RIPG). Este algoritmo consigue mejorar los algoritmos usados hasta el momento en una comparación multi-objetivo de parejas de los tres objetivos: minimización del makespan, minimización de la tardanza y minimización del tiempo de flujo. Para el problema PFSP con el objetivo de minimización del tiempo total de flujo se diseña en Tasgetiren, Q.-K. Pan, Suganthan y A. H. Chen (2011) el algoritmo de la colonia de

abejas artificiales y discretas (Discrete Artificial Bee Colony - DABC), integran en el algoritmo una búsqueda local híbrida basada en el algoritmo iterativo voraz. El estudio incluye también un algoritmo evolutivo al que llaman hDDE (hybrid Discrete Differential Evolution). Para ese mismo problema Dong, P. Chen, Huang y Nowak (2013) proponen un algoritmo ILS con un método de multi-reinicio al que llaman MRSILS. Una variante poco estudiada es la de la re-programación de la producción (o *re-scheduling*). En estos problemas los talleres encuentran incidencias que alteran el estado del taller. Un ejemplo lo encontramos en Katragjini, Vallada y Ruiz (2013) donde se estudian varias metaheurísticas state-of-the-art resultando el iterativo voraz (IG) la mejor. Nótese que en estos problemas el objetivo es generar *schedules* que sean resistentes a incidencias. Tosun (2014) presentan un ABC para el problema PFSP con el objetivo de minimizar el makespan y lo comparan con sets de instancias anteriores a las instancias de Taillard. Otra versión del PSO se introduce en Zhang y Wu (2014) para resolver el problema PFSP con el objetivo de minimizar el tiempo total de flujo, este PSO es híbrido usando diferentes métodos como SA, VNS o path relinking.

En Figielska, Ewa (2014) se estudia el problema del taller de flujo híbrido con recursos renovables enteros. El taller solo tiene dos etapas: en la primera etapa hay una sola máquina y en la segunda etapa tiene máquinas paralelas no relacionadas. Se idea una heurística para el problema donde los recursos son compartidos entre ambas etapas. De nuevo resuelve la evaluación en el algoritmo usando dos etapas de forma similar a Figielska, Ewa (2009) aunque la distribución del taller es diferente y los cálculos varían. En Vallada, Ruiz y J. M. Framinan (2015) se presentan nuevas instancias difíciles de resolver para el problema del taller de flujo y el objetivo makespan. Además se estudian las instancias y su dificultad utilizando algunas heurísticas NEH o metaheurísticas como IG (Ruiz y Stützle 2007) o HGA (Ruiz, Maroto y Alcaraz 2006) para demostrar que son instancias especialmente difíciles de resolver. Se consideraban difíciles las instancias que mostraban mayor diferencia entre su cota superior y cota inferior.
En Gonzalez-Neira, Ferone, Hatami y Juan (2017) se utiliza una simheurística para resolver la versión estocástica del problema DAPFSP (distributed assembly permutation flowshop problem). El método propuesto BR-SimGRASP se utiliza para minimizar el *makespan* esperado.

3.4.1 — "Metaheurísticas la metáfora expuesta". Con el título "Metaheurísticas la metáfora expuesta" en Sörensen (2015) se plantean algunos de los problemas que se encuentran en la literatura científica los últimos años, donde el área de estudio de metaheurísticas ha sufrido una explosión de trabajos proponiendo "nuevas metaheurísticas" inspiradas en la naturaleza u otro símil como justificación de una nueva metaheurística. En alguno de estos trabajos se ha demostrado que son técnicas ya conocidas que están rodeadas de toda una explicación basada en una metáfora pero que en realidad no aportan realmente novedad relevante aparte de la propia metáfora. El análisis planteado por Sörensen ha sido apoyado por más investigadores en el reciente análisis de la situación que encontramos en Aranha et al. (2022), donde se explica con más detalle toda la problemática y se proponen acciones concretas para evitar esta mala praxis en el campo de las metaheurísticas.

En Figielska, Ewa (2018) se estudia el problema del taller de flujo híbrido con dos etapas en las que ambas son de máquinas paralelas no relacionadas. En el problema se utilizan recursos adicionales pero en este caso no son renovables. Se propone un algoritmo de generación de columnas (Column Generator - CG) que descompone el problema. En este algoritmo se genera la primera solución con relajación lineal (Linear Relaxation - LR) o con una heurística simple y directa llamada Straight Forward (SF). El problema descompuesto se estudia después con un método de búsqueda tabú (Tabu Search - TS) o un método recocido simulado (Simulated Annealing - SA) para resolver un trozo de la solución. Para el problema DAPFSP (Distributed Assembly Permutation Flowshop Problem), pero ahora no es estocástico, se presenta en Ferone et al. (2020)

un trabajo para optimizar el makespan utilizando un biased-randomized ILS (BR-ILS). Los autores demuestran superar en rendimiento a otras metaheurísticas utilizadas en ese problema. En Laribi, Yalaoui y Sari (2019) se encuentra el trabajo que más se aproxima al problema planteado en esta tesis: donde se resuelve el problema del taller de flujo con permutación y recursos adicionales. La diferencia radica en que en este caso son recursos no renovables, es decir, existe un calendario de liberación de recursos con los instantes en los que los recursos pasan a estar disponibles para ser consumidos. Se plantean adaptaciones de: un modelo MILP, el algoritmo de Palmer, la NEH, la NEH mejorada de Kalczynski y Kamburowski (2008) y finalmente un algoritmo genético (Hybrid Genetic Algorithm - HGA). Todos estos métodos se comparan en una experimentación donde se demuestra que el HGA obtiene los mejores resultados. Aunque una crítica a este trabajo sería que la comparación entre algunos de los métodos no se realiza por tiempo, sino por iteraciones. En Vallada, Villa y Fanjul-Peyro (2019) se resuelve el problema UPMR con metaheurísticas para el objetivo makespan. Partiendo de un trabajo anterior con heurísticas Villa, Vallada y Fanjul-Peyro (2018), se extiende proponiendo una búsqueda dispersa *(Scatter Search - SS)*, una búsqueda dispersa enriquecida *(Enriched Scatter Search - ESS)* y un algoritmo iterativo voraz enriquecido *(Enriched Iterated Greedy - EIG)*.

3.4.2 — El diseño automático/automatizado de algoritmos. En los últimos años ha aparecido un enfoque diferente en el diseño de metaheurísticas que intenta mejorar el diseño de algoritmos. Está técnica se ha usado en muchos problemas como: el problema del taller de flujo con permutación (Pagnozzi y Stützle 2019), el problema del taller de flujo híbrido (Alfaro-Fernández, Ruiz, Pagnozzi y Stützle 2020) o el problema del taller de flujo sin permutación (Brum, Ruiz y Ritt 2022). En este enfoque, al que se hace referencia en algunos trabajos como diseño automático de algoritmos (Automatic Algorithm Design - AAD), las metaheurísticas se consideran como un componente compuesto a su vez de un conjunto de componentes ordenados. Teniendo así diseñadas varias metaheurísticas diferentes, se dispone de un gran conjunto de componentes que en base a unas reglas pueden generar nuevos algoritmos mezclando otros algoritmos conocidos, de tal forma que se asegure que el resultado es un algoritmo válido. Se necesita ahora una especie de meta-algoritmo que genere nuevos algoritmos y los pruebe en problemas concretos generando resultados que apoyándose en alguna herramienta estadística comprueba la calidad de ese nuevo algoritmo. De esta manera se pueden obtener nuevos algoritmos basándose en la extensa literatura científica y demostrando que el nuevo algoritmo propuesto es de alguna forma mejor que los demás y por extensión mejor que los algoritmos planteados inicialmente.

Autores / Año	Tipo de Problema	Algoritmo	Recursos	Objetivo
Osman y Potts (1989)	PFSP	SA	No	C_{max}
Widmer y Hertz (1989)	PFSP	TS	No	C_{max}
Taillard (1990)	PFSP	TS	No	C_{max}
Reeves (1993)	PFSP	TS	No	C_{max}
Daniels y Mazzola (1993)	FRFSP	TS	De aceleración flexibles	C_{max}
Nowicki y Smutnicki (1996b)	PFSP	TS	No	C_{max}
Nowicki y Smutnicki (1996a)	JSP	TS	No	C_{max}
Nowicki y Smutnicki (1998)	HFSP	TS	No	C_{max}
Reeves (1995)	PFSP	GA	No	C_{max}
C.-L. Chen, Vempati y Aljaber (1995)	PFSP	GA	No	C_{max}
Reeves y Yamada (1998)	PFSP	HGA(PR)	No	C_{max}

Continua en la página siguiente

Table 3.3 – continua desde la página anterior

Autores / Año	Tipo de Problema	Algoritmo	Recursos	Objetivo
Murata, Ishibuchi y Tanaka (1996a)	PFSP	HGA (LSySA)	No	C_{max}
Murata, Ishibuchi y Tanaka (1996b)	MOPFSP	HGA	No	C_{max} / $\sum T_j$
Stützle (1998)	PFSP	GA	No	C_{max}
Leu y Hwang (2002)	PFSP	GA	De aceleración renovables	C_{max} y $\sum F_j$
Rajendran y Ziegler (2004)	PFSP	ACO	No	C_{max}
Ruiz, Maroto y Alcaraz (2006)	PFSP	HGA	No	C_{max}
Onwubolu y Davendra (2006)	PFSP	DE	No	C_{max} $\overline{F}$ $\sum T_j$
Tasgetiren, Y.-C. Liang, Sevkli y Gencyilmaz (2007)	PFSP	PSO,VNS	No	C_{max} $\sum F_j$
Vallada, Ruiz y Minella (2008)	PFSP	*	No	$\sum T_j$
Ruiz y Stützle (2007)	PFSP	IG	No	C_{max}
Saravanan, Noorul Haq, Vivekraj y Prasad (2008)	PFSP	SS	No	C_{max}
Ruiz-Torres y Centeno (2008)	PFSP	SA	De aceleración estáticos	$\sum U_j$
Figielska, Ewa (2009)	HFSP	SA, GA	Renovables	C_{max}
Vallada y Ruiz (2009)	PFSP	CGA, CIG	No	C_{max} $\sum T_j$
Minella, Ruiz y Ciavotta (2011)	MOPFSP	RIPG	No	C_{max} / $\sum F_j$ C_{max} / $\sum T_j$ $\sum T_j$ / $\sum F_j$
Tasgetiren, Q.-K. Pan, Suganthan y A. H. Chen (2011)	PFSP	DABC, hDDE	No	$\sum F_j$
Dong, P. Chen, Huang y Nowak (2013)	PFSP	MRS, ILS	No	$\sum F_j$
Katragjini, Vallada y Ruiz (2013)	Re-Sched	-	No	C_{max}
Tosun (2014)	PFSP	ABC	No	C_{max}
Zhang y Wu (2014)	PFSP	HPSO	No	$\sum F_j$
Figielska, Ewa (2014)	HFSP	SA, GA	Renovables	C_{max}
Vallada, Ruiz y J. M. Framinan (2015)	PFSP	-	No	C_{max}
Gonzalez-Neira, Ferone, Hatami y Juan (2017)	DAPFSP	BR-SimGRASP	No	$E[C_{max}]$
Figielska, Ewa (2018)	HFSP	CG - TS, SA	No	C_{max}
Pagnozzi y Stützle (2019)	PFSP	*	No	C_{max}
Ferone et al. (2020)	DAPFSP	BR-ILS	No	C_{max}
Laribi, Yalaoui y Sari (2019)	PFSPR	HGA	No-renovables	C_{max}
Vallada, Villa y Fanjul-Peyro (2019)	UPMR	ESS, EIG	No	C_{max}
Alfaro-Fernández, Ruiz, Pagnozzi y Stützle (2020)	HFSP	*	No	C_{max} $\sum F_j$ $\sum wEwT^{ddw}$

Continua en la página siguiente

Table 3.3 – continua desde la página anterior

Autores / Año	Tipo de Problema	Algoritmo	Recursos	Objetivo
Yepes-Borrero, Villa, Perea y Caballero-Villalobos (2020)	UPMSR-S	GRASP	Renovables en Setup	C_{max}
Yepes-Borrero, Perea, Ruiz y Villa (2021)	BO-UPMSR-S	T-RIPG	Renovables en Setup	C_{max}
Brum, Ruiz y Ritt (2022)	NPFSP	*	No	C_{max}
Yepes-Borrero, Perea, Villa y Vallada (2023)	PFSP-RS	GRASP	Renovables en Setup	C_{max}

Tabla 3.3: Trabajos con metaheurísticas de problemas cercanos a PFSPR

En Yepes-Borrero, Villa, Perea y Caballero-Villalobos (2020) se enfrenta un nuevo problema con recursos adicionales. El problema de máquinas paralelas con tiempos de cambio y recursos adicionales en los tiempos de cambio (unrelated parallel machine scheduling problem with setup times and additional limited resources in the setups - UPMSR-S). Nótese que el nombre UPMSR-S de este trabajo parece igual al problema enfrentado en Fanjul-Peyro (2020), sin embargo la diferencia es que los recursos de este nuevo problema están en el tiempo de cambio, no en el tiempo de procesado. Se presenta un modelo MILP y se proponen dos aproximaciones de construcción, una sin considerar recursos y otra considerándolos, y se proponen 3 algoritmos GRASP que se prueban con ambas aproximaciones. En el estudio se demuestra que la aproximación que ignora recursos es peor que la que los considera. En Yepes-Borrero, Perea, Ruiz y Villa (2021) el problema anteriormente estudiado se modifica haciéndolo bi-objetivo, siendo el objetivo reducir el makespan pero también reducir los recursos consumidos. Al nuevo problema se le conoce por las siglas en inglés BO-UPMSR-S, es el problema bi-objetivo de máquinas paralelas no relacionadas con tiempos de cambio dependientes de la secuencia y recursos renovables adicionales requeridos durante los tiempos de cambio. Para resolver el problema se propone un algoritmo de Pareto iterativo voraz restrictivo y truncado (Truncate Restarted Iterated Pareto Greedy algorithm - T-RIPG) y se compara con otros algoritmos de la literatura como son: MOIGS (Multi-Objective Iteative Greedy Search), RIPG (Restarted Iterated Pareto Greedy algorithm) y NSGA-II (Non-Dominated Sorting Genetic Algorithm II). Obteniendo mejores resultados con el algoritmo propuesto.

En Yepes-Borrero, Perea, Villa y Vallada (2023) se plantea un nuevo problema muy realista, el taller de flujo con permutación con tiempos de cambio que requieren recursos adicionales renovables (PFSR-S - Permutation Flowshop Scheduling problem with additional Resources during Setups). Para resolverlo o explicar el problema se plantean dos modelos matemáticos (MILPs) y un algoritmo exacto de tres fases (TPE). Debido a la enorme complejidad del problema ninguna de estas soluciones era buena en instancias de tamaño medio o grande, por ello se incluye una metaheurística GRASP en dos versiones: GRASP_R y GRASP_NR, siendo el segundo un GRASP en el que los recursos no se consideran en la fase constructiva, dando este peores resultados.

La revisión respecto a metaheurísticas es muy extensa por ello se facilita un resumen de los trabajos más importantes mencionados a lo largo de esta sub-sección en la tabla 3.3, en ella se ve qué trabajo es, qué problema se intenta resolver, cuál es el algoritmo más importante que se aporta como solución, si se necesitan recursos y qué objetivo se enfrenta. En algunos casos aparece un asterisco, en estos casos se presentan muchas soluciones o no se presentan ninguna por lo que no aparecen detalladas en la tabla.

Tras esta revisión bibliográfica exhaustiva, no se ha encontrado ningún trabajo que investigue el problema que se estudia en esta tesis, el problema del taller de flujo con permutación y recursos

adicionales renovables. Sin embargo, si se ha logrado analizar los trabajos referentes a problemas similares, con y sin recursos. De este análisis desprenden algunos apuntes que pueden ayudar a enfrentar el problema con mayores probabilidades de éxito. Uno es que los algoritmos exactos más utilizados, como son los modelos MILPs o los algoritmos B&B, no parecen ser la solución más prometedora para resolver problemas de tamaños realistas con estas características. Aunque el estudio de algún modelo MILP permitiría comprobar que la declaración anterior es válida y al mismo tiempo ayudaría a explicar y definir el problema. Otra es ver que por su sencillez y rapidez muchos trabajos proponen algunas reglas o heurísticas simples enfocadas al problema. Por otra parte la heurística NEH sigue siendo la más utilizada para estos problemas, especialmente, a la hora de iniciar metaheurísticas. Por último, observamos la enorme variedad de metaheurísticas disponibles para resolver estos problemas, donde los algoritmos genéticos híbridos han sido la solución más popular entre los algoritmos con poblaciones y al mismo tiempo búsquedas de trayectoria como SA, TS, ILS o IG, también son muy populares, en muchas ocasiones, ofrecen mayor rendimiento.

4. Heurísticas para el taller de flujo de permutación con recursos adicionales

En este capítulo se proponen diferentes heurísticas constructivas para resolver el problema del taller de flujo de permutación con recursos adicionales (Permutation Flowshop Scheduling Problem with additional Resources -PFSPR-) con el objetivo de minimizar el *makespan*. Tras definir formalmente el problema, se propone un modelo de programación lineal mixta y heurísticas constructivas eficientes para abordar el problema planteado. El objetivo principal de este capítulo es obtener heurísticas eficaces y eficientes para poder ser utilizadas como soluciones de partida de las metaheurísticas propuestas en el capítulo 5.

4.1 Definición formal del problema

Para definir el taller de flujo de permutación de permutación con recursos adicionales, es necesario introducir los siguientes conceptos y notación:

- Se tiene un conjunto de trabajos $N = (1,..., n)$, en total n trabajos que procesar.
- Cada trabajo ha de ser procesado por todas las máquinas del conjunto $M = (1,..., m)$.
- Al ser un taller de flujo de permutación, el orden de procesado de todos los trabajos es el mismo en todas las máquinas.
- A cada procesado en cada máquina de un trabajo lo conocemos como tarea, y un trabajo j tiene m tareas, una tarea por cada máquina.
- Se define p_{ij} como la cantidad de tiempo que tarda en procesarse la tarea del trabajo j en la máquina i.
- Para poder procesar una tarea, es necesario contar con recursos adicionales disponibles que no estén siendo utilizados por otras tareas. Cada tarea requiere una cantidad de recursos adicionales r_{ij} que será diferente para cada máquina y cada tarea. Estos recursos r_{ij} son un valor entero.
- Se requiere disponer de suficientes recursos adicionales para el procesado de las tareas, siendo estos recursos limitados. El límite es un valor entero denominado R_{max}, que son los recursos disponibles para todo el taller y que se mantiene constante.

El objetivo del problema consiste en determinar el orden de los trabajos y el momento de procesado de sus tareas, de tal forma que se minimice el tiempo de finalización máximo (C_{max}), en otras palabras, el objetivo es que el taller acabe lo antes posible de procesar todos los trabajos, cumpliendo, al mismo tiempo, todas las restricciones del problema.

4.1.1 Modelo de programación lineal entera mixta

En esta sección, se propone un modelo matemático que llamamos $PFSPR-Scheduling$, basado en el modelo $UPMR-S$ propuesto en Fanjul-Peyro, Perea y Ruiz (2017) para problemas de máquinas paralelas no relacionadas.

Para formular el problema, se introducen las siguientes variables y parámetros:

- Parámetro p_{ij}: tiempo de procesamiento del trabajo j en la máquina i.
- Parámetro r_{ij}: cantidad de recursos adicionales requeridos para procesar el trabajo j en la máquina i.
- Parámetro $R_{\max}$: cantidad de recursos adicionales disponibles.
- Parámetro K_{max}: marca el horizonte temporal en el modelo para el índice k, es una cota superior. Más adelante se detalla el cálculo de K_{max}.
- Variables binarias x_{ijk}: $x_{ijk} = 1$ si la tarea del trabajo j procesada en la máquina i acaba su procesamiento en el instante de tiempo k. En caso contrario $x_{ijk} = 0$.
- Variable continua C_{max}: tiempo de finalización en la última máquina m del último trabajo.
- Variable binaria $F_{j,g}$: variable con valor 1 cuando el trabajo j se procesa después que el trabajo g, y 0 en caso contrario. Al ser un problema con permutación esto implica que cada una de las tareas i del trabajo j tendrá que procesarse antes que cada una de las tareas i del trabajo g para cumplir esta restricción.

El modelo propuesto es como sigue:

$$Minimize \quad C_{\max} \tag{4.1}$$

$$\sum_{k \geq p_{mj}} k x_{mjk} \leq C_{\max}, \qquad \forall\, j \tag{4.2}$$

$$\sum_{k \geq p_{ij}} x_{ijk} = 1, \qquad \forall\, j,i \tag{4.3}$$

$$\sum_{k \geq p_{ij}} x_{ijk} \leq \sum_{l \geq p_{ij}} x_{i+1,j,l-p_{i+1,j}}\ , \qquad \forall\, j, i \in \{1..m-1\} \tag{4.4}$$

$$\sum_{j} \sum_{s \in \left\{\max\left\{k,p_{ij}\right\},\ldots,k+p_{ij}-1\right\}} x_{ijs} \leq 1, \qquad \forall\, i,k \tag{4.5}$$

$$\sum_{i}\sum_{j} \sum_{s \in \left\{\max\left\{k,p_{ij}\right\},\ldots,k+p_{ij}-1\right\}} r_{ij} x_{ijs} \leq R_{\max}, \qquad \forall\, k \tag{4.6}$$

$$\sum_{i}\sum_{j} x_{ijk} \leq m, \qquad \forall\, k \tag{4.7}$$

$$\sum_{k \geq p_{ij}} k x_{ijk} \leq \sum_{l \geq p_{ig}} l x_{igl} + M F_{jg}, \qquad \forall\, i,j,g, g \neq j \tag{4.8}$$

$$F_{jg} + F_{gj} = 1, \qquad \forall\, j,g, g \neq j \tag{4.9}$$

El objetivo es la minimización del tiempo de finalización máximo, conocido también como *makespan* (explicado en la sección 2.3). Definido por las ecuaciones 4.1 y 4.2, la primera indica la minimización y la segunda marca que ningún trabajo acaba después de C_{max}, ajustando el valor así al tiempo de finalización de la última tarea del último trabajo en la última máquina. La restricción 4.3 fuerza que todos los trabajos se ejecutan una sola vez en cada una de las máquinas. La restricción 4.4 obliga a cumplir la precedencia de tareas en un taller de flujo, para un trabajo j su tarea en la máquina i tiene que estar finalizada antes que su tarea en la máquina $i+1$. La restricción 4.5 asegura que ninguna máquina procesa más de un trabajo a la vez. La restricción 4.6 hace que la suma de los recursos utilizados en cualquier momento no sobrepase la capacidad de recursos máximos R_{max}. La restricción 4.7 fuerza que no hay más máquinas activas de las m máquinas que hay en el taller. La restricción 4.8 valida el orden de los trabajos, haciendo que la variable F_{jg} tome

el valor 0 siempre que el trabajo j preceda al trabajo g en todas las máquinas. La restricción 4.9 combinada con la expresión anterior nos dice que si un trabajo se procesa antes que otro, no se da el caso contrario.

K_{max} se define como un valor entero positivo, una cota superior (Upper Bound), que delimita el horizonte temporal del problema en cuestión.

Para el problema del taller de flujo con permutación y sin recursos (PFSP), se propone la cota superior trivial de la expresión 4.10, esta consiste en encontrar el tiempo de procesado acumulado en la máquina que toma más tiempo total en procesar todos los trabajos ($max\ p_i$), obtener el tiempo de procesado total que necesita el trabajo más costoso ($max\ p_j$) y juntar ambos en un *schedule*. Se ha tenido que añadir la resta del tiempo de proceso de la tarea i en el trabajo j, porque esta tarea se ha sumado dos veces, tanto en la máquina como en el trabajo.

$$max\ p_i + max\ p_j - p_{ij} \tag{4.10}$$

Al tener recursos adicionales esta cota no es válida, a no ser que haya más recursos disponibles de los que se necesitan. Si para el número de máquinas disponibles h se cumple la condición 4.11, que todas las máquinas trabajando simultáneamente en tareas que tienen un requisito de recursos que es el máximo del problema no superan los recursos disponibles (R_{max}), entonces los recursos no influyen en el problema y se puede utilizar la cota que se plantea para (PFSP).

$$max(r_{ij}) \times h \leq R_{max} \tag{4.11}$$

Ahora, si se subdivide el problema con recursos $n \times m$, en varios problemas con menos máquinas. Creando un primer problema con las máquinas m_1 a m_h, un segundo problema con m_{h+1} a $m_{2\times h}$,.... Se resuelve la cota de estos problemas PFSP, y concatenamos el *schedule* (sumamos los valores). El valor resultante será una cota superior para el problema PFSPR.

Para calcular cuantas máquinas h pueden funcionar simultáneamente se utiliza la fórmula 4.12. Los problemas se dividen en problemas con h o menos máquinas, se resuelve la cota PFSP, se suman los valores y se genera una cota superior bastante holgada que se utiliza como K_{max}.

$$h = \left\lceil \frac{R_{max}}{max(r_{ij}) \times m} \right\rceil \tag{4.12}$$

Como se ha explicado anteriormente los modelos matemáticos como este suelen tener muchas limitaciones en la capacidad de resolución, resolviendo a optimalidad problemas de reducido tamaño pero siendo incapaz de resolver problemas de tamaño medio o grande. Por ello es necesario utilizar otros métodos de resolución.

4.2 Heurísticas constructivas basadas en las reglas de despacho

Una heurística es un método que utiliza el conocimiento sobre el problema para construir una nueva solución. Las heurísticas son métodos para la resolución de problemas que han sido muy utilizados en el campo de *scheduling*. Los primeros trabajos para resolver problemas del taller de flujo empezaron con el trabajo de S. Johnson (1954). Desde entonces se han seguido utilizando hasta ahora, aunque tienen bastantes limitaciones, su sencillez las convierte en herramientas inestimables. Una heurística constructiva es aquella que parte de una solución vacía y mediante un proceso iterativo va añadiendo elementos hasta conseguir una solución completa.

En los problemas de *scheduling* una de las heurísticas constructivas más sencillas e intuitivas, que se suele utilizar, tanto en las empresas como en la investigación académica, son las reglas de despacho o reglas de secuenciación. Incluso hoy en día un gran porcentaje de compañías utilizan expertos que con hojas de cálculo y reglas sencillas organizan la programación de la producción (J. M. Framinan, Leisten y Ruiz García 2014).

Como se reporta en el capítulo 3 no se han encontrado trabajos previos en el problema concreto de esta tesis, pero sí se han encontrado múltiples trabajos similares de *scheduling* que aplican este tipo de heurísticas. Por poner algunos ejemplos, en problemas de máquinas paralelas no relacionadas con recursos adicionales (Villa, Vallada y Fanjul-Peyro 2018), también en talleres de flujo híbridos con recursos adicionales (Figielska, Ewa 2014; Figielska, Ewa 2018), en problemas PFSPR con recursos no renovables (Laribi, Yalaoui y Sari 2019) o en el más reciente artículo del problema del taller de flujo con tiempos de cambio que tienen recursos adicionales (Yepes-Borrero, Perea, Villa y Vallada 2023).

En esta tesis doctoral, se han adaptado reglas clásicas como la *Shortest Processing Time* y *Longest Processing Time* (Pinedo 2016), por ser sencillas y eficientes. Se han creado heurísticas que utilizan los recursos para marcar la ordenación como *Shortest Resources Constraints* y *Longest Resources Constraints*. También se han creado las reglas *Shortest Average Area*, *Longest Average Area*, *Shortest Average Ratio* y *Longest Average Ratio*, que utilizan la proporción o relación entre los tiempos de proceso y los recursos para la ordenación. El algoritmo de Palmer se ha adaptado por ser la mejor de entre las heurísticas sencillas y haber sido utilizada antes en problemas de recursos en Laribi, Yalaoui y Sari (2019). Finalmente se ha adaptado la mejor regla de ordenación para la heurística NEH obtenida en Kalczynski y Kamburowski (2008), y se utilizará también en la siguiente sección 4.4.

Por tanto, las siguientes reglas y heurísticas se adaptan o proponen para el problema *PFSPR*:

- **Shortest Processing Time (SPT):** para cada trabajo j se calcula el tiempo total de procesado (p_j), como la suma de los tiempos de proceso de cada una de sus tareas (p_{ij}). La regla SPT secuencia los trabajos en orden no decreciente de p_j.
- **Longest Processing Time (LPT):** solo se diferencia de la anterior en que esta regla secuencia los trabajos en orden no creciente de p_j.
- **Shortest Resources Constraints (SRC):** para cada trabajo j se calcula la cantidad total de recursos adicionales requeridos (r_j), como la suma de los recursos requeridos de cada una de sus tareas (r_{ij}). La regla SRC secuencia los trabajos en orden no decreciente de r_j.
- **Longest Resources Constraints (LRC):** solo se diferencia de la anterior en que esta regla secuencia los trabajos en orden no creciente de r_j.
- **Shortest Average Area (SAA):** para cada trabajo j se calcula el área promedio de acuerdo a la expresión 4.13. La regla SAA secuencia los trabajos en orden no decreciente de AA_j.

$$AA_j = \sum_{1..m} (p_{ij} \times r_{ij})/m \tag{4.13}$$

- **Longest Average Area (LAA):** solo se diferencia de la anterior en que esta regla secuencia los trabajos en orden no creciente de AA_j.
- **Shortest Average Ratio (SAR):** para cada trabajo j se calcula el ratio promedio de acuerdo a la expresión 4.14. La regla SAR secuencia los trabajos en orden no decreciente de AR_j.

$$AR_j = \sum_{1..m} (p_{ij}/r_{ij})/m \tag{4.14}$$

- **Longest Average Ratio (LAR):** solo se diferencia de la anterior en que esta regla secuencia los trabajos en orden no creciente de AR_j.

• **Slope Index Heuristic (Palmer):** el algoritmo de Palmer (1965) es una heurística que da prioridad a los trabajos con poco tiempo de proceso en sus primeras tareas y mucho tiempo de proceso en las últimas tareas. Para hacerlo genera el índice de la pendiente o *slope index*, que se define en la expresión 4.15 y prepara la secuencia de los trabajos en orden no creciente de SI_j.

$$SI_j = \sum_{1..m} ((2i - m - 1)/2) \times p_{ij}\,, \qquad j = (1,...,n) \tag{4.15}$$

• **Kalczynski And Kamburowski rule (KK1):** esta regla fue propuesta para la ordenación inicial de la heurística NEH en Kalczynski y Kamburowski (2008) que se explicará en detalle en la sección 4.4. Utilizando las fórmulas 4.16 y 4.17 se generan los índices $\hat{a}_j$ y $\hat{b}_j$ para cada trabajo. Así el índice $\hat{a}_j$ da más peso a las primeras tareas, mientras que $\hat{b}_j$ da más peso a las últimas. Con estos dos índices se genera el tercero c_j que es el mínimo de ambos para cada trabajo, con este último índice se genera una lista que se ordena de forma no creciente para darnos la secuencia u orden de los trabajos.

$$\hat{a}_j = \sum_{1..m} ((m-1)(m-2)/2 + m - i) \times p_{ij}\,, \qquad j = (1,...,n) \tag{4.16}$$

$$\hat{b}_j = \sum_{1..m} ((m-1)(m-2)/2 + i - 1) \times p_{ij}\,, \qquad j = (1,...,n) \tag{4.17}$$

En ocasiones surgen los empates, por lo que es necesario decidir qué trabajo se secuencia primero, siendo que dos o más trabajos tienen la misma prioridad. A esta decisión se la conoce como desempate o *tie break* en inglés. Para tomar esta decisión se utilizan o el acumulado de los tiempos de procesado de todas sus tareas (p_j), o la suma de todos los recursos adicionales necesarios (r_j). Además, se puede desempatar eligiendo el menor o el mayor valor, representando menor como t de *true* y mayor por la f de *false*. Así la notación de la heurística $LARr_t$ será la heurística LAR cuando se utilizan los recursos para desempatar y por ello contiene la "r" y se elige el menor valor de recursos para desempatar y por ello tiene el subíndice "t". En la tabla 4.1 se muestran todas las versiones de las reglas que se han analizado dependiendo del criterio utilizado para desempatar. En las reglas SPT, LPT, SRC y LRC que utilizan uno de los requisitos para la ordenación, no se puede desempatar usando el mismo valor por lo que se usa el requisito opuesto. Sin embargo, en las otras cuatro reglas se probarán las cuatro posibilidades de la tabla 4.1, que se obtienen con las opciones (r, p) separadas en dos columnas y separadas en dos filas (t, f).

	Tiempos de proceso (p)	Recursos (r)
Menor (t)	SRC_t , LRC_t $SAAp_t$, $LAAp_t$ $SARp_t$, $LARp_t$	SPT_t , LPT_t $SAAr_t$, $LAAr_t$ $SARr_t$, $LARr_t$
Mayor (f)	SRC_f , LRC_f $SAAp_f$, $LAAp_f$ $SARp_f$, $LARp_f$	SPT_f , LPT_f $SAAr_f$, $LAAr_f$ $SARr_f$, $LARr_f$

Tabla 4.1: Tabla con los posibles desempates.

Estas reglas y heurísticas solo producen un orden o secuencia, también conocido como permutación, este es el orden en el que los trabajos serán procesados por el taller. Para transformar esta secuencia en una solución completa y factible se requieren tres acciones: por una parte, determinar en qué instante de tiempo se inicia y finaliza cada tarea de cada trabajo en cada una de las máquinas. Por otra parte garantizar la factibilidad de la solución comprobando que se cumplen todas las restricciones del problema. Por último, el cálculo del valor objetivo, es decir, la evaluación de la solución.

4.3 Evaluación de la solución

La resolución de problemas de programación con reglas, heurísticas o metaheurísticas normalmente requiere la utilización de una versión esquemática de la solución denominada representación de la solución. La representación más utilizada es una una lista con la secuencia de los trabajos (Fernandez-Viagas, Perez-Gonzalez y J. M. Framinan 2019). Esta lista marca el orden en el que los trabajos son procesados en el taller. Mediante métodos, esta representación se transforma en una solución en la que cada tarea de cada trabajo tiene un inicio y un fin en su tiempo de procesamiento. El proceso de transformación de la representación de la solución en una solución concreta es conocido como evaluación y es un método generalmente determinista que organiza y asigna todos los recursos para llevar a cabo todos los trabajos.

En un taller de flujo de permutación (PFSP), sin recursos adicionales, la transformación desde la secuencia a una solución concreta es bastante directa, no existen decisiones intermedias. La secuencia se mantiene constante en todas las máquinas, y con objetivos como el *makespan* la generación de la solución real a partir de una secuencia es directa.

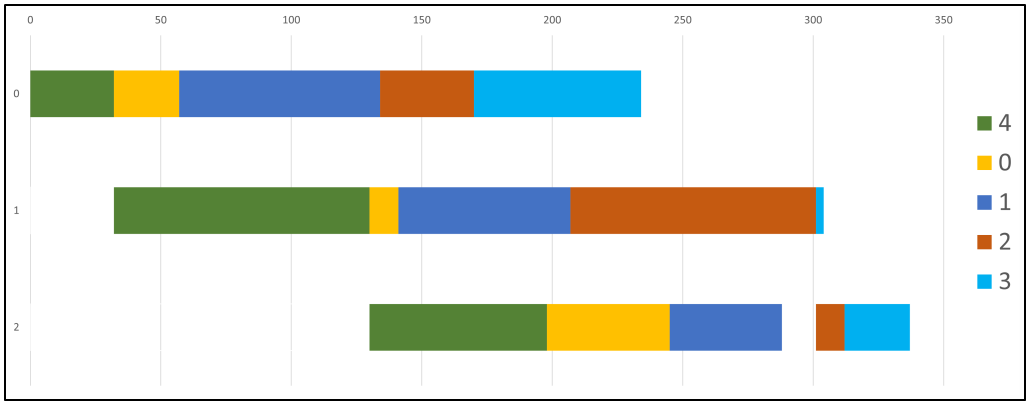


Figura 4.1: Diagrama de Gantt de una solución PFSP con la secuencia $J = \{4,0,1,2,3\}$

Para realizar la evaluación del problema PFSP (problema sin recursos adicionales) con el objetivo *makespan*, se pueden calcular los tiempos inicio y final para cada trabajo, una máquina tras otra. También se pueden calcular los tiempos inicio y final en cada máquina, un trabajo tras otro. En ambos casos el resultado será la misma solución y el mismo *makespan*. En la Tabla 4.2 se presentan los datos de entrada para el Ejemplo 1 que es un taller de flujo de permutación con cinco trabajos, tres máquinas y un recurso adicional. En la sección izquierda están los tiempos de proceso (p_{ij}), mientras que en la sección derecha se ilustra el consumo de recursos para cada tarea en cada máquina (r_{ij}). Dada la siguiente secuencia de trabajos $J = \{4,0,1,2,3\}$ se realiza una evaluación para el problema PFSP sin recursos. La solución resultante de esa evaluación se observa en el diagrama de Gantt de la figura 4.1 (esta secuencia es una solución óptima para esta instancia del problema PFSP). En la solución se aprecia que para la evaluación de esa secuencia se ha colocado cada trabajo y tarea en el primer instante disponible, sin insertar tiempos ociosos innecesarios y comenzando cada tarea lo antes posible.

Esta situación cambia notablemente cuando se trabaja sobre el problema del taller de flujo de permutación con recursos adicionales (PFSPR).

Supongamos el mismo ejemplo 1, con el mismo *schedule* pero ahora considerando los recursos adicionales de la tabla 4.2. En esas circunstancias, la utilización de recursos adicionales en el taller en cada instante de tiempo sería de la forma que aparece en la figura 4.2; si se dispone de un máximo de 18 unidades de recurso ($R_{max} = 18$) se observa como los recursos necesarios sobrepasarían los recursos disponibles en algunos instantes de tiempo y esto haría que la solución planteada no fuese factible para el problema PFSPR.

Tiempos de proceso (p_{ij})						Recursos requeridos (r_{ij})					
	J_0	J_1	J_2	J_3	J_4		J_0	J_1	J_2	J_3	J_4
M_0	25	77	36	64	32	M_0	3	9	6	4	2
M_1	11	66	94	3	98	M_1	3	8	7	1	9
M_2	47	43	11	25	68	M_2	6	7	1	1	9

Tabla 4.2: Tiempos de proceso de un taller de flujo con 3 máquinas, 5 trabajos y consumo de recursos.

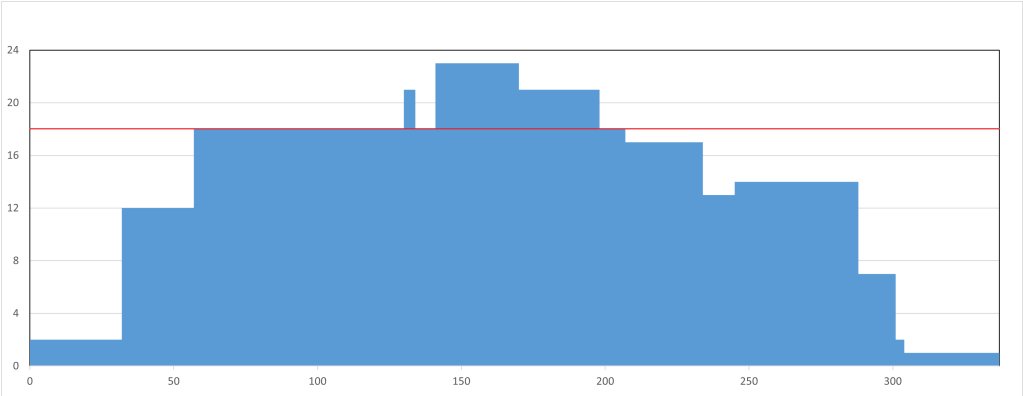


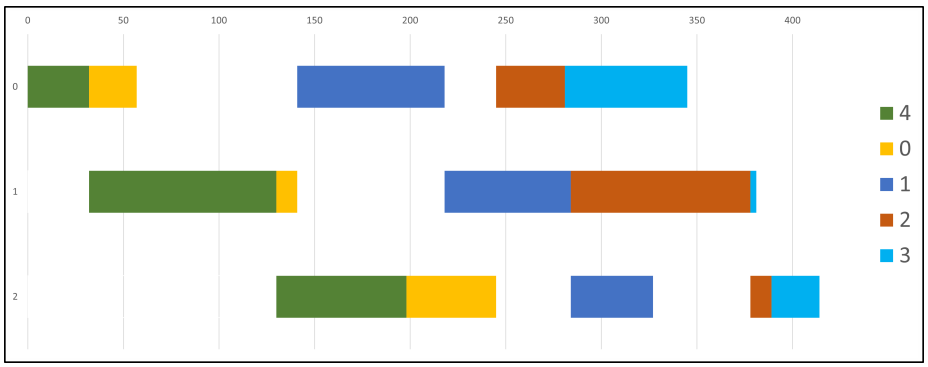
Figura 4.2: El diagrama de áreas del uso de recursos para la figura 4.1 (ejemplo 1)

En el proceso de una evaluación para PFSPR, en muchas ocasiones un trabajo o una tarea tiene que esperar a tener recursos adicionales disponibles para poder ser procesada. Esta situación obliga a tomar decisiones durante la evaluación que afectarán considerablemente al valor objetivo.

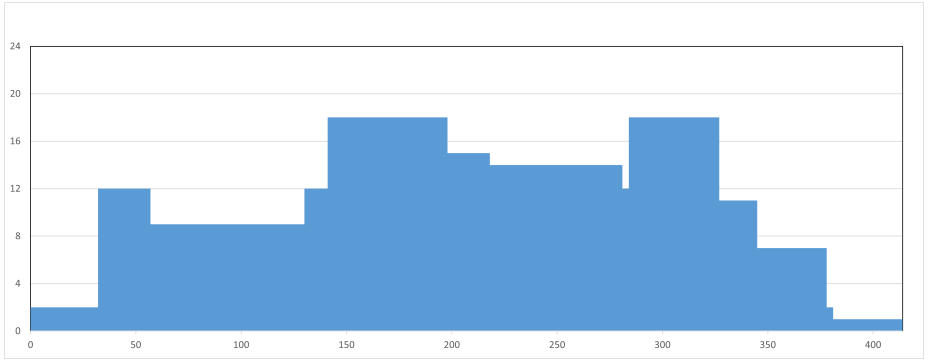
Se han realizado pruebas con las dos formas de decisión en la evaluación.

- La primera consiste en **secuenciar por trabajo**, se coloca cada una de las tareas de un trabajo hasta acabar y entonces se empieza con el siguiente trabajo. Cuando no se dispone de suficientes recursos para procesar una tarea, se inserta un tiempo ocioso hasta tener suficientes recursos. La evaluación por trabajos del ejemplo 1 de la tabla 4.2 se puede ver en las figuras 4.3a y 4.3b. En la primera aparece el diagrama de Gantt con los tiempos de proceso y se observa un tiempo ocioso entre los trabajos 0 y 1, que justamente se da en la primera tarea provocado por la falta de recursos. También se observa un tiempo ocioso entre los trabajos 1 y 2, sin embargo, este tiempo ocioso no impacta en el camino crítico. El *makespan* es 414, mayor que el que se obtendría si no se consideran los recursos adicionales que sería 337.
- La segunda forma de evaluar es **secuenciando por máquina**, así se secuenciaría la primera tarea de todos los trabajos, después la segunda, etc. Calculando en orden los tiempos inicio y fin de procesado de cada máquina. La falta de recursos en este caso no se da en las primeras máquinas sino que suele darse en las últimas. Estos tiempos ociosos en las últimas máquinas tienen un gran impacto en el valor objetivo. La solución resultante de la evaluación por máquina se muestra en las figuras 4.3c y 4.3d. El resultado es una solución que obtiene un *makespan* de 428, peor que la obtenida por trabajo, esto se debe al tiempo ocioso incluido en la tercera tarea del cuarto trabajo.

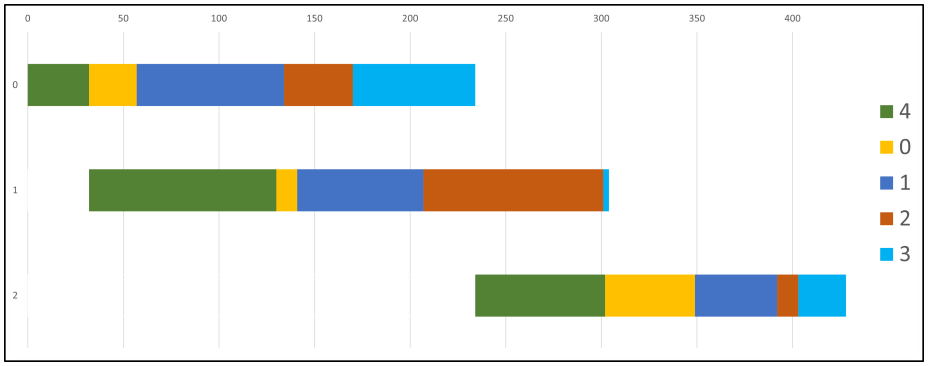
En múltiples ocasiones se encuentra este problema, donde el *schedule* obtenido al evaluar por máquinas introduce muchos tiempos ociosos en las últimas máquinas y como resultado la solución acaba obteniendo peor valor de makespan respecto al obtenido evaluando por trabajo. Por esta razón y tras algunos experimentos que nos permitieron constatar que la forma de evaluación por trabajos obtiene mejores resultados, se ha utilizado esta forma de evaluación a lo largo de esta tesis, tanto en heurísticas como metaheurísticas.



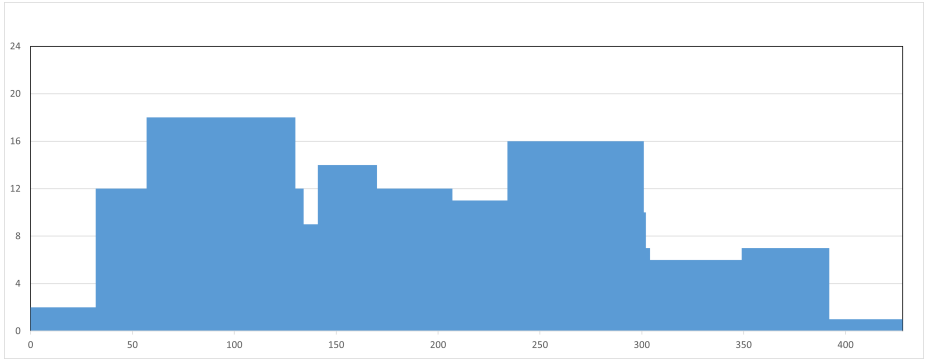
(a) Diagrama de Gantt de una solución PFSP, con secuenciación por trabajo. $J = \{4,0,1,2,3\}$, con $C_{max} = 414$



(b) Diagrama recursos utilizados en base al tiempo, con secuenciación por trabajo, $R_{max} = 18$



(c) Diagrama de Gantt de una solución PFSP, con secuenciación por máquina. $J = \{4,0,1,2,3\}$, con $C_{max} = 428$



(d) Diagrama recursos utilizados en base al tiempo, con secuenciación por máquina, $R_{max} = 18$

Figura 4.3: Comparación del resultado de la evaluación con secuenciación por trabajo (a,b) y la secuenciación por máquina (c,d)

4.4 Heurísticas basadas en la NEH

Entre los constructivos heurísticos destaca el método NEH, nombrado en base a sus creadores Nawaz, Enscore Jr y Ham (1983). Esta heurística es muy utilizada para problemas del taller de flujo (J. M. Framinan, Leisten y Ruiz García 2014; Kalczynski y Kamburowski 2008). La NEH parte de una solución vacía que poco a poco se va construyendo, añadiendo un elemento a la solución en cada iteración. Utiliza una lista de todos los trabajos con una ordenación inicial, esta es la lista ordenada de trabajos pendientes (ϕ) e indica en qué orden se considera la inserción de los trabajos. En cada iteración se prueba a insertar el primer elemento de la lista de trabajos pendientes en todas las posibles posiciones de la solución parcial, finalmente fija el elemento en la posición en la que se obtiene un mejor valor objetivo y elimina el trabajo de la lista de pendientes.

Desgraciadamente la heurística NEH y sus diferentes versiones han demostrado una capacidad limitada en problemas realistas con recursos (Laribi, Yalaoui y Sari 2019; Villa, Vallada y Fanjul-Peyro 2018), debido principalmente a tres razones:

1. La evaluación de soluciones con recursos adicionales es muy costosa en tiempo de computación.
2. La evaluación con recursos es más compleja y no permite adaptar las aceleraciones que se pueden emplear en el problema base, estas aceleraciones reducen mucho el tiempo de cómputo (Taillard 1990).
3. Los algoritmos basados en NEH sin aceleraciones crecen en tiempo de ejecución en base al orden: $O(m \times n^3)$.

Por ello, de forma similar a lo ocurrido en otros problemas, las heurísticas basadas en la NEH no son capaces de resolver problemas grandes en un tiempo razonable.

Se conocen aceleraciones para la heurística NEH en problemas PFSP (Taillard 1990); sin embargo, hasta donde se tiene conocimiento, no han sido adaptadas con éxito en problemas similares que involucren recursos adicionales. Dado que estos recursos adicionales deben ser tenidos en cuenta, las aceleraciones previamente conocidas no se consideran viables para este tipo de problemas.

Por estas razones, ya que se busca poder resolver problemas de todos los tamaños, especialmente de los benchmark actuales, una NEH normal con recursos no podrá resolverlos en un tiempo aceptable. Se facilitarán más detalles sobre estos problemas en la sección 4.5.1 de instancias de problemas, pero se ha comprobado que la ejecución de una NEH con recursos con una instancia grande podría llegar a necesitar varios días de cómputo.

Como queda acreditado en la bibliografía, la NEH es una heurística potente y extendida en los problemas de *scheduling*. Por lo tanto, se considera importante su implementación para este problema, con los cambios y adaptaciones necesarios para que nos permita obtener una buena solución factible al problema PFSPR. Entre los retos que se han de abordar para ello están el problema del tiempo, y si este se resuelve parando por tiempo, cómo gestionar el hecho de no tener una solución completa, o si se finaliza con una solución que no es factible. Para delimitar el tiempo disponible de los métodos se utiliza un **criterio de parada por tiempo**, en este caso será variable respecto al tamaño de la instancia. El criterio de parada que denominaremos criterio de parada (CP) se define en la formula 4.18, siendo ρ un parámetro de entrada:

$$CP = \frac{n \times m}{2} \cdot \rho \ (ms) \tag{4.18}$$

Para las heurísticas basadas en la NEH se ha fijado ρ a 90 ms. Por ejemplo, en el caso de una instancia de 300 trabajos con 60 máquinas con un ρ de 90 ms tendremos 810.000 milisegundos, esto es, 13,5 minutos. Criterios de parada iguales o similares se han empleado en muchos trabajos (Alfaro-Fernández, Ruiz, Pagnozzi y Stützle 2020; Fanjul-Peyro y Ruiz 2010; Vallada, Villa y Fanjul-Peyro 2019), este criterio permite que los métodos tengan una cantidad de tiempo variable

respecto a la instancia lo que evita parcialmente la gran disparidad de tiempo de cómputo necesaria entre las instancias en el proceso de evaluación. Además, se puede utilizar el parámetro ρ con diferentes valores para estudiar el comportamiento de algunos métodos respecto al tiempo, esto es muy útil por ejemplo en las metaheurísticas.

Se proponen las siguientes heurísticas basadas en la NEH:

1. NEH cut and fill
La NEH cut and fill (cortar y rellenar), identificada como NEH_{cf}, se detalla en el algoritmo 4.4.1. Esta es la primera adaptación propuesta. Utiliza una NEH que considera el tiempo y que una vez acabado el tiempo de cómputo máximo asignado para los cálculos de la NEH, y completa la solución parcial con los trabajos pendientes. Los trabajos pendientes se agregan al final de la solución parcial en el orden que ofrece la regla de ordenación y a continuación se evalúa con recursos para obtener una solución completa y factible.

```
Procedure Constructor NEH cut & fill
begin
    // Obtenemos el orden de trabajos pendientes con una regla
    φ = OrderingRule()
    π'_empty = φ_[0]
    while NotTimeout & φ not empty do
        TryAllPositions(π', φ_next)
        π' = InsertInBestPosition(π', φ_next)
    end
    if NEH timeout then
        // En caso de no llegar a completar la NEH paramos la ejecución
           y añadimos los trabajos pendientes.
        π' = ConcatenatePermutations(π', φ)
    return π'
end
```

Algoritmo 4.4.1: Pseudocódigo del constructor heurístico NEH_{cf}.

```
Procedure Constructor NEH abortion
begin
    // Obtenemos el orden de trabajos pendientes con una regla
    φ = OrderingRule()
    // Resolvemos la regla
    π = Evaluation(φ)
    π'_empty = φ_[0]
    while NotTimeout & φ not empty do
        TryAllPositions(π', φ_next)
        π' = InsertInBestPosition(π', φ_next)
    end
    if NEH timeout then
        // En caso de no llegar a completar la NEH devolvemos la regla
        return π
    return π'
end
```

Algoritmo 4.4.2: Pseudocódigo del constructor heurístico NEH_a.

2. NEH abortion
Esta versión NEH abortion (NEH_a) es una NEH que considera los recursos en sus evaluaciones y también utiliza un criterio de parada para gestionar el problema del tiempo. La NEH_a se presenta en el algoritmo 4.4.2. Al comienzo del algoritmo se aprecia una diferencia entre esta NEH y la anterior, la regla de ordenación se utiliza para obtener el orden de los trabajos pendientes, pero además se almacena en π la solución evaluada de la secuencia obtenida con la regla de ordenación (línea 2 del algoritmo 4.4.2). Esta solución no se utiliza a no ser que se acabe el tiempo de cómputo y se dispare el criterio de parada, si esto pasa, la solución parcial (π') es descartada y se utiliza la solución evaluada de la regla de ordenación (π).

3. NEH con reparación
Por último, se propone la llamada NEH con reparación (NEH_{nr}) recogida en el algoritmo 4.4.3. Esta NEH clásica, obtiene la solución para el problema del taller de flujo de permutación sin recursos y una vez obtenida se hace factible añadiendo los recursos y previsiblemente aumentando bastante el *makespan*. Ahora bien, se busca poder comprobar cuál es la diferencia en tiempo de cálculo, comprobar si es una solución buena para el problema base y sigue manteniendo parte de sus características al añadir recursos y por último, se deben calcular y cuantificar las diferencias. Como se verá en la experimentación, un método peor pero mucho más rápido puede ser muy interesante como parte de la propuesta de metaheurísticas que se presentan en el capítulo 5.

```
Procedure Constructor NEH con reparación
begin
    // Obtenemos el orden de trabajos pendientes con una regla
    π = Evaluation(φ)
    π'_empty = φ_[0]
    while φ not empty do
        TryAllPositions(π', φ_next)
        π' = InsertInBestPosition(π', φ_next)
    end
    // Se repara la solución
    π'' = Factibilize(π')
    return π''
end
```

Algoritmo 4.4.3: Pseudocódigo del constructor heurístico NEH_{nr}.

El algoritmo NEH ha sido utilizado en múltiples problemas de *scheduling* y *scheduling* realista. En Villa, Vallada y Fanjul-Peyro (2018) se resuelven problemas de máquinas paralelas no relacionadas con recursos adicionales (*UPMR*), en problemas de taller de flujo con permutación con recursos no renovables Laribi, Yalaoui y Sari (2019) aplica NEH a un problema similar donde los recursos son perecederos. En este caso concreto, utilizan la versión del mismo basado en las mejoras de Kalczynski y Kamburowski (2008), este método lo denotamos como NEH_{kk1}.

Un factor importante para el uso de esta heurística es encontrar qué regla o heurística emplear para generar la lista ordenada de trabajos pendientes. Para poder conocer que sistema permite obtener la mejor ordenación, se comparan todas las reglas y heurísticas constructivas explicadas en la sección 4.2 combinadas con las tres versiones de la NEH.

4.5 Análisis computacional

En el ámbito del análisis de grandes experimentos y para las comparaciones del rendimiento de variables objetivo, como el *makespan*, se recomienda recurrir a una métrica estandarizada y normalizada.

Se emplea la métrica más utilizada, la desviación porcentual relativa (*Relative Percentage Deviation*), abreviada como RPD (J. M. Framinan, Leisten y Ruiz García 2014). La fórmula RPD que se muestra en la expresión 4.19 produce un valor para cada tratamiento, siendo un tratamiento una ejecución singular de alguno de los métodos estudiados o soluciones (subíndice s) con una instancia del problema específica (subíndice i). En la fórmula se utilizan también los valores de la función objetivo (VFO), en nuestro caso el *makespan*. Así restamos al makespan de este tratamiento concreto el makespan de la mejor solución conocida para esa misma instancia y se divide esta diferencia por ese mismo valor de la mejor solución. Así, si un método consigue la mejor solución del conjunto para una instancia, el RPD de esa ejecución será 0.

$$RPD_{s,i} = \frac{VFO_{s,i} - VFO_{mejor,i}}{VFO_{mejor,i}} \cdot 100 \tag{4.19}$$

Será necesario poder validar y comparar la eficiencia de los métodos propuestos. Para ello se requiere un *Benchmark*, es decir, uno o varios sets de instancias del problema PFSPR, donde cada instancia es un problema concreto con datos diferentes. El *Benchmark* deberá ser variado para garantizar que son métodos robustos y poder analizar su comportamiento en entornos diferentes.

4.5.1 Benchmark

En los problemas de talleres de flujo se utilizan habitualmente dos sets de instancias. En primer lugar, el tradicional set de instancias de Taillard (1993), que es el más utilizado. En segundo lugar, se utilizan los sets de instancias más recientes de Vallada, Ruiz y J. M. Framinan (2015), al que nos referiremos por las iniciales de sus autores VRF, donde se propone un Benchmark con instancias de hasta 800 trabajos y 60 máquinas, diseñadas para ser difíciles de resolver con metaheurísticas.

Estos benchmarks se crearon utilizando generadores de números aleatorios para asignar el tiempo de procesado de cada tarea, así que existe mucha variabilidad. En la tabla 4.3 se muestran el set de Taillard y el set VRF, separado en instancias grandes (VRFL) y pequeñas (VRFS). Además, se incluye un pequeño set de instancias para modelos que se explicarán más adelante. En la segunda y tercera columnas aparecen dos factores de instancia, a estos, hay que añadir la réplica de instancia (*rep*) que es el número de creaciones que se han hecho con cada combinación de factores de instancia. Así, el número de trabajos a procesar n, el número de máquinas del taller m, junto con la réplica serían los factores de instancia utilizados por Taillard (1993) y Vallada, Ruiz y J. M. Framinan (2015). En ambos sets se generaron 10 réplicas por cada combinación de $n \times m$. Los sets de instancia de Taillard no son ortogonales, sin embargo los de Vallada si, esto quiere decir que Taillard no tiene todas las posibles combinaciones de $n \times m$.

Set	Nº de trabajos	Nº de máquinas	Ortogonal	Nº de instancias	Con recursos
Taillard	{20, 50, 100, 200, 500}	{5, 10, 20}	No	120	480
VRF Small	{10, 20, ... 60}	{5, 10, 15, 20}	Si	120	480
VRF Large	{100, 200, ... 800}	{20, 40, 60}	Si	120	480
Modelos	{5, 7, 9}	{3, 4, 5}	Si	NA	360

Tabla 4.3: Características de los sets de instancias.

Para probar todos los métodos propuestos adaptaremos estas instancias de taller de flujo al problema de taller de flujo de permutación con recursos adicionales. La adaptación consiste en

añadir la tabla de recursos requeridos (r_{ij}) y un número de recursos máximos disponibles ($R_{\max}$). Los principios que se han utilizado para añadir recursos adicionales en instancias de PFSPR están basados en los que proponen Fanjul-Peyro, Perea y Ruiz (2017) para máquinas paralelas no relacionadas. Se utilizan los factores de instancia específicos para recursos que se van a explicar a continuación.

Recursos requeridos (r_{ij}): Los recursos requeridos para el procesado de cada tarea (r_{ij}), es un valor del intervalo [1,9] que tiene cada tarea del taller, este valor se calcula de dos formas diferentes en base al factor de instancia "Distribución" (Dis). Este factor indica si en la instancia se aplica una cierta correlación entre los valores p_{ij} y r_{ij}, para reflejar que en muchos ámbitos industriales los trabajos más costosos en tiempo también suelen necesitar más recursos. Se generan instancias con y sin correlación y se usan los términos *cor* y *uni*, respectivamente, para diferenciarlos. Para las instancias no correlacionadas los recursos requeridos (r_{ij}) serán valores entre 1 y 9, generados de manera aleatoria. Se obtienen con la expresión 4.20 generando números aleatorios con una distribución uniforme. Para las instancias con correlación se usa la expresión 4.21, donde la primera parte de la expresión genera un valor correlacionado al que se le aplica un modificador aleatorio $U(3,-3)$ generando así un valor parcialmente correlacionado. Se mantendrá r_{ij} como un valor en el intervalo $[1,9]$ y en caso de salir de este, se truncará a 1 o 9.

Recursos disponibles ($R_{\max}$): hay un número máximo de recursos disponibles durante todo el horizonte temporal, denotado como $R_{\max}$. Para calcular la cantidad concreta para una instancia se calculan los recursos disponibles base ($R'_{\max}$) con la expresión 4.22, después se aplica el factor de corrección de disponibilidad (RFactor), que permite ajustar la disponibilidad de recursos. Ajustando la disponibilidad de recursos a dos valores, uno bastante restrictivo y otro menos restrictivo, en la experimentación posterior esto nos permite comprobar el rendimiento de los métodos dependiendo de la disponibilidad de recursos. Tras experimentaciones previas con varios factores (0,5, 0,6, 0,7, 0,8, 0,9, 1 y 1,1), se consideró que 0,9 resultó ser un valor holgado pero que afectaba al makespan, en contraposición se eligió el valor 0,6 ya que tenía un gran impacto en el makespan.

$$r_{ij} = U(r_{\min}, r_{\max}) \qquad donde \quad r_{\min} = 1,\ r_{\max} = 9 \tag{4.20}$$

$$r_{ij} = \lfloor \frac{p_{ij} - p_{\min}}{d} \rfloor + r_{\min} + U(3,-3) \quad donde \quad r_{\min} = 1,\ p_{\min} = 1,\ d = 98/9 \tag{4.21}$$

$$R_{\max} = RFactor \times R'_{\max} \quad donde \quad R'_{\max} = \max\left\{ \lceil \left(\frac{r_{\min} + r_{\max}}{2} \right) \times m \rceil, r_{\max} \right\} \tag{4.22}$$

Al tener los dos factores de instancia con recursos (*RFactor* y *Distribución*), y cada uno de ellos con dos niveles, se obtendrán 4 instancias por cada instancia del Benchmark original. Para explicar cómo crece el número de instancias, se comenta usando el set *VRF small* como ejemplo, el set *VRF large* tendrá los mismos valores. El set *VRF small* original tiene 6 opciones para n, 4 opciones para m y 10 réplicas, así que se obtienen $6 \times 4 \times 10 = 240$ instancias. A cada una de estas instancias se le añaden los factores nuevos: factor de recursos $0,6$ y $0,9$ y el factor de distribución uniforme o correlacionada, resultando en $240 \times 2 \times 2 = 960$ instancias nuevas. Teniendo ya estos sets de instancias completos se suele utilizar todo el grupo de las instancias para test y un subgrupo de estas para entrenamiento. Para VRFS y VRFL, al disponer de tantas instancias se seleccionan 5 réplicas para generar el set de test y una réplica independiente para el set de entrenamiento. Así

tendremos 480 instancias de test y 96 instancias de entrenamiento para cada set VRF. Para Taillard sacamos una instancia de cada combinación teniendo 480 instancias de test y 48 de entrenamiento.

Por último, se ha generado un nuevo set de instancias muy pequeñas al que nos referimos como set "Modelos". Este set se utiliza exclusivamente en los modelos matemáticos, ya que los modelos MILPs no serán capaces de resolver problemas con recursos que no sean extremadamente pequeños. En este caso, los tiempos de proceso se han generado con números aleatorios y una distribución uniforme ($U(1,99)$). En la tabla 4.3, se observa que en este set tenemos 3 valores para n y tres valores para m. Además, se incluyen 2 valores para $RFactor$ y las dos posibles distribuciones. Para cada una de las combinaciones posibles de estos factores de instancia se generan 10 réplicas, dando un total de 360 instancias que ya incluyen recursos. Como este set ha sido generado automáticamente para instancias nuevas en la columna "Nº de instancias", que representa las instancias originales sin recursos se pone N.A. (No Aplica).

4.5.2 Resultados de los experimentos

En esta sección se comparan los resultados de los experimentos de todas las reglas y heurísticas propuestas en este capítulo. Se comentan los resultados del modelo MILPs, las heurísticas constructivas y las diferentes versiones de las heurísticas basadas en *NEH* para acabar comparando la eficiencia y eficacia de todas ellas.

4.5.1 — "Experimentación comparando Algoritmos". Sobre las comparaciones entre heurísticas. Para hacer posible una comparación de heurísticas u otros métodos normalmente se utiliza el "diseño de experimentos" (Design Of Experiments - DOE). El DOE es un método para planificar y ejecutar experimentos de manera eficiente (Montgomery 2012). Se requiere de un cuidadoso trabajo en el diseño de una buena experimentación y un análisis correcto para poder interpretar el comportamiento de la variable respuesta dependiendo de los valores de un conjunto de factores o variables de entrada.

Antes de avanzar en la explicación del DOE, los conceptos de factor, nivel y variable respuesta han de ser explicados:

- Factor: Son las variables que se van a estudiar en el DOE, en una comparación entre algoritmos, el factor principal será el algoritmo en sí. Pero suelen tener en cuenta los factores de instancia, y el tiempo disponible cuando existen varias opciones.
- Nivel: Representan los diferentes valores que cada factor puede tener. Pueden ser cuantitativos o cualitativos. Por ejemplo, en el caso de una comparación de heurísticas, el principal factor es cada una de esas heurísticas.
- Variable respuesta: La variable respuesta es el RPD, que se ha definido en la sección 4.5.

Para las comparaciones y los experimentos se ha seguido un diseño de experimentación *full factorial* o factorial completo, esto es, que se van a ejecutar todas las combinaciones de los valores de los factores posibles. Un tratamiento en este experimento será cada ejecución de un algoritmo contra una instancia concreta. Esta experimentación ahora ha de analizarse, para ello seleccionamos el análisis de la varianza (ANOVA), que descompone la variabilidad de la variable respuesta entre la parte atribuible a los factores y el error (la parte atribuible simplemente a la aleatoriedad). Gracias a esto podemos ver que factores tienen un impacto significativo en la variable respuesta. Para el caso de las heurísticas basadas en la NEH necesitaremos añadir el factor de tiempo ρ pero como es simplemente un tope de tiempo, no se estudiarán más valores, solo 90.
En estos análisis añadimos los factores de instancia (n, m, Dis y $rFactor$), y así analizar las interacciones que puedan tener un impacto significativo.

Es importante destacar que se ha realizado el correspondiente análisis de residuos para verificar que las hipótesis del ANOVA se cumplían siempre que se ha utilizado el análisis de la varianza (ANOVA) para analizar la significatividad estadística de los resultados. Todos los análisis ANOVA se han hecho utilizado la herramienta "R Tools" de Visual Studio (RTVS) que utiliza el motor "Microsoft R Open" 4.0.2, que es equivalente a R-4.0.2. Cambiando la configuración automática de R para que utilice análisis ANOVA Tipo III ya que existen fuertes interacciones entre los factores y queremos estudiarlas.

Para implementar todos estos métodos del capítulo se ha utilizado Visual Studio 2020 con el lenguaje de programación C#. Se han integrado en la plataforma de desarrollo de algoritmos FACOP que se detalla en el capítulo 6. Las experimentaciones se llevaron a cabo en un clúster de ordenadores con CPUs "Intel Xeon Gold 5220" a 2.2GHz. Las máquinas virtuales con sistema operativo Windows 10 simulan un equipo con un procesador de 4 núcleos y con 16GB de memoria RAM disponible.

4.5.2.1 Resultados del Modelo

El modelo propuesto en la sección 4.1 obtiene óptimos solo en 11 instancias, siendo todas ellas de menor tamaño. Con las instancias con 7 trabajos o más ya no obtiene la solución óptima en ningún caso. Estos resultados presentados en la tabla 4.4 son consistentes con los resultados de otros problemas donde los modelos no obtienen óptimos para las instancias que no sean muy pequeñas, y justifica la utilización de heurísticas y metaheurísticas que se realiza en este trabajo. Los resultados son los obtenidos con IBM ILOG CPLEX versión 20.1 [1], como criterio de parada se utilizó tiempo de forma constante, ofreciendo 3600s de tiempo de ejecución para cada instancia.

n	*m*	Resueltos	Óptimos	GAP promedio
5	3	16	3	11,01
	4	16	4	15,45
	5	16	4	16,67
Promedio		48	11	14,38
7	3	16	0	36,41
	4	13	0	41,56
	5	12	0	58,54
Promedio		41	0	44,52
9	3	15	0	59,97
	4	7	0	72,2
	5	2	0	83,24
Promedio		24	0	65,47
Promedio Total		113	11	36,17

Tabla 4.4: Resultados para el Modelo MILP propuesto ($PFSPR_{Scheduling}$).

4.5.2.2 Resultados de las heurísticas constructivas

Para analizar el rendimiento de las diferentes heurísticas descritas en la sección 4.2 se han diseñado experimentos que utilizan las instancias de los sets descritos en el apartado 4.5.1, usando el set de entrenamiento para las primeras pruebas y el set de test para los tests finales. Al ser métodos deterministas, los valores objetivo del *makespan* no varían al realizar múltiples ejecuciones con los mismos valores de entrada, por lo que solo se realiza una ejecución por cada caso. Al ser tres sets de instancias diferentes, también se compara el rendimiento con los distintos sets.

[1] Fuente: https://www.ibm.com/es-es/products/ilog-cplex-optimization-studio

Orden	*Heur*	Promedios de RPD				Tiempo Promedio
	Heurísticas Constructivas	**Todos los sets**	**Set VRF Small**	**Set Taillard**	**Set VRF Large**	**Todos los sets (ms)**
1	*Palmer*	1,80	2,99	2,23	0,40	544,12
2	LPT_t	2,79	4,46	2,90	1,06	930,20
3	LPT_f	2,85	4,55	2,95	1,09	940,53
4	$KK1$	2,89	4,71	2,85	1,09	561,57
5	*SAR*	3,34	5,18	3,78	1,28	938,96
6	$LAAr_f$	3,69	5,73	4,17	1,42	941,10
7	$LAAp_t$	3,69	5,68	4,07	1,50	953,10
8	$LAAp_f$	3,73	5,81	4,25	1,39	942,53
9	$LAAr_t$	3,74	5,75	4,17	1,51	940,72
10	SRC_f	3,77	6,01	3,91	1,46	962,47
11	$SAAr_t$	3,84	6,15	4,10	1,41	972,38
12	$SAAp_f$	3,85	6,14	4,20	1,38	959,99
13	$SAAr_f$	3,90	6,20	4,27	1,40	953,38
14	$SAAp_t$	3,90	6,24	4,21	1,41	944,53
15	SPT_f	3,91	6,90	3,43	1,17	937,01
16	SPT_t	3,92	6,85	3,55	1,18	952,46
17	*LAR*	3,95	6,68	3,49	1,44	960,70
18	LRC_f	3,97	6,13	4,32	1,63	941,06
19	SRC_t	4,05	6,65	4,19	1,39	936,42
20	LRC_t	4,11	6,46	4,42	1,62	949,37

Tabla 4.5: Resultados RPD promedio de las heurísticas constructivas con instancias de entrenamiento.

Primero se presentan en la Tabla 4.5 los resultados promedios de la desviación porcentual relativa (Average Relative Percentage Deviation -ARPD-) obtenida por los diferentes métodos estudiados de manera independiente sobre las instancias de entrenamiento. Esta tabla contiene 7 columnas para representar los resultados. En la tercera columna están los resultados RPD promedio con todas las instancias, las siguientes tres columnas contienen los resultados desagregados por set de instancias y finalmente la última columna contiene el tiempo promedio de cálculo, de nuevo con todas las instancias juntas en milisegundos. La tabla está ordenada en base a la tercera columna, el RPD promedio de todas las instancias. En esta tabla no aparece ninguna de las opciones de *tie break* de las reglas *LAR* y *SAR*, ya que en estas reglas no aparecen empates. A partir de la tabla 4.5 se observa que la heurística de Palmer es la que obtiene mejores resultados promedio en todos los sets, siendo además muy rápida. Para comprobar que reglas son "mejores" se probaron dos medidas: la primera consiste en observar el número de ocasiones en que una heurística obtiene la mejor solución y la segunda observando los RPD promedio de los métodos. Ambas medidas ofrecen resultados muy similares pero en la primera se pudo constatar que incluso las heurísticas con peor promedio obtienen algunas de las mejores soluciones. Existen métodos que son buenos en situaciones diferentes, de tal forma que se plantea como interesante combinar varias reglas o heurísticas juntas de tal forma que, aunque se duplique el tiempo de cálculo, al ser métodos muy rápidos, el coste en tiempo de cómputo sigue siendo muy pequeño.

Se añaden las siguientes heurísticas multipasada que utilizan más de una regla para generar una solución:

- **Best three (B3):** Este método resuelve el problema con las tres mejores heurísticas (*Palmer*, *LPT* y $KK1$) quedándose con la mejor solución.

- **Best four (B4):** Se añade al algoritmo multipasada B3, la heurística *SAR*.

– **Best ten (B10):** Algoritmo multipasada con las heurísticas: *Palmer*, LPT_t, $KK1$, *SAR*, $LAAp_t$, SRC_f, $SAAr_t$, SPT_f, $LARr_f$ y LRC_f.

– **Todas (All):** Se crea un algoritmo multipasada que básicamente prueba todas las reglas con todos sus parámetros y se queda con la mejor. El tiempo de cálculo estimado será muy superior a las demás reglas, pero tendrá el mejor resultado de todas ellas.

Los resultados de estos algoritmos multipasada sobre las mismas instancias de entrenamiento se encuentran en la tabla 4.6. Todos los métodos mejoran a *Palmer*, heurística que todos los algoritmos contienen. La mejora en eficacia viene acompañada de un empeoramiento en la eficiencia, como es de esperar, ejecutar varios métodos es más costoso en tiempo.

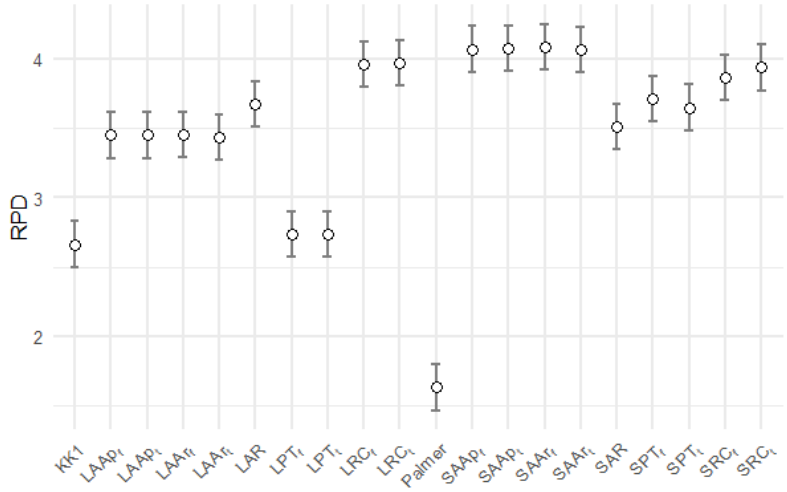
Orden	*Heur*	Promedios de RPD				Tiempo Promedio
	Heurísticas Multipasada	**Todos los sets**	**Set VRF Small**	**Set Taillard**	**Set VRF Large**	**Todos los sets (s)**
1	*All*	0	0	0	0	23,74
2	*B*10	0,22	0,36	0,21	0,07	8,64
3	*B*4	0,59	1	0,68	0,15	2,86
4	*B*3	0,91	1,63	0,93	0,18	2,02

Tabla 4.6: Resultados RPD promedio de las agrupaciones de heurísticas constructivas con instancias de entrenamiento.

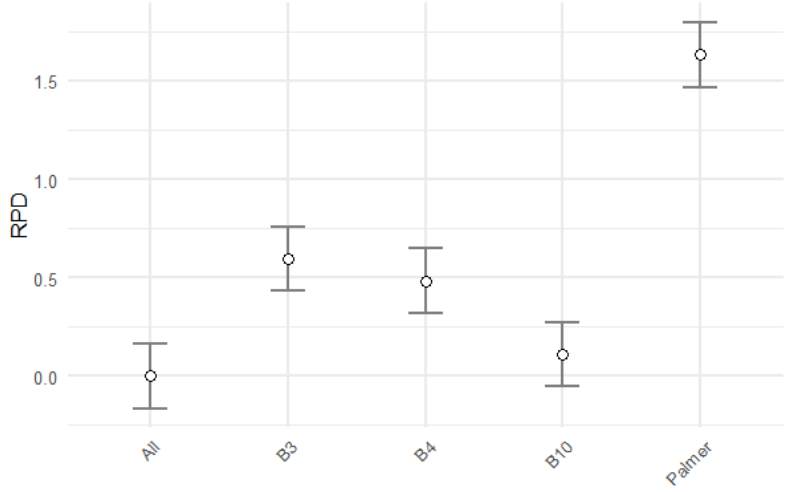
A partir de los datos anteriores podemos medir la eficacia y eficiencia de los métodos propuestos. Para observar la eficiencia necesitaremos centrarnos en el tiempo de cómputo. Para ello examinamos la última columna de la tabla 4.5 de tiempos de cálculo promedio en milisegundos, en ella encontramos el tiempo que han tardado en resolver el problema estas heurísticas. El mejor caso es *Palmer*. Podemos ver que cada caso independiente tarda entre medio y un segundo, sin embargo, las agrupaciones de heurísticas que se ven en la tabla 4.6, tardan unos 2, 3, 9 y 24 segundos aproximadamente. Al ser heurísticas muy rápidas, es asequible ejecutar varias y quedarte con la mejor, dada su eficiencia.

Sabiendo que el método *All* sin duda obtendrá los mejores resultados, la duda que persiste es si se debería utilizar este método, otro de los grupos de heurísticas o si se debería utilizar una sola heurística ya que se obtiene una mejora en resultados moderada, pero se incrementa el tiempo de cálculo. Comparando se puede ver la mejora de eficacia de un $35,1\%$ de B4 sobre B3 en el RPD promedio, mientas el incremento de tiempo de cómputo es de un $29,3\%$. Entre B10 y B4 el incremento en tiempo de cómputo es de un 202% pero la mejora en la solución es $168,2\%$. Para *All* no tiene sentido mirar el RPD promedio, pero en tiempo es $8,29$ veces mayor que B4. Tanto a la hora de comparar la calidad de la solución de las heurísticas como su velocidad de cálculo, encontramos que el algoritmo de Palmer obtiene mejores resultados promedios y requiere muy poco tiempo de cálculo, esto convierte a esta heurística en la mejor heurística independiente.

La experimentación anteriormente descrita también se ha realizado sobre el conjunto de instancias de test del apartado 4.5.1. Los resultados obtenidos son muy similares y se pueden encontrar en las tablas A.1 y A.2 de los anexos. Para ilustrar con más detalle los resultados, se incluyen las figuras 4.4a y 4.4b. En la figura 4.4a se observa la gráfica de medias de todos los métodos independientes junto con los intervalos de Tukey, en ella un punto importante es ver como los diferentes desempates influyen en los promedios de las heurísticas (excepto SAR y LAR). Sin embargo, las diferencias observables en la figura no son estadísticamente significativas ya que los intervalos de Tukey se solapan claramente. En la figura 4.4b se muestra la gráfica de medias de todos los métodos multipasada junto con Palmer, en la gráfica se incluyen los intervalos de Tukey, en ella se aprecia una diferencia estadísticamente significativa entre la heurística multipasada B10 y las heurísticas B3 y B4. Asimismo se aprecia que no hay diferencias estadísticamente significativas



(a) Heurísticas constructivas con desempates.



(b) Heurísticas multipasada con Palmer.

Figura 4.4: Resultados del RPD promedio de las heurísticas constructivas sobre las instancias de test del apartado 4.2 con intervalos de Tukey.

entre B10 y *All*, ya que se solapan en la imagen. Los intervalos de Tukey se han obtenido con un estudio Análisis de la Varianza (ANOVA) sobre los resultados de las instancias de test, análisis en el que se han comprobado los supuestos ANOVA. La mejora en eficiencia de las heurísticas multipasada, pueden justificar el sobre coste en tiempo ya que, siguen siendo métodos muy rápidos.

4.5.2.3 Resultado de las heurísticas NEH

De forma análoga a la subsección anterior se ha realizado una serie de experimentos sobre los métodos propuestos basados en la NEH de la sección 4.4 que se detallan y muestran a continuación. Recordemos que se han propuesto tres heurísticas basadas en la NEH (NEH_{cf} , NEH_a y NEH_{nr}) y que todas ellas requieren de un método que obtenga una ordenación inicial. Para discernir qué método de ordenación resulta más eficaz y eficiente se han utilizado todos las heurísticas que aparecen en la sección 4.2 para cada uno de los 3 métodos propuestos, resultando en un total de 24 ordenaciones, por 3 versiones, un total de 72 casos. Así, esta experimentación permite tanto evaluar el comportamiento de cada ordenación con cada variante de NEH, como comparar los diferentes

tipos entre sí. Estos métodos se han ejecutado sobre los sets de instancias de entrenamiento que se describen en la sección 4.5.1, y se ha utilizado el criterio de parada (*CP*) que se define en la fórmula 4.18, siendo ρ en este experimento siempre 90.

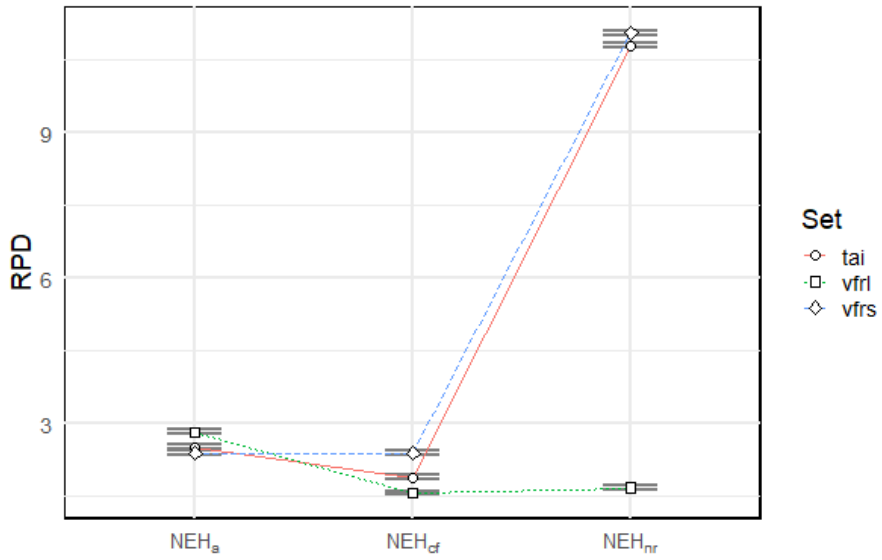
Nº	Tipo de	Todas las instancias		VRF Small		Taillard		VRF Large	
	NEH	ARPD	Tiempo (s)	ARPD	Tiempo (s)	ARPD	Tiempo (s)	ARPD	Tiempo (s)
0	NEH_{cf}	**1,97**	339,26	**2,40**	1,36	**1,89**	64,76	**1,57**	814,40
1	NEH_a	2,60	339,08	**2,40**	1,38	2,53	64,80	2,83	813,92
2	NEH_{nr}	7,26	**11,35**	11,07	**0,034**	10,8	**1,12**	1,68	**27,78**

Tabla 4.7: Resultados de la desviación porcentual relativa promedio (ARPD) para los diferentes métodos basados en la NEH.

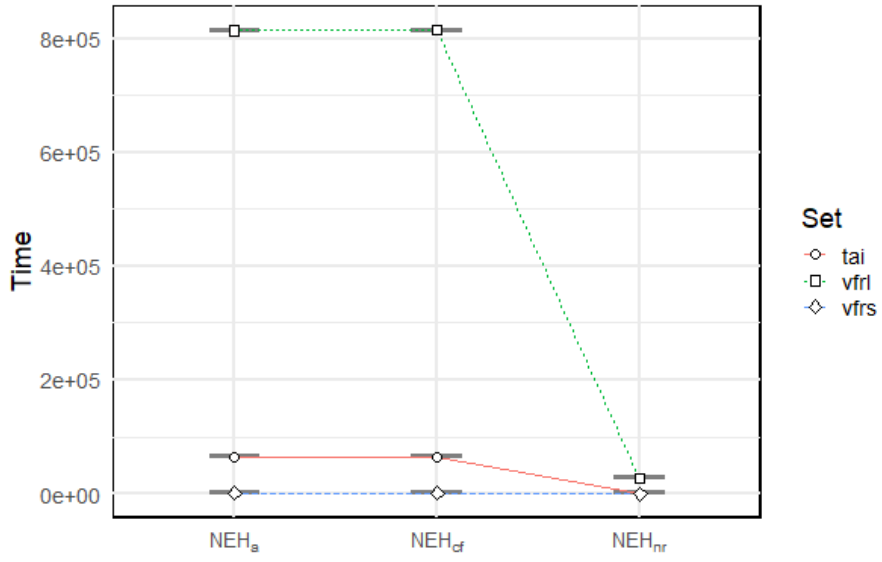
Inicialmente se analizan los resultados para evaluar el sistema de ordenación de cada heurística NEH. Estos resultados al ser tablas muy grandes se encuentran en los apéndices A por consideraciones de espacio. En la tabla A.4 se muestran los valores promedio de NEH_{nr} y en la tabla A.3 encontramos las heurísticas NEH_{cf} y NEH_a, cada una de ellas junto con sus 24 posibles ordenaciones. En cada caso, los mejores valores promedio de RPD para todas las instancias son $NEHcf_{kk1}$, $NEHa_{B10}$ y $NEHnr_{SRCt}$, aunque existen variaciones según el set de instancias. Para completar los resultados se han realizado estudios ANOVA de estos datos para ver la significación estadística de estas diferencias. La representación gráfica en las figuras A.1 permite evaluar estas diferencias fácilmente ya que se observan los datos promedio con el intervalo de confianza de Tukey. En los tres casos existen muchas ordenaciones y en muchos casos las diferencias son estadísticamente significativas. En cuanto a la comparación entre las diferentes versiones de las heurísitcas basadas en la NEH, se observan claras diferencias en eficacia y eficiencia. En la tabla 4.7 se muestran las tres versiones NEH, cada una con la heurística de ordenación que mejores resultados ha tenido en base al promedio de la desviación porcentual relativa (Average Relative Percentage Deviation - ARPD) para el conjunto de todas las instancias. En la tabla están las columnas referentes al RPD promedio y el tiempo promedio en segundos para cada set de instancias, y además el RPD promedio y el tiempo de todas las instancias juntas. En la figura 4.5a se muestran las diferencias de eficacia de los diferentes métodos. Para todas las instancias, NEH_{cf} funciona mejor que NEH_a, excepto en las instancias pequeñas, donde obtienen el mismo resultado. NEH_{cf} obtiene mejores RPD promedio que NEH_{nr}, aunque las diferencias promedio son grandes, en las instancias de mayor tamaño las diferencias son muy pequeñas. En la figura 4.5b se vuelven a mostrar las interacciones, pero en este caso sobre el tiempo utilizado, y es en este caso donde observamos como NEH_{nr} utiliza mucho menos tiempo de cómputo, siendo una heurística mucho más eficiente.

4.6 Conclusiones del capítulo

En este capítulo se define y explica el problema del taller de flujo de permutación con recursos adicionales (PFSPR). Para completar la definición formal del problema se presenta un modelo matemático junto con un pequeño set de instancias de test para evaluarlo. El modelo matemático no es efectivo en problemas realistas como el PFSPR excepto en problemas pequeños, por lo que se han propuesto heurísticas para resolverlo. En este capítulo se explican las heurísticas propuestas: se han adaptado algunas heurísticas conocidas como SPT y LPT, se han creado nuevas heurísticas como SRC, LRC, SAA, LAA, SAR y LAR, y finalmente se han adaptado las heurísticas SIH (Palmer) o KK1. En aquellos casos en que es posible, se añaden políticas de desempate a estos métodos. Se han propuesto algoritmos multipasada donde se agrupan reglas para obtener mejores soluciones sin llegar a empobrecer mucho la eficiencia. En el capítulo se incluye una explicación sobre la representación de la solución y su sucesiva evaluación de los problemas de este ámbito. Se realiza una extensa experimentación para poder evaluar el rendimiento de las heurísticas planteadas,



(a) Interacciones respecto al promedio RPD.



(b) Interacciones respecto al tiempo promedio utilizado.

Figura 4.5: Resultados de interacciones entre el tipo de NEH utilizado y el set de instancias con intervalos HSD de Tukey

creando para ello tres benchmarks diferentes partiendo de los ya existentes en la literatura de PFSP, procedentes de Taillard (1993) y Vallada, Ruiz y J. M. Framinan (2015). A través del análisis de los resultados en esta experimentación se puede discernir las diferencias en eficacia y eficiencia de estos métodos. En el proceso de estudio de estas heurísticas se proponen algunas multipasada para obtener un nuevo método que obtenga mejores soluciones aunque necesite más tiempo. A continuación, se proponen tres formas diferentes de adaptar la heurística NEH al problema del taller de flujo con permutación y recursos adicionales. Estas tres formas se explican detalladamente y de nuevo se realiza una extensa experimentación para poder evaluar el rendimiento entre ellas. Al mismo tiempo, y utilizando las heurísticas anteriores como sistema de ordenación para las versiones de la NEH, se evalúa el rendimiento de cada una de ellas con cada una de las ordenaciones.

Del análisis de la experimentación se concluye que las heurísticas más sencillas son muy eficientes pero poco eficaces, siendo la mejor regla independiente Palmer. Se observa que los desempates tienen impacto en la solución pero no son estadísticamente significativos. Se ha comparado el rendimiento de los diferentes algoritmos multipasada, constatando que B10 ofrece el mejor resultado, es decir, que añadiendo todas las heurísticas pero solo con su mejor regla de desempate se puede mejorar a agrupaciones con unas pocas de las mejores heurísticas. En la experimentación de los métodos basados en la NEH se comprueba que el método conocido como *cut and fill* es claramente mejor que los demás en eficacia, sin embargo, en los problemas más grandes no consigue terminar, dando como resultado una solución mejor que los demás métodos, pero aún muy mejorable. Podemos descartar la NEH_a por ser la que peor rendimiento muestra, pero por otra parte el método NEH_{nr} es muy eficiente lo que lo puede convertir en buen candidato para iniciar la solución o soluciones de métodos metaheurísticos.

Las limitaciones que muestran las heurísticas lleva a utilizar metaheurísticas para enfrentar el problema. En el capítulo 5 se definen y explican algunas metaheurísticas para el problema PFSPR, utilizando las heurísticas que mejor rendimiento ofrecen en este capítulo como soluciones de partida.

5. Metaheurísticas

A través de un análisis detallado, se ha comprobado en el capítulo precedente que las reglas, las heurísticas y el modelo introducido, aunque permiten obtener soluciones al problema del taller de flujo de permutación con recursos adicionales (PFSPR), presentan ciertas limitaciones en términos de eficiencia y eficacia. Se detecta la necesidad de utilizar técnicas más avanzadas para mejorar la calidad de las soluciones. En consecuencia, el objetivo del presente capítulo es contribuir al avance del conocimiento en el ámbito de la resolución del problema PFSPR, mediante el estudio de posibles algoritmos metaheurísticos. Estos se presentan como una alternativa complementaria a los modelos y heurísticas, pudiendo mejorar la eficacia de las heurísticas y permitiendo resolver problemas que no son manejables por los modelos. En la literatura académica, se pueden hallar numerosos trabajos que respaldan las bondades de este tipo de soluciones para problemas difíciles de *scheduling* (Ribas, Leisten y Framiñan 2010; Rossit, Tohmé y Frutos 2018; Ruiz y Vázquez-Rodríguez 2010; Talbi 2009).

El término metaheurística fue acuñado en Glover (1986). "Una metaheurística es una estructura algorítmica de alto nivel e independiente del problema que provee de un conjunto de pautas o estrategias para desarrollar algoritmos de optimización heurística. El término metaheurística también se refiere a una implementación concreta en un problema específico de un algoritmo de optimización siguiendo las pautas de dicha estructura" (Sörensen y Glover 2013). Como puntos fuertes son métodos robustos y flexibles, además, en muchos casos se utilizan parámetros que permiten a las metaheurísticas adaptarse mejor a los problemas después de ser calibrados (J. M. Framinan, Leisten y Ruiz García 2014). Las metaheurísticas son métodos aproximados de mejora, ya que no pueden garantizar la optimalidad de la solución, ni detectar cuando han llegado a un óptimo. Las metaheurísticas utilizan métodos sencillos y rápidos como las heurísticas para generar una o varias soluciones factibles de partida, e intentan mejorar estas soluciones, previamente creadas, aplicando técnicas de mejora. Este funcionamiento implica que, al aumentar el tiempo de cálculo de una metaheurística para un problema, suele mejorar la calidad de la solución obtenida, pero incluso con tiempo muy limitado se suelen encontrar soluciones buenas.

Se observa que los Algoritmos Genéticos (GA), los Algoritmos de Recocido Simulado (SA) o las Búsquedas Tabú (TS), son de los algoritmos más estudiados en los primeros años analizados, y aún hoy se utilizan, aunque de la revisión se desprende que los algoritmos genéticos se utilizan más, en su versión híbrida (HGA). También se desprende otros métodos más actuales como el Algoritmo Iterativo Voraz y el Algoritmo Iterativo de Búsqueda Local han sido también muy utilizados y cuando han sido propuestos han dado mejores resultados que los algoritmos Genéticos contra los que se probaron. A estas ideas hay que añadir la reflexión sobre "la metáfora expuesta", una serie

de críticas a la investigación con metaheurísticas en 3.4.1 (Aranha et al. 2022; Sörensen 2015). Esta reflexión ha ayudado a restringir la decisión de que metaheurísticas plantear para este problema. Estando entre los algoritmos considerados consistentes que están basados en metáfora encontramos el Algoritmo Genético Híbrido y además teniendo un problema similar estudiado en Laribi, Yalaoui y Sari (2019), el HGA se ha elegido para resolver el problema PFSPR pero ahora con recursos renovables. Además se van a considerar un Algoritmo Iterativo Voraz y un Algoritmo Iterativo de Búsqueda Local, por las razones expuestas anteriormente.

Estos algoritmos se explican de forma detallada a lo largo del capítulo, posteriormente se detallará el proceso de calibración completado con el análisis de la misma y por último se expondrán y analizarán los resultados de comparación entre las tres metaheurísticas propuestas. Como se ha mencionado anteriormente, una metaheurística está formada por métodos. En primer lugar se explicará uno de estos métodos que aparece de forma recurrente en multitud de algoritmos utilizados en *scheduling* como es la búsqueda local.

5.1 Solución inicial

Como se ha mencionado anteriormente, las metaheurísticas son métodos aproximados de mejora, y por lo tanto, parten de una solución inicial que normalmente se genera con un método heurístico. También existen algoritmos poblacionales que requieren la generación de una población de soluciones.

Para este método se puede aplicar el conocimiento obtenido en el capítulo 4 para seleccionar los mejores algoritmos heurísticos utilizados. A lo largo de este capítulo, se presentan dos algoritmos de trayectoria que necesitarán utilizar una heurística para generar la solución inicial. Se han seleccionado la heurística $B10$, la heurística NEH_{nr} con la ordenación SRC_t y la NEH_{cf} con la ordenación $KK1$.

En las secciones posteriores referentes a cada algoritmo, se ofrecen más detalles sobre la solución inicial para cada algoritmo. Además, al ser elegible se usará como un factor, aparecerá en la sección de calibración y se podrá saber qué métodos son mejores para ese papel.

5.2 La búsqueda local (LS)

Una búsqueda local es un método o procedimiento de mejora que partiendo de una solución completa, la modifica utilizando el concepto de vecindario para intentar mejorarla. Por lo tanto es un elemento de intensificación, busca la solución buena más cercana pudiendo alejarse de la solución óptima. Dos soluciones son vecinas si una de ellas se puede obtener a partir de una operación de modificación bien definida desde la primera (Pinedo 2016), así el vecindario de una solución engloba generalmente todas las soluciones que podrían obtenerse al aplicar una sola operación de modificación. La búsqueda local no incluye la solución inicial, esta se obtiene desde un método ajeno. La búsqueda local suele ser determinista, es decir, dada una solución π, al aplicarle una búsqueda local se obtendrá una solución π' y si volvemos a aplicar la misma búsqueda local a π volverá a salir π' como resultado, esto es así en la mayoría de búsquedas locales.

En las búsquedas locales, existen numerosos vecindarios, pero los siguientes tres vecindarios son los más utilizados en los problemas de *scheduling*: inserción, intercambio e intercambio adyacente:

- El vecindario de inserción viene definido por todas la posibles soluciones que se obtendrían a retirar un trabajo j y colocarlo en todas las posiciones posibles, excepto en la que estaba anteriormente.
- El vecindario de intercambio se define por las posibles soluciones que surgen al realizar una operación de intercambio, en la operación de intercambio se parte de una solución y dos trabajos j y k, estos trabajos cambian su posición, quedando j donde antes estaba k y k donde antes estaba j.

- El vecindario de intercambio adyacente viene definido por la misma operación de intercambio pero ahora solamente de dos trabajos consecutivos en la permutación j y k. Estos se cambian de orden pasando a estar ahora k seguido de j.

A continuación, se pueden observar en los ejemplos de la figura 5.1, tres ejemplos de las operaciones de vecindarios partiendo desde una permutación inicial que es (3,6,1,2,7). En el ejemplo de inserción j es el trabajo que está en la segunda posición (6) y se desplaza a la posición 4 seleccionada, obligando a los trabajos de las posiciones 3 (1) y 4 (2) a adelantarse una posición. En el ejemplo de intercambio j=2 y k=4 por lo que el trabajo 6 toma la posición 4 y el trabajo 2 toma la posición 2. Por último, en el intercambio adyacente en las posiciones 2 y 3 se intercambian los trabajos pasando de ser 6,1 a ser 1,6.

$$
\begin{array}{rlll}
\text{Permutación original:} & (3,6,1,2,7) & & j=2 \\
\text{Inserción:} & (3,\mathbf{6},1,2^{\downarrow},7) & \rightarrow (3,1,2,\mathbf{6},7) & j=2,k=4 \\
\text{Intercambio:} & (3,\mathbf{6},1,\mathbf{2},7) & \rightarrow (3,\mathbf{2},1,\mathbf{6},7) & j=2,k=4 \\
\text{Intercambio adyacente:} & (3,\mathbf{6},\mathbf{1},2,7) & \rightarrow (3,\mathbf{1},\mathbf{6},2,7) & j=2,k=3
\end{array}
$$

Figura 5.1: Ejemplos de los diferentes vecindarios de la búsqueda local.

```
Procedure Local Search (ExternalSolution)
begin
    π = ExternalSolution
    while termination criterion not satisfied AND local optima not found do
        // Búsqueda local en vecindario
        for 1 to n do
            π' = NeighbourhoodOperation
        end
        if C_max(π') < C_max(π) then
            π = π'
        else
            // Óptimo local encontrado
        end
    end
    return π
end
```

Algoritmo 5.2.1: Pseudocódigo del procedimiento de Búsqueda Local (LS), utilizando el diseño de *mejor solución*.

Otro concepto importante en el diseño de las búsquedas locales es la estrategia de selección. Hay principalmente dos opciones: *mejor solución* y *primera mejora*. En el algoritmo 5.2.1 se aprecia la búsqueda local con la estrategia conocida como *mejor solución* ya que recorre todo el vecindario (prueba todas las posibles operaciones) y se queda con la mejor solución de las estudiadas. La conocida como *primera mejora* o primera mejora, va realizando movimientos (o evaluando vecinos) hasta que encuentra una solución que mejora la solución inicial y una vez encontrada se para, aunque haya más posibles vecinos por evaluar. Como último concepto, se encuentra el criterio de terminación de la búsqueda local, esto es, la condición que determina cuándo se ha terminado el procedimiento. La búsqueda local acaba cuando ya no existe mejora

posible en el vecindario de la solución, estas soluciones se conocen como óptimo local porque utilizando el operador de vecindario no se puede encontrar ninguna solución mejor.

Las búsquedas locales son procedimientos muy útiles pero con capacidades limitadas, ya que solo permiten mejorar a una solución hasta cierto punto y allí se quedan atrapadas en un óptimo local. Al ser estrategias de intensificación necesitan incorporar elementos que ayuden a la diversificación para escapar de ese óptimo local Lourenço, Martin y Stützle (2019).

5.3 El Algoritmo de Búsqueda Local Iterada (ILS)

El algoritmo de búsqueda local iterada (ILS) se ha propuesto para multitud de problemas de optimización, como por ejemplo el viajante de comercio (TSP) en Lourenço, Martin y Stützle (2019), problemas de asignación cuadrática (QAP) en Stützle (2006) y, también, se ha utilizado en multitud de problemas de *scheduling* en Q.-K. Pan, Ruiz y Alfaro-Fernández (2017) y Stützle (1998). A este tipo de algoritmos se les denomina algoritmos de trayectoria, debido a que solamente trabajan con una solución y no con una población de soluciones. Así el proceso iterativo de mejora se puede representar como una trayectoria o traza a través del espacio de soluciones.

La búsqueda local iterada requiere de una solución inicial y después intenta mejorarla utilizando un conjunto de componentes de forma iterativa hasta que se termina la ejecución en base a un criterio de parada. Los componentes principales son: el método que construye la solución inicial, la búsqueda local, la perturbación y el criterio de aceptación. En el algoritmo 5.3.1 se puede ver la estructura general de un método ILS.

```
Procedure IteratedLocalSearch
begin
    π = NEH
    π_b = Local Search(π)
    while termination criterion not satisfied do
        // Perturbación
        π' = Perturbation(strength, history, π)
        // Búsqueda Local
        π'' = Local Search(π')
        // Criterio de aceptación de temperatura constante
        if C_max(π'') < C_max(π) then
            π = π''
            if C_max(π) < C_max(π_b) then
                π_b = π
            end
        else
            if random ≤ exp{−(C_max(π'') − C_max(π))/T_c} then
                π = π''
            end
        end
    end
    return π_b
end
```

Algoritmo 5.3.1: Pseudocódigo del algoritmo Iterated Local Search (ILS)

El método que se utiliza para obtener **la solución inicial** es un componente totalmente independiente de lo que sería el ILS. En muchas ocasiones se han utilizado constructores aleatorios, reglas o heurísticas constructivas como las explicadas en el capítulo 4. Sin embargo, hay muchos trabajos que sustentan que para obtener un buen funcionamiento del ILS es menester que el método inicial

ofrezca una buena solución (Stützle 1998; Taillard 1990). Como el algoritmo 5.3.1 es el basado en Stützle (1998) se utiliza la heurística constructiva NEH clásica para obtener la solución inicial.

El componente de **la búsqueda local** del que obtiene el nombre, ya ha sido definida en la sección anterior. La búsqueda local aquí es la encargada de intensificar la búsqueda e intentar obtener mejores soluciones respecto a la solución inicial buscando en el vecindario correspondiente.

Otro de los mecanismos característicos de el ILS es la **perturbación**, que se desarrolla para evitar que la solución quede atrapada en un óptimo local. Una perturbación consiste en alterar la solución lo suficiente para que tras varias búsquedas locales el algoritmo no vuelva a atascarse en el mismo óptimo local. Al mismo tiempo la solución perturbada debe presentar la suficiente similitud con la solución no alterada como para mantener algunas de las características que la hacían buena en un principio. Podemos decir que la perturbación es la encargada de diversificar la búsqueda y asegurarse que se explora el espacio de soluciones posibles. La perturbación suele ser la combinación de dos parámetros, por una parte la operación que se realiza (el vecindario), y por otra, el número de operaciones que se realizan, parámetro conocido como fuerza o *strength* (Lourenço, Martin y Stützle 2019; Stützle 1998).

En el trabajo original de Stützle (1998) se utiliza un **criterio de aceptación** conocido como *"de temperatura constante"* que está basado en los principios de otra metaheurística conocida como *Simulated Annealing* (Osman y Potts 1989) y permite continuar la búsqueda desde una solución una solución ligeramente peor. Este criterio de aceptación concreto se le conoce como de temperatura constante ($ConstTemp(s,s'')$) y se regula mediante un parámetro de entrada llamado temperatura que tras un ajuste basado en las características de la instancia se utiliza para decidir si aceptar estas peores soluciones (Osman y Potts 1989; Ruiz y Stützle 2007; Stützle 1998). Algunas variantes de ILS además almacenan información sobre el punto de la última mejora en la búsqueda local para retomarla en un punto determinado y evitar así retroceder hacia soluciones recientemente evaluadas. Si acaba de estudiar el vecindario de inserción en un elemento y realiza un movimiento, la próxima iteración comenzará en el siguiente elemento de la permutación.

El **criterio de parada** del algoritmo no sería en si un componente, es más bien la forma en la que se decide cuando termina la ejecución del mismo. En nuestro caso será un criterio basado en el tiempo de ejecución y variable según el tamaño del problema a tratar (*CP*), como ya se ha explicado en la sección 4.5.2 y en la expresión 4.18.

5.4 El Algoritmo Voraz Iterado (IG)

El algoritmo voraz iterado, conocido en inglés como *Iterated Greedy* (IG), se propone por primera vez en Ruiz y Stützle (2007) y ha sido aplicado a una gran variedad de problemas de *scheduling* obteniendo muy buenos resultados. En el artículo original, los autores estudian el problema del taller de flujo de permutación. Más adelante, en Ruiz y Stützle (2008), los mismos autores aplican el algoritmo al problema de taller de flujo de permutación con tiempos de cambio dependientes de la secuencia y dos objetivos. También ha sido aplicado en el estudio del problema de máquinas paralelas (UPMR Fanjul-Peyro y Ruiz 2010) y otros autores también han propuesto soluciones a diferentes variaciones de problemas de taller de flujo, como el problema del taller de flujo con bloqueo que encontramos en Ribas, Companys y Tort-Martorell (2011), el problema del taller de flujo de permutación con tiempos de cambio dependientes de la secuencia en Minella, Ruiz y Ciavotta (2011), el taller de flujo multi-objetivo de Ciavotta, Minella y Ruiz (2013), el problema del taller de flujo de permutación distribuido Ruiz, Q.-K. Pan y Naderi (2019) o el problema del taller de flujo con bloqueos, tiempos de cambio y mantenimientos que se explora en Miyata y Nagano (2022).

El algoritmo IG, al igual que el ILS, es una metaheurística de trayectoria. Los componentes de esta metaheurística son: el método constructivo, la búsqueda local, el proceso de destrucción, el proceso de reconstrucción y el criterio de aceptación. Como en todos los algoritmos de trayectoria,

se comienza con una heurística constructiva a la que en este caso se le aplica una búsqueda local. A partir de esa solución generada se entra en el bucle del método, un proceso iterativo de mejora que incluye un proceso de destrucción-reconstrucción y una búsqueda local. Este funcionamiento se continúa hasta que se cumple el criterio de parada y se finaliza la ejecución.

El primer componente del algoritmo es el método que obtiene **la solución inicial** y suele ser una heurística constructiva independiente del IG. En el pseudocódigo que se ofrece en el algoritmo 5.4.1, se vuelve a plantear como ejemplo de constructor la heurística NEH.

```
Procedure IteratedGreedy (d, T)
begin
    π = NEH
    π_b = π
    // Fase de búsqueda local
    π = Iterative Improvement Local Search(π)
    while termination criterion not satisfied do
        // Fase de destrucción-construcción
        // Procedimiento de destrucción
        for 1 to d do
            π' = destroy a random element from π
        end
        // Procedimiento de construcción
        for 1 to d do
            π' = re-insert one removed element into all positions and keep the best one
        end
        // Fase de búsqueda local
        π'' = Iterative Improvement Local Search(π')
        // Criterio de aceptación de temperatura constante
        if C_max(π'') < C_max(π) then
            π = π''
            if C_max(π) < C_max(π_b) then
                π_b = π
            end
        else
            if random ≤ exp{−(C_max(π'') − C_max(π))/Temperature} then
                π = π''
            end
        end
    end
    return π_b
end
```

Algoritmo 5.4.1: Pseudocódigo del algoritmo Iterated Greedy (IG)

La **fase de destrucción** está vinculada a la fase de reconstrucción y por ello en ocasiones se plantean juntas como fase de destrucción-construcción. En la fase de destrucción se altera la solución destruyendo una serie de elementos de la misma, es decir, se eliminan trabajos de la solución. El número de trabajos destruidos viene determinado por el parámetro d, cuanto más grande sea este parámetro mayor será la alteración en la solución. Teniendo ahora esta solución parcial de *n-d* elementos y d elementos extraídos de la solución, se procede a **la fase de reconstrucción**. Como norma general en problemas de *scheduling*, esta se hace siguiendo las mismas pautas que haría una heurística NEH, probando a insertar esos trabajos uno por uno en todas las posiciones posibles generando así soluciones y eligiendo entre ellas la mejor encontrada,

antes de pasar al siguiente trabajo. De esta forma, la fase de destrucción-construcción cumple el cometido de diversificación en la exploración del campo de soluciones, pero a diferencia de una Perturbación-ILS, en estas fases se intenta obtener una mejor solución al insertar los trabajos de forma "enfocada" a buscar mejoras.

La solución ya *reconstruida*, pasa ahora por otra fase, **una búsqueda local** (LS), como la descrita en la sección 5.2.1. La búsqueda local que se propone en el algoritmo IG de esta tesis es la descrita en la sección 5.2.1, pero esta viene prefijada y pero no utiliza los parámetros descritos, estos vienen ya que vienen prefijados, se utiliza siempre el vecindario de inserción con la estrategia de *"primera mejora"*. El papel de la búsqueda local es de nuevo el de intensificación, intentando encontrar la mejor solución cercana en el espacio de soluciones.

Por último, esta solución pasa por el **criterio de aceptación** que en este caso es igual al del algoritmo ILS, también es un criterio de aceptación de temperatura constante que se explica en la sección 5.3 y proviene del algoritmo SA (Osman y Potts 1989). El **criterio de parada** (*CP*) es el que aparece en la expresión 4.18 y es el mismo en todos los algoritmos.

5.5 Algoritmo Genético Híbrido (HGA)

Los algoritmos genéticos (GAs) son metaheurísticas muy populares desde que fueron desarrollados por Holland a principios de los años 70 (Talbi 2009). En Holland (1975) se establecen los principios básicos. Los algoritmos genéticos son un tipo de metaheurística evolutiva con poblaciones, no trabaja con una sola solución sino con un conjunto de soluciones conocido como población. Por ello a las soluciones se les suele llamar individuos. Los algoritmos genéticos imitan la evolución genética de un grupo o población de seres vivos, está basada en la biología y la teoría de la evolución. La población de individuos se somete a procesos evolutivos como son la reproducción, la mutación y la selección (imitando la evolución de las especies en la naturaleza). A lo largo de las generaciones sobrevivirán los individuos mejor adaptados y los peor adaptados perecerán, es decir, serán descartados de forma *"natural"*. De esta forma un individuo mejor adaptado que otro será el que tenga un valor de la función objetivo mejor. Mejor dependerá de si el objetivo es de minimización o maximización. Muchas veces se utiliza el termino *fitness* para referirse a este valor objetivo en el contexto de los algoritmos evolutivos. Los algoritmos genéticos utilizan una representación de la solución que codifica una solución, ha de ser compatible con estos algoritmos y se le denomina cromosoma. Como ya se explicó en la sección 4.3, la representación de la solución en los problemas del taller de flujo suele ser una permutación de trabajos y aunque tradicionalmente, los GAs se asociaban a representaciones de la solución binarias, existen muchos tipos de representaciones y la representación de permutación encaja como cromosoma de un algoritmo genético, por ello, se ha utilizado en múltiples ocasiones como se apreciaba en la tabla 3.3.

Algunas variantes de los algoritmos genéticos incluyen búsquedas locales para mejorar los individuos, esta mejora sobre el clásico GA se puede denominar algoritmos meméticos o algoritmos genéticos híbridos. En la revisión bibliográfica se aprecia que es una práctica común. De ahora en adelante nos referiremos a este algoritmo como *HGA* en base al nombre en inglés *Hybrid Genetic Algorithm*. Téngase en cuenta que este *HGA* tiene un parámetro adicional que aplica o no búsquedas locales al algoritmo, así que técnicamente según los parámetros que se usen será un HGA o un GA.

Para abordar este problema, se replica el algoritmo genético de Laribi, Yalaoui y Sari (2019), donde se propone un algoritmo genético híbrido para el problema del taller de flujo de permutación con recursos adicionales no renovables. Los recursos no renovables utilizados en este problema serían, por ejemplo, materias primas. Por lo tanto, el comportamiento de los recursos en su problema es diferente al de nuestro problema; sin embargo, dada la similitud entre los problemas, el algoritmo no sufre cambios significativos.

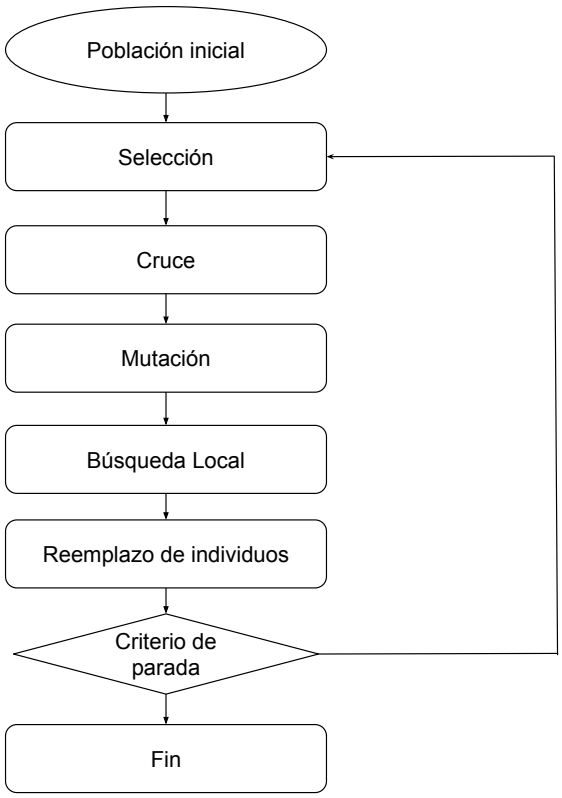


Figura 5.2: Diagrama de un algoritmo genético híbrido.

Como se aprecia en la figura 5.2, un algoritmo genético híbrido normalmente se compone de una serie de componentes o métodos: un método de generación de la población inicial desde cero, un método de selección, un método de cruce o reproducción, un método de mutación, una búsqueda local y finalmente una estrategia para el remplazo de individuos de la población. Otra representación más detallada es la que se muestra en el algoritmo 5.5.1.

5.5.1 Inicialización de la población

A diferencia de los algoritmos de trayectoria explicados anteriormente, los algoritmos genéticos son algoritmos poblacionales. Esto quiere decir que no trabajan con una sola solución, trabajan con un grupo de soluciones. La población en un algoritmo genético es por lo tanto el conjunto de individuos o soluciones con las que se realiza la búsqueda en el espacio de soluciones del problema. Esta población va cambiando, evolucionando, mediante operadores genéticos a lo largo del tiempo (de forma iterativa). A la población *superviviente* de este proceso en la iteración número X, se la conoce como generación X.

La población inicial tiene dos parámetros:

- El primero es el tamaño de la población (pop o P_{size}), que es un número entero que representa con cuantos individuos va a trabajar el HGA.
- El segundo es la forma en la que se genera la población inicial. La población puede ser generada de manera completamente aleatoria. Y también se pueden generar algunos buenos individuos usando heurísticas y se completa la población restante con individuos aleatorios.

Para el HGA de Laribi, Yalaoui y Sari (2019) cuando se genera la población inicial no es completamente aleatoria, se utilizan tres constructivos heurísticos: el algoritmo de Palmer, la

heurística NEH y la heurística NEH mejorada (NEH_{kk1}); y se completa el resto de la población con individuos aleatorios. El tamaño y la forma de generar la población inicial puede afectar mucho a la velocidad en la que este algoritmo converge hacia buenas soluciones y a la diversidad de la población en el algoritmo. Por ello elegir los parámetros correctos es crucial.

```
Procedure HybridGeneticAlgorithm
begin
    // Población Inicial
    POP = InicialPopulation(P_size)
    while termination criterion not satisfied do
        // Método de selección
        POP' =WheelSelection(POP)
        // Método de cruce
        POP'' =CrossOver(POP', P_c)
        // Mutación de la descendencia
        POP''' =Mutation(POP'', P_m)
        // Búsqueda Local (en este ejemplo, la que se aplica solo al
           mejor elemento)
        π_ls = LocalSearch(π_best)
        // Incluir todos los individuos en la población
        POP = IncludeNewIndividuals(POP'', POP''', π_ls)
        // Selección de la siguiente generación
        POP = DiscardWorstIndividuals(P_size)
    end
    return π_best
end
```

Algoritmo 5.5.1: Pseudocódigo del algoritmo Hybrid Genetic Algorithm (HGA) de Laribi, Yalaoui y Sari (2019)

5.5.2 Selección

La selección de candidatos para la reproducción en un algoritmo genético nos permite reflejar que los mejores individuos tengan mayor probabilidad de reproducirse. Así se transmitirá la información genética de los candidatos mejor adaptados a las generaciones posteriores y se perderá la de los peor adaptados. Simultáneamente existe el problema de la endogamia, esto es, si siempre son los mismos individuos los que se reproducen, la descendencia acaba estando llena de los mismos cromosomas estancando la búsqueda y generando siempre individuos muy similares. Por lo que se ha de buscar un equilibrio.

Existen otros operadores de selección pero en el *HGA* que presentamos en esta tesis, así como en el *HGA* de Laribi, Yalaoui y Sari (2019) se utiliza el operador conocido como *selección por ruleta*, que incrementa la probabilidad de selección de cada individuo en base a lo bueno que es su fitness, en este caso el *Makespan*. En el algoritmo 5.5.1 se representa por "WheelSelection(POP)" y el método utiliza la población actual para realizar la selección. La idea principal de este procedimiento es que todos los individuos tengan la oportunidad de reproducirse de forma proporcional a su nivel de adaptación o adecuación (Laribi, Yalaoui y Sari 2019). El operador de selección por ruleta usa el valor objetivo o fitness de un individuo (f_i), aplicando un principio de proporcionalidad y calculando así la probabilidad de selección del individuo i (p_i). Como es un problema de minimización del valor objetivo, el valor fitness del individuo para este operador se calcula como $F_i = 1/f_i$, de esta forma los individuos con menor valor objetivo tendrán mejores probabilidades de ser elegidos para

la reproducción. La fórmula para calcular la probabilidad de selección es: $p_i = F_i/(\sum_1^n F_i)$.

5.5.3 Operador de cruce

Este proceso en ocasiones es llamado reproducción o recombinación, pero se le suele referir por el nombre en inglés *crossover*. En este proceso se generan nuevos individuos, a los que conocemos como hijos o descendencia (*offspring*), para ello se combina la información genética de otros dos individuos, a los que llamamos padres. Aunque los individuos ya han sido seleccionados, ahora se aplica otro parámetro que puede hacer que estos no se reproduzcan, este parámetro es la probabilidad de cruce (P_c), con ella se decide si el cruce se dará entre ellos o no. Existen varios tipos, en el *HGA* propuesto en esta tesis se prueban los dos operadores de cruce más utilizados en *scheduling*:

- Cruce de un punto (OPX - *One-point crossover*): Se selecciona un punto de corte en la representación de la solución de los padres, y se generan dos nuevos individuos combinándolas en los hijos. Así, siendo los padres A y B, se dividirán en trozos de genoma: $A_{izquierda}$, $A_{derecha}$, $B_{izquierda}$ y $B_{derecha}$; con estas cuatro partes de cromosomas se generarán nuevos individuos C y D, donde C se generará a partir de combinar $A_{izquierda}$ y $B_{derecha}$ y D se generará combinando $A_{derecha}$ y $B_{izquierda}$.
- Cruce de dos puntos (TPX - *Two-point crossover*): Se seleccionan dos puntos de corte en la representación de la solución de los padres. De esta forma ahora cada padre se divide en 3 secciones. Para el primer hijo se utiliza la sección central del primer padre y los extremos del segundo, mientras que para el segundo hijo se hará con los extremos del primero y el centro del segundo padre.

Si se aplica directamente el operador de cruce, se obtendrían soluciones no factibles, porque en una permutación los trabajos no se pueden repetir. Si se heredara el genoma de forma fija, los hijos tendrían trabajos repetidos y faltaría algún trabajo. Por ello una técnica habitual es utilizar los puntos de cruce por orden. Esto significa que un hijo hereda el genoma de uno de los padres de forma literal, con los mismos elementos, en el mismo orden y en la misma posición. Sin embargo, del segundo progenitor no se hereda su parte de forma literal, se heredan los trabajos que faltan, en las posiciones que faltan por llenar y su posición exacta vendrá determinada por el orden en el que aparecen en el segundo padre. Para poder comprobar cual de los dos operadores de cruce es mejor se probarán ambos en la calibración.

5.5.4 Mutación

Después de aplicar la operación de cruce cabe la posibilidad de que los descendientes pueden sufrir una mutación de acuerdo a una probabilidad P_m.

Esta mutación ayuda al algoritmo a escapar de óptimos locales fuertes en el proceso de diversificación (Laribi, Yalaoui y Sari 2019).

Otro parámetro a considerar es el tipo de mutación a aplicar, basada en los vecindarios de inserción y de intercambio explicados en la sección 5.2. Se selecciona un trabajo aleatoriamente de cada descendiente y se aplica la inserción en una posición también seleccionada al azar. En el caso del vecindario de intercambio, se seleccionan dos trabajos aleatoriamente de cada descendiente y se intercambian.

5.5.5 Búsqueda local

En el algoritmo genético *HGA* que se diseña en Laribi, Yalaoui y Sari (2019) la búsqueda local aparece asociada a una serie de parámetros, ya que se puede aplicar de diferentes formas. Se utiliza una búsqueda local basada en la heurística NEH pero se aplica de tres formas diferentes:

- Posible búsqueda local sobre el mejor individuo: se aplica la búsqueda local exclusivamente sobre el mejor elemento de la población actual, pero se hace en base a una probabilidad (P_{ls}).

Así, en cada generación se "lanza un dado" y según el resultado se realiza la búsqueda local o no se realiza.
- Porcentaje de población: se aplica la búsqueda local de forma prefijada a un porcentaje de toda la población en todas las generaciones.
- Sin búsqueda local: no se aplica ninguna búsqueda local, el algoritmo en este caso funcionaría como una algoritmo genético que no fuese híbrido.

5.5.6 Reemplazo

Una vez generados los descendientes, se debe decidir si se aceptan o no en la población, ya que el tamaño de la población se ha de mantener constante en cada iteración. A lo largo de una iteración del algoritmo, los nuevos individuos nuevos se van sumando a la generación actual: inicialmente se tiene los de la generación anterior, a los cuales se añaden los descendientes generados en la reproducción, también se han incluido los elementos que han sufrido mutaciones y por último, los que se han obtenido de la búsqueda local. Esta población con todos los elementos juntos tiene muchos más individuos que lo determinado por el tamaño de la población P_{size}, así que el algoritmo ordena las soluciones en base a su adecuación (C_{max}) y retiene las mejores soluciones hasta llenar P_{size}. El resto de individuos serán descartados.

5.6 Análisis computacional

Existen varios puntos referentes a la experimentación y comparación de metaheurísticas que son similares o idénticos a los utilizados para la comparación de heurísticas o de modelos realizada en el capítulo 4. Por ello, en esta sección se explican de forma resumida algunas de estas partes similares y más adelante se exploran de forma más detallada las diferencias.

Como métrica estandarizada para comparar los resultados se vuelve a utilizar la medida de la desviación porcentual relativa (RPD en la fórmula 5.1) que ya se explicó en el capítulo anterior en la sección 4.5. Recordemos que RPD es una medida de rendimiento de los algoritmos que depende de la instancia y del resultado obtenido en la mejor ejecución de todos los métodos de resolución comparados.

$$RPD_{s,i} = \frac{VFO_{s,i} - VFO_{mejor,i}}{VFO_{mejor,i}} \cdot 100 \tag{5.1}$$

La validez y eficiencia de las metaheurísticas se validan con los conjuntos de instancias del problema PFSPR presentados en el capítulo anterior, en la sección *Benchmark* 4.5.1. En dicha sección se explican dos sets de instancias muy conocidos del problema PFSP y como estos se han adaptado al problema PFSP. Los sets son las instancias de Taillard (1993) y las instancias difíciles de Vallada, Ruiz y J. M. Framinan (2015), el cual está a su vez dividido en dos sets según el tamaño de las instancias. Sobre los sets generados se crearon dos agrupaciones de instancias, unas instancias de entrenamiento para calibrar y las instancias de test para los experimentos finales.

A continuación, se detallan las dos diferencias principales entre la experimentación y comparación de metaheurísticas y la utilizada para la comparación de heurísticas o de modelos del capítulo 4. En el caso de las metaheurísticas, es necesario realizar experimentos de calibración para seleccionar los valores de los parámetros explicados en las secciones 5.3, 5.4 y 5.5. Una buena elección en los valores de estos parámetros puede mejorar de forma sustancial las prestaciones de la metaheurística. La segunda diferencia es que las metaheurísticas utilizan probabilidades y no son deterministas, es decir, cada ejecución puede obtener una solución diferente. Por lo tanto, para analizar la eficacia de dos metaheurísticas hacer solamente una comparación puntual no se considera una buena praxis, ya que podría ofrecer un muy buen resultado una primera ejecución solo por "suerte". Esta es la razón por la que en la comparación de la ejecución de metaheurísticas se utilizan réplicas de ejecución en las que una misma metaheurística se ejecuta múltiples veces

contra la misma instancia cambiando la semilla del generador de números aleatorios que tiene en su interior. De esa manera, al comparar 2 algoritmos y disponer de 5 ejecuciones de cada algoritmo contra la misma instancia, se puede realizar un análisis mas adecuado de qué metaheurística es realmente mejor.

5.6.1 Calibración de metaheurísticas

El procedimiento habitual para calibrar los algoritmos es seleccionar un conjunto de instancias, resolver el problema probando diferentes valores para sus parámetros y elegir los que obtienen mejores resultados (J. M. Framinan, Leisten y Ruiz García 2014). Para hacer esto de forma correcta se recurre al "diseño de experimentos" (Design Of Experiments - DOE) que se explica en la sección 4.5.1 y es necesario para una correcta experimentación. Para poder comprender el comportamiento de la variable respuesta ese experimento deberá ser analizado correctamente. Para ello se considera el conjunto de variables de entrada y sus posibles interacciones.

Como los conceptos de factor, nivel y variable ya fueron explicados, solo los aclararemos para esta experimentación:

- Factor: Las variables de estudio en el DOE, pero ahora en el contexto de una calibración de un algoritmo, serían sus parámetros y en algunos estudios también las variables utilizadas para la generación de instancias. Un ejemplo de factor puede ser el parámetro *Strength* = (2,3,4) y un ejemplo de factor de instancia puede ser *Distribución* = (cor,uni).
- Nivel: Los distintos valores que se van a tener en el experimento para cada factor. Pueden ser cuantitativos o cualitativos. El ejemplo *s* tiene 3 valores, el ejemplo *Distribución* tiene dos valores.
- Variable respuesta: vuelve a ser la calidad de la solución usando directamente la desviación porcentual relativa (RPD).

Para la calibración de las tres metaheurísticas se va a realizar una experimentación factorial completa (*full-factorial*), todas las posibles combinaciones de todos los posibles valores de los parámetros se van a probar. Cada ejecución de un algoritmo con un conjunto concreto de valores para sus parámetros, con una réplica específica (5 réplicas con 5 semillas diferentes), un tiempo concreto y resolviendo una instancia diferente, es lo que conocemos como tratamiento. Por tanto, cada tratamiento es diferente de todos los demás al menos en uno de estos valores.

Los resultados de esta experimentación se analizarán utilizando un análisis de la varianza (ANOVA), en el que se validan los supuestos ANOVA, este proceso se explicó en 4.5.1. Todo el proceso y el análisis nos permitirá discernir si existen factores con un impacto significativo en la variable respuesta.

El **criterio de parada** de todos los algoritmos para todas las ejecuciones de calibración ha sido el mismo, se usa el criterio de parada *CP* con valor $\rho = 90$ (más detalles sobre *CP* en la sección 4.5.2).

El proceso puede requerir varios análisis ANOVA, en estos análisis se estudia cada factor y las interacciones entre ellos. Se estudian los parámetros del algoritmo y además se incluyen todos los factores de instancia: número de trabajos (n), número de máquinas (m), réplica de la instancia ($instRep$), Distribución = cor de correlacionada y uni de uniforme (Dis), factor de corrección de disponibilidad ($rFactor$). Tener todos estos factores permite obtener un mejor análisis comprobando las interacciones entre los factores de la instancia y los parámetros de los algoritmos. En este primer análisis se revisa si existen interacciones entre los factores y qué factores influyen más en la variable respuesta. Tras este análisis se fija el parámetro más significativo y se apartan los resultados del estudio para, a continuación, volver a realizar otro análisis ANOVA completo de los datos restantes. Este nuevo análisis nos permite comprobar la significación estadística de los demás

parámetros, fijar parámetros si es necesario, etc. Esto se repite hasta que no existen más parámetros estadísticamente significativos.

Para seleccionar el algoritmo constructivo para obtener la solución inicial seleccionamos las heurísticas que presentaron un mejor comportamiento en la evaluación computacional realizada en el capítulo 4. Se han elegido la heurística $B10$, la heurística NEH_{nr} con la ordenación SRC_t y la NEH_{cf} con la ordenación $KK1$. Las razones es que son las que ofrecen mejores resultados frente a las demás. $B10$ es muy eficiente, NEH_{nr} ofrece un buen balance eficiencia eficacia, NEH_{cf} es la más eficaz pero la menos eficiente. Estos tres métodos se han elegido como posibles constructores iniciales para los algoritmos de trayectoria.

5.6.2 Calibración ILS

El algoritmo ILS propuesto está basado en el algoritmo de Stützle (1998), tanto en el trabajo original como en el libro posterior sobre metaheurísticas (Lourenço, Martin y Stützle 2019). El trabajo original se centra más en obtener las mejores posibles soluciones para el problema PFSP y las instancias de Taillard que en realizar y explicar una calibración al uso.

Los parámetros utilizados originalmente son: una NEH como constructor inicial, una perturbación que son algunas combinaciones de unos pocos movimientos en vecindario, varios criterios de aceptación, la temperatura T, dos vecindarios para la búsqueda local (intercambio o *swap* e inserción o *insert*) y dos estrategias de búsqueda local, (primera mejora o *first* y mejor solución o *best*). El resultado de su calibración es que la mejor perturbación son dos swaps consecutivos, la mejor búsqueda local es en el vecindario de inserción con estrategia de la primera mejora, el mejor criterio de aceptación es el $ConstTemp(s, s'')$ o de temperatura constante, aunque la temperatura T es un factor no significativo.

Para el ILS propuesto en esta tesis, para el problema del PFSPR, se necesitan los mismos parámetros más algunos adicionales que se detallan a continuación:

1. Solución inicial: se selecciona entre 3 heurísticas propuestas en el capítulo 4 (*Constructor* = $B10$, NEH_{nr} y NEH_{cf}).
2. Fuerza de la perturbación: Representa cuantas operaciones se van a realizar sobre la solución para perturbarla, a mayor fuerza mayor diversificación (*Strength = 2, 3, 4 y 5*).
3. Vecindario de perturbación: Se prueban los vecindarios de intercambio e inserción (*Perturbation neighbourhood = swap, insert*)
4. Vecindario búsqueda local: Se prueban los vecindarios de intercambio e inserción (*Neighbourhood Local Search = swap, insert*)
5. Tipo búsqueda local: Este parámetro marca la estrategia de la búsqueda local, si se busca el mejor elemento o la primera mejora (TypeLS = first, best).
6. Tipo de criterio de aceptación y temperatura: Utilizamos el criterio de temperatura constante con las temperaturas (T = 0,3, 0,4 y 0,5).

El número de tratamientos que vamos a tener en una calibración de este ILS depende de los factores y niveles que se usan: tres constructores, 4 valores para la fuerza de la perturbación, 2 valores para el vecindario de la perturbación, 2 valores para el vecindario de la búsqueda local, 2 valores para la estrategia de la búsqueda local y tres valores para la temperatura. Además debemos contar con el número de réplicas (5) y el número de instancias de calibración que tenemos para cada set de instancias (48, 96, 96). Los tratamientos serán por lo tanto: $3 \times 4 \times 2 \times 2 \times 2 \times 3 \times 5 \times (48 + 96 + 96) = 1.440 \times 240 = 345.600$.

5.6.3 Resultado calibración ILS

El primer análisis ANOVA para la calibración del ILS, se presenta en la tabla 5.1. En esta tabla, se muestran los resultados con los factores del algoritmo y sus interacciones. Dado el impacto altamente significativo de algunos factores, no se puede emplear el *p-Value*, por lo que se recurre al F-value para identificar qué parámetros son más significativos. El parámetro más representativo es la forma de aplicar la búsqueda local (*TypeLS*), el segundo es el tipo de constructor (*ConstructorOptions*) y se aprecia que existe un impacto estadísticamente significativo en la interacción entre ambos.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
ConstructorOptions	2	29201,23	14600,62	**12608,72**	0,0000
Strength	3	73,06	24,35	21,03	0,0000
T	2	0,35	0,18	0,15	0,8580
PerturbationNB	1	331,06	331,06	285,90	0,0000
NeighbourhoodLS	1	1321,10	1321,10	1140,87	0,0000
TypeLS	1	27868,00	27868,00	**24066,09**	0,0000
Interactions					
ConstructorOptions:Strength	6	1,91	0,32	0,27	0,9490
ConstructorOptions:T	4	0,40	0,10	0,09	0,9869
ConstructorOptions:PerturbationNB	2	1,34	0,67	0,58	0,5615
ConstructorOptions:NeighbourhoodLS	2	1238,19	619,10	534,64	0,0000
ConstructorOptions:TypeLS	2	11275,05	5637,52	**4868,42**	0,0000
Strength:T	6	1,07	0,18	0,15	0,9883
Strength:PerturbationNB	3	35,27	11,76	10,15	0,0000
Strength:NeighbourhoodLS	3	7,17	2,39	2,06	0,1027
Strength:TypeLS	3	2,08	0,69	0,60	0,6161
T:PerturbationNB	2	0,03	0,01	0,01	0,9879
T:NeighbourhoodLS	2	0,52	0,26	0,22	0,7986
T:TypeLS	2	0,12	0,06	0,05	0,9500
PerturbationNB:NeighbourhoodLS	1	167,36	167,36	144,53	0,0000
PerturbationNB:TypeLS	1	1,14	1,14	0,98	0,3211
NeighbourhoodLS:TypeLS	1	775,28	775,28	669,52	0,0000
Residuals	345260	399803,49	1,16		

Tabla 5.1: Tabla ANOVA tipo III con los resultados de la calibración del algoritmo *ILS*.

Se proporcionan gráficos de medias en las figuras 5.3, donde se incluyen los intervalos de confianza de Tukey HSD al 95% para los diferentes parámetros. Las figuras están ordenadas de acuerdo a su significación. A partir de los datos y la gráfica, se puede observar un importante impacto estadísticamente significativo de la estrategia de la búsqueda local. Elegir quedarse con la primera mejora ($first$) es significativamente mejor, algo que se puede entender al considerar que las instancias con más de 200 trabajos tendrían que hacer 200 operaciones en cada iteración de la búsqueda local para obtener una sola mejora. El segundo factor es el constructor, donde NEH_{nr} es claramente superior, mientras que la heurística $B10$ resulta claramente peor.

En la tabla 4.7 donde aparecen los resultados promedio de las diferentes NEH, se aprecia que la heurística NEH_{cf} resulta ser la mejor heurística constructiva que se ha propuesto. Pero incluir una heurística que es tan costosa en términos de tiempo dentro de la metaheurística no resulta una buena opción en instancias medianas o grandes. Esta heurística puede consumir gran parte o la totalidad del tiempo haciendo que la metaheurística no tenga tiempo suficiente para realizar mejoras.

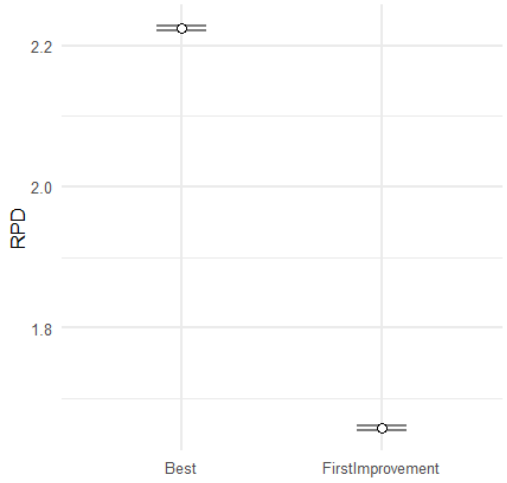
Los siguientes parámetros más significativos son el vecindario de la perturbación y el vecindario de la búsqueda local en ese orden, siendo en ambos casos mejor el vecindario de intercambio

Parámetro	Niveles	Original	Propuesta
Constructor	3	NEH	NEH_{nr}
Strength	4	2	2
T	3	0,5 (ns)	0,3 (ns)
Perturbation Neighbourhood	2	Swap	Swap
Local Search Neighbourhood	2	Insert	Swap
Type of Local Search	2	first	first

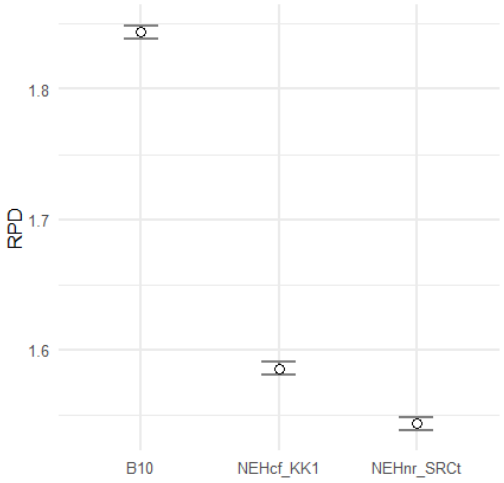
Tabla 5.2: Valores obtenidos en las calibraciones del ILS.

(*swap*). Finalmente, encontramos la fuerza de la perturbación ($Strength$) con valores muy similares y la temperatura (T) que no resulta ser estadísticamente significativa. Los valores que obtienen un menor RPD y son estadísticamente significativos son los que se eligen como los mejores valores para el algoritmo. Estos valores están en la tabla 5.2 en la columna "Propuesta". En esta misma tabla en la columna "Original", se muestran los valores que los autores obtuvieron en sus calibraciones del algoritmo. En la tabla, cuando no existen diferencias estadísticamente significativas entre los posibles valores se indica con "(ns)" de *no significativo*, por ejemplo el factor T.

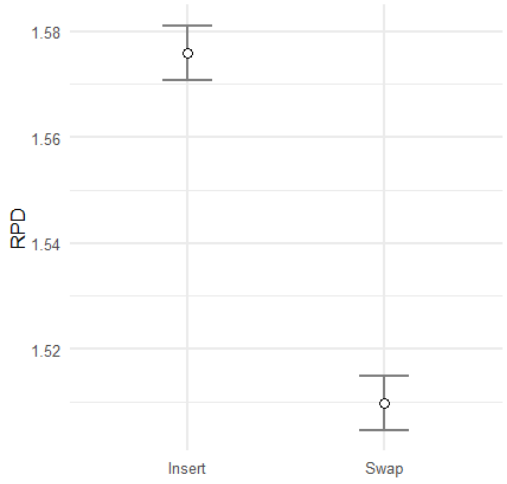
Durante el proceso de calibración se han detectado algunas interacciones que el análisis ANOVA nos marcaba como relevantes, sin embargo, tras revisarlas se ha constatado que su impacto es menor que el de los factores de forma independiente y no alteran la decisión de qué valores deberían emplearse.



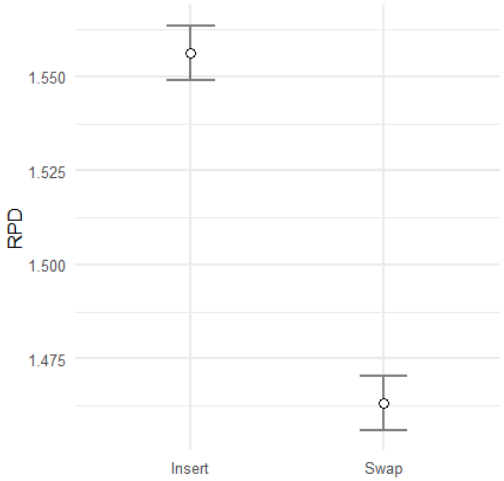
(a) Tipo de búsqueda local (*TypeLS*).



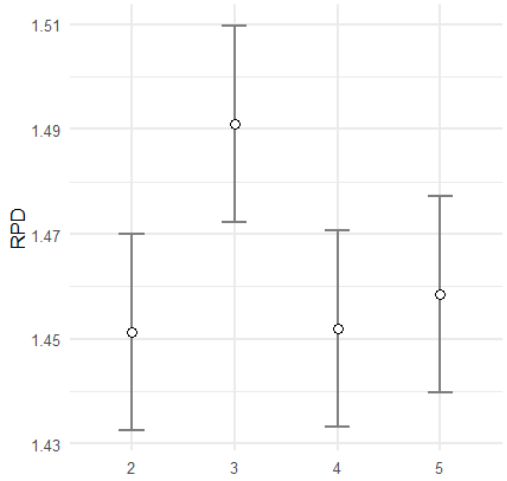
(b) Solución inicial (*ConstructorOptions*)



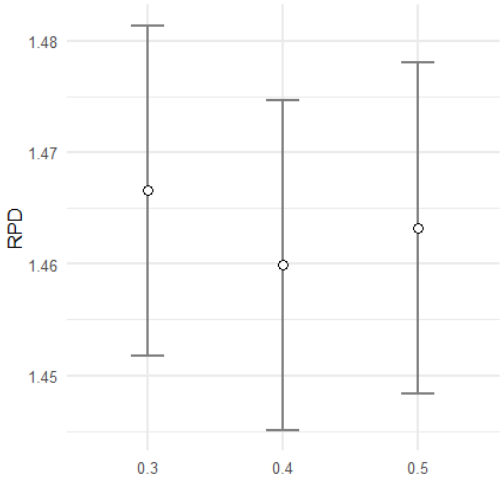
(c) Vecindario de la perturbación (*PerturbationNB*)



(d) Vecindario de la BL (*NeighbourhoodLS*)



(e) Fuerza de la perturbación (*Strength*)



(f) Temperatura (*T*)

Figura 5.3: Gráficos de medias con los intervalos HSD de Tukey del RPD promedio para cada parámetro del algoritmo *ILS* en el set *All*.

5.6.4 Calibración IG

El algoritmo iterativo voraz (IG) propuesto para este problema es una adaptación del algoritmo IG de Ruiz y Stützle (2007), que se detalla en la sección 5.4. Aunque este algoritmo se propuso inicialmente para el problema PFSP con el objetivo makespan, se ha utilizado en multitud de problemas. En el trabajo original Ruiz y Stützle (2007) sobre las instancias de Taillard (sin recursos), se presenta un concienzudo análisis ANOVA factorial completo con todos los valores que los autores consideran para sus parámetros, basándonos en el trabajo inicial calibramos los parámetros para el problema PFSPR.

En el trabajo original se utiliza como constructor una heurística constructiva NEH y dos parámetros: el parámetro d que representa la fuerza de la destrucción y la temperatura T. Este algoritmo tiene el mismo criterio de aceptación de temperatura constante que el algoritmo *ILS*, ambos basados en el *SA* (Osman y Potts 1989). Los valores que salen mejores en la calibración de los autores son $d = 4$ seguido de cerca por 3, y respecto a la temperatura T eligen $0,5$ aunque aparece como un factor no significativo en la variable respuesta.

El IG propuesto en esta tesis, considera los siguientes parámetros y valores:

1. Solución inicial: se utilizan 3 heurísticas propuestas en el capítulo 4 (*Constructor* = $B10$, NEH_{nr} y NEH_{cf}).
2. Fuerza de la destrucción (d): señala cuantos trabajos se van a destruir en la fase de destrucción-construcción (*d = 3, 4 y 5*).
3. Tipo de criterio de aceptación y temperatura: se utiliza el criterio de temperatura constante con las temperaturas (t = 0,3, 0,4 y 0,5).

En el caso de la calibración del IG al solo tener 3 parámetros vamos a tener una calibración algo más sencilla. El número de tratamientos depende del número de niveles y factores que se usan, tenemos: tres constructores, 3 valores para la fuerza de la destrucción (d) y tres valores para la temperatura. Se consideran además el número de réplicas (5) y el número de instancias de entrenamiento que tenemos para cada set de instancias (48, 96, 96). Los tratamientos serán: $3 \times 3 \times 3 \times 5 \times (48 + 96 + 96) = 135 \times 240 = 32.400$

5.6.5 Resultado calibración IG

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
ConstructorOptions	2	677,59	338,80	**638,99**	0,0000
d	2	0,16	0,08	0,15	0,8581
T	2	0,30	0,15	0,28	0,7531
Interactions					
ConstructorOptions:d	4	10,21	2,55	4,82	0,0007
ConstructorOptions:T	4	0,15	0,04	0,07	0,9912
d:T	4	0,18	0,04	0,08	0,9874
Residuals	32149	17045,67	0,53		

Tabla 5.3: Tabla ANOVA tipo III con los resultados de la calibración del algoritmo *IG*

En la tabla 5.3 se presentan los resultados completos del ANOVA factorial completo realizado para la calibración del algoritmo IG. En la tabla se observa que el factor con mas impacto en la variable respuesta es el tipo de constructor (*ConstructorOptions*), también se aprecia que solo existe interacción entre el Constructor y la fuerza de la destrucción (d).

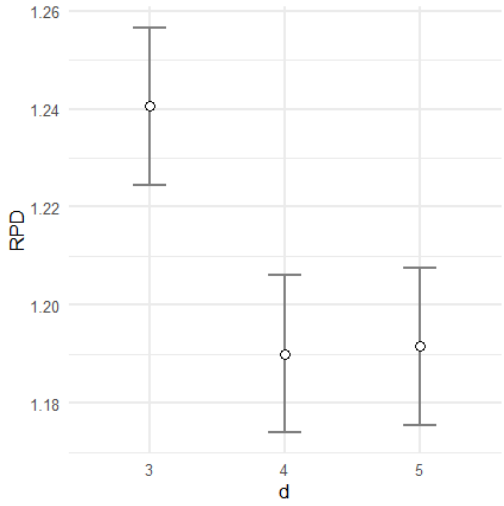
Se aportan gráficos de medias en las figuras 5.4. Estos gráficos incluyen los intervalos de confianza de Tukey HSD al 95% para los tres factores. Las figuras se presentan ordenadas de acuerdo al nivel de significación de cada parámetro. En estos gráficos, se puede observar la influencia de cada parámetro en los resultados obtenidos y la variabilidad asociada a cada uno de ellos. En primer lugar se ve que el constructor NEH_{nr} es el mejor, así como que el algoritmo $B10$ es el peor, estos resultados son similares a los obtenidos en la calibración del *ILS*. Para obtener la significación real de los otros parámetros se realiza un nuevo análisis ANOVA con el constructor fijado y se obtienen las gráficas para d y para T. Es interesante ver que los resultados en el caso de d y T, son casi idénticos a los resultados de la calibración original a pesar de estar tratando un problema diferente.

Parámetro	Niveles	Original	Propuesta
Constructor	3	NEH	NEH_{nr}
d	3	4	4
T	3	0,5 (ns)	0,4 (ns)

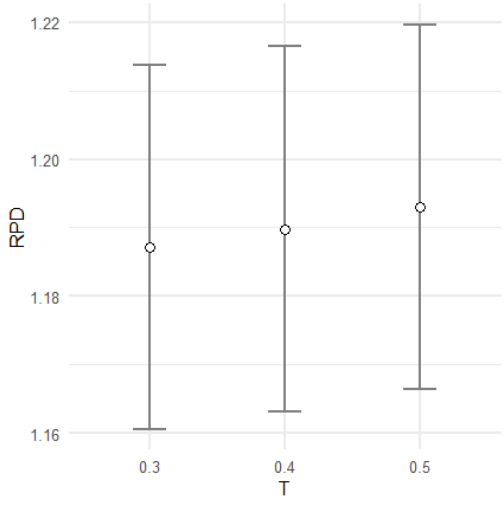
Tabla 5.4: Valores obtenidos en la calibración del IG.

Los valores que obtienen un menor valor del RPD y son estadísticamente significativos, son los que finalmente se seleccionan, cuando no hay una elección clara en muchos casos se elige el que tenga la menor media o el que ha usado el autor en el trabajo original. Estos valores están reflejados en la tabla 5.4, se ven en la columna "Propuesta" los que se han elegido en esta calibración y en la columna "Original" los que eligieron los autores. En la tabla no se aprecian los detalles como en las figuras, pero añadiendo la marca (ns) de "no significativo" podemos al menos reflejar aquellos factores en los que ningún valor resulto significativamente mejor ni peor.

(a) Solución inicial



(b) Fuerza de la destrucción (d)



(c) Temperatura (T)

Figura 5.4: Gráficos de medias con los intervalos HSD de Tukey del RPD promedio para cada parámetro del algoritmo *IG* en el set *All*.

5.6.6 Calibración HGA

El algoritmo HGA, Hybrid Genetic Algorithm, que se calibra a continuación, es el algoritmo propuesto por Laribi, Yalaoui y Sari (2019) descrito en la sección 5.5. Esta metaheurística se propuso para el problema PFSPR con recursos no renovables con el objetivo de optimizar el *makespan* y es adaptada al problema del PFSPR que se estudia en esta tesis. La metaheurística tiene muchos parámetros, algunos de ellos son parámetros clásicos de los algoritmos genéticos, pero tiene también varias formas de integrar la búsqueda local.

La calibración que se realiza está basada en la ofrecida originalmente por los autores. Sin embargo, es importante señalar que los autores originales realizaban sus comparaciones basándose en el número de iteraciones, y no en el tiempo. Por tanto, se espera obtener resultados bastante diferentes a pesar de ser la metaheurística que se calibró originalmente para el problema más parecido al que se estudia en esta tesis.

El HGA tiene los siguientes parámetros y sus correspondientes valores:

1. Población inicial: Se estudian dos formas de generar la población inicial: completamente aleatoria y con la heurística de *Palmer*, la NEH y la NEH_{kk1}, que fueron las propuestas para el trabajo original de Laribi, Yalaoui y Sari (2019). Selecciona una u otra con el parámetro *PopHasHeur = true y false* siendo true utilizar heurísticas.
2. Tamaño de población: cuantos individuos existen cada vez que empieza una iteración (*PopSize = 100, 200*).
3. Operador de cruce: tipo de operador de cruce (*crossover = OPX y TPX*).
4. Operador de mutación: tipo de operador de mutación (*mutation = insert y swap*).
5. Probabilidad de cruce: P_c (*Pcrossover = 0,8 y 1*)
6. Probabilidad de mutación: P_m (*Pmutation = 0,1 y 0,2*)
7. Forma de aplicar la Búsqueda Local: este parámetro marca como se aplica la búsqueda local y puede tomar 3 valores:
   a. Sin búsqueda local, no se aplica LS (*typeLocalSearch = "NO"*)
   b. Aplica búsqueda local solo al mejor individuo de la población según una probabilidad (0,1) (*typeLocalSearch = 0,1*).
   c. Aplica búsqueda local a un porcentaje de la población (30%, 50% o 80%) eligiendo un porcentaje de las mejores soluciones. (*typeLocalSearch = 30%, 50% o 80%*)

El número de tratamientos que se calculan para HGA es muy grande por tener muchos parámetros, además debemos incluir el número de réplicas (5) y el número de instancias de calibración que tenemos para cada set de instancias (48, 96, 96). Los tratamientos serán por lo tanto: $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 4 \times 5 \times (48 + 96 + 96) = 1.280 \times 240 = 307.200$.

5.6.7 Resultado calibración HGA

El primer análisis de la varianza (ANOVA) para la calibración del algoritmo HGA se presenta en la tabla 5.6. En esta tabla se muestran los resultados asociados a los factores del algoritmo y sus interacciones. Dado que el *p-Value* es 0,00 se recurre al valor F para determinar qué parámetros son más significativos.

Factores	Niveles	Original	Propuesta
Tipo de población inicial	2	Heurística	Heurística
Tamaño de población	2	200	200
Operador de cruce	2	TPX	OPX (ns)
Operador de mutación	2	insert	insert
Probabilidad de cruce	2	0,9	0,8 (ns)
Probabilidad de mutación	2	0,15	0,2
Forma de aplicar la búsqueda local	4	100%	Probab (0,1)

Tabla 5.5: Tabla con los factores analizados y los mejores valores obtenidos para *HGA*

Además de la tabla, para facilitar la comprensión de los datos, se proporcionan gráficos de medias en las figuras 5.5, que incluyen los intervalos de confianza de Tukey HSD al 95% para cada parámetro. Las gráficas se han ordenado en función de la relevancia de cada parámetro.

El factor con mayor impacto en la variable respuesta es el tipo de población inicial. Generar la población inicial usando heurísticas es claramente mejor (*PopHasHeuristics = True*) como se aprecia en la figura 5.5a. El segundo parámetro con mayor impacto es el la forma de aplicación de la búsqueda local siendo la mejor opción la que considera aplicar la búsqueda local sobre el mejor elemento con una probabilidad de 0,1.

Existe una interacción entre estos dos parámetros, aunque su impacto no es muy grande. Para ilustrar esto, se facilita la figura 5.6a donde se observa que para estos dos parámetros los valores *0,1 y true* son significativamente mejores que las demás combinaciones de valores. La no se aprecia en la figura 5.6b que sea relevante la interacción entre el tamaño de la población y la forma de aplicar la búsqueda local.

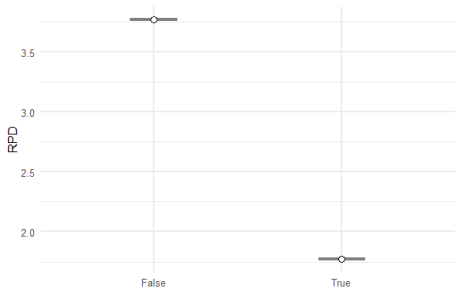
Una vez fijados estos parámetros se realiza otro análisis ANOVA con los datos restantes para los demás parámetros. El siguiente factor más determinante pasa a ser la probabilidad de mutación (P_m - *Pmutation*) siendo el mejor valor 0,2. El tipo de mutación es el siguiente factor más significativo donde la mejor opción es fijarlo en *insert*.

A continuación, tenemos el tamaño de la población (*PopSize*), el tipo de operador de cruce (*crossover*) y la probabilidad de cruce (P_c - Pcrossover). En estos tres factores que están en las figuras 5.5e, 5.5f, 5.5g, se aprecia como los valores son cada vez menos significativos y los intervalos que no se llegan a solapar son solamente los del tamaño de la población donde una población más grande ofrece mejores resultados. En la columna "Propuesta" de la tabla 5.5 están los valores seleccionados en esta calibración para el conjunto de todas las instancias.

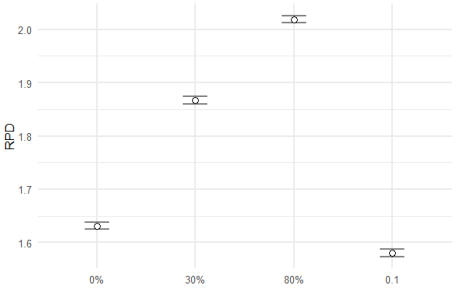
En base a los resultados de los experimentos, se ha calibrado cada uno de los tres algoritmos propuestos. Se han fijado los valores de los parámetros a la mejor opción posible encontrada. Los algoritmos presentados con los valores de sus parámetros fijados son ILS_g, IG_g y HGA_g. Dado que este trabajo se compara principalmente con el trabajo de Laribi, Yalaoui y Sari (2019) y teniendo en cuenta que el resultado de nuestra calibración es muy distinto, se ha optado por añadir el algoritmo *HGA* también con la calibración original, al que nos referimos por HGA_o.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
typeLocalSearch	3	268958,24	89652,75	**31479,07**	0,0000
PopSize	1	1343,51	1343,51	471,73	0,0000
Pcrossover	1	195,73	195,73	68,72	0,0000
crossover	1	0,03	0,03	0,01	0,9182
Pmutation	1	122,66	122,66	43,07	0,0000
mutation	1	58,09	58,09	20,40	0,0000
PopHasHeuristics	1	305464,93	305464,93	**107255,51**	0,0000
Interactions					
typeLocalSearch:PopHasHeuristics	3	177008,66	59002,89	**20717,22**	0,0000
PopSize:PopHasHeuristics	1	841,22	841,22	**295,37**	0,0000
Pcrossover:PopHasHeuristics	1	144,55	144,55	50,75	0,0000
crossover:PopHasHeuristics	1	1,94	1,94	0,68	0,4086
Pmutation:PopHasHeuristics	1	5,19	5,19	1,82	0,1772
mutation:PopHasHeuristics	1	1,85	1,85	0,65	0,4207
PopSize:Pcrossover	1	55,42	55,42	19,46	0,0000
typeLocalSearch:PopSize	3	2376,48	792,16	**278,14**	0,0000
PopSize:mutation	1	2,58	2,58	0,90	0,3414
PopSize:Pmutation	1	2,71	2,71	0,95	0,3294
typeLocalSearch:Pcrossover	3	231,61	77,20	27,11	0,0000
Pcrossover:mutation	1	0,00	0,00	0,00	0,9921
Pcrossover:Pmutation	1	0,31	0,31	0,11	0,7404
typeLocalSearch:mutation	3	59,57	19,86	6,97	0,0001
typeLocalSearch:Pmutation	3	247,22	82,41	28,93	0,0000
Pmutation:mutation	1	0,35	0,35	0,12	0,7249
Residuals	306888	874020,58	2,85		

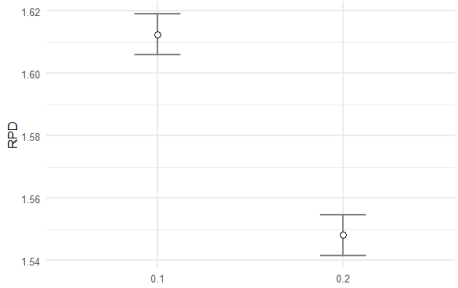
Tabla 5.6: Tabla ANOVA tipo III con los resultados de la calibración del algoritmo *HGA*.



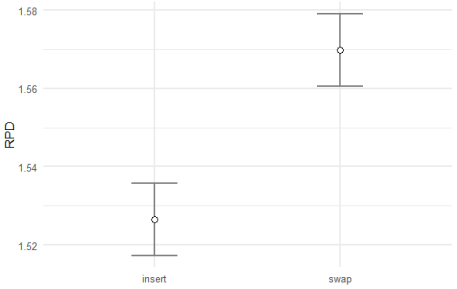
(a) Población inicial (*PopHasHeuristics*).



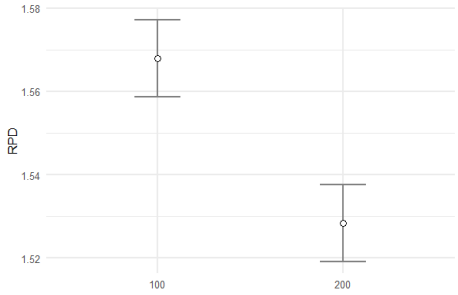
(b) Forma de aplicar la Búsqueda Local (*typeLocalSearch*)



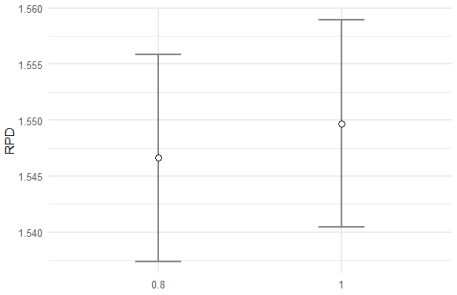
(c) Probabilidad de mutación (*Pmutation*)



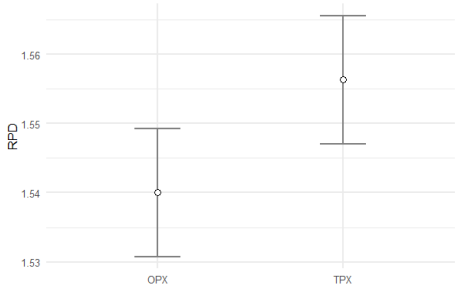
(d) Tipo de mutación (*mutation*)



(e) Tamaño de la población (*PopSize*)

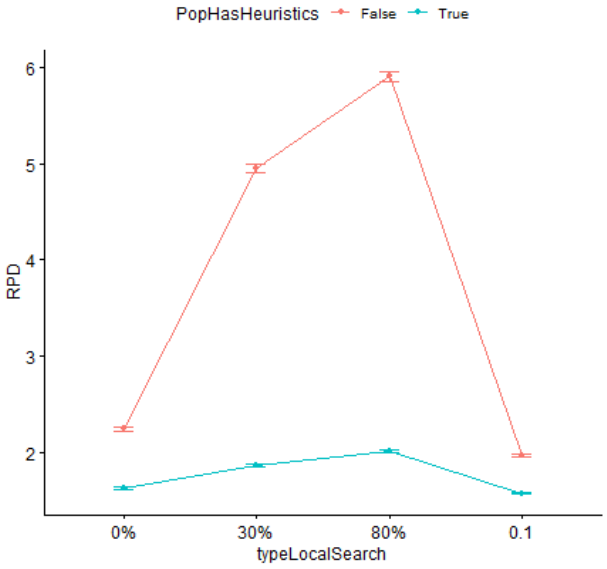


(f) Probabilidad de cruce (*Pcrossover*)

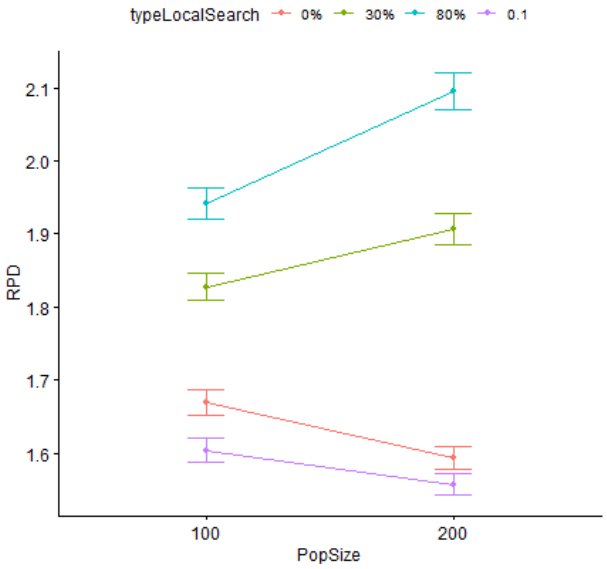


(g) Tipo de operador de cruce (*crossover*).

Figura 5.5: Gráficos de medias con los intervalos HSD de Tukey del RPD promedio para cada parámetro del algoritmo *HGA* en el set *All*.



(a) Interacción entre constructor inicial y la forma de aplicar la búsqueda local.



(b) Interacción entre el tamaño de la población y la forma de aplicar la búsqueda local.

Figura 5.6: Gráficos de las interacciones entre los parámetros del algoritmo *HGA* en las calibraciones.

5.7 Comparación computacional entre metaheurísticas

En esta sección se presenta una comparación exhaustiva de las tres metaheurísticas propuestas para resolver el problema PFSPR, incluyendo además el algoritmo original de Laribi, Yalaoui y Sari (2019). Para ello se ha realizado una experimentación con todas las instancias de test planteadas en la sección 4.5.1 separadas en tres sets de instancias: Taillard, VRFS (VRF small) y VRFL (VRF large). Cada conjunto contiene 480 instancias, lo que hace un total de 1440 instancias.

Los resultados computacionales van a mostrar dos ideas: en primer lugar el comportamiento de cada metaheurística en cada uno de los conjuntos de instancias y en segundo lugar la eficiencia de las metaheurísticas al incrementar el tiempo de computación. Para analizar cómo las metaheurísticas se comportan con respecto al tiempo, se ha recurrido de nuevo al criterio de parada *CP* de la fórmula 4.18. A diferencia de la calibración, aquí se emplean tres valores diferentes para ρ = (90, 135, 180). Además debido a la naturaleza estocástica de las metaheurísticas se han realizado 5 ejecuciones para cada instancia (5 réplicas) con el objetivo de obtener resultados más robustos y representativos.

Tamaño	$\rho = 90$	$\rho = 135$	$\rho = 180$
20x10	9s	13,5s	18s
200x20	180s	270s	360s
700x60	1.890s	2.835	3.780s

Tabla 5.7: Ejemplo de tiempos de ejecución para las metaheurísticas con diferentes valores en ρ.

El número de tratamientos es determinado por el número de configuraciones diferentes, el número de valores de ρ utilizados, el número de instancias y el número de réplicas, implica realizar un total de $4 \times 3 \times 1.440 \times 5 = 86.400$ ejecuciones. Para ilustrar los tiempos de cómputo reales se ofrecen tres ejemplos en la tabla 5.7, podemos ver como crece el tiempo del criterio de parada en base al tamaño de la instancia y a ρ

Los experimentos se han realizado en un clúster de ordenadores con procesadores "Intel Xeon Gold 5220" a 2,20 GHz y hasta 216 cores, generando máquinas virtuales con Windows 10, que simulan ordenadores con un procesador de 4 núcleos y 16 Gb de memoria RAM. Los algoritmos utilizados se han codificado usando C# y Visual Studio 2020, para facilitar el desarrollo y la ejecución de algoritmos se ha utilizado la plataforma de desarrollo de algoritmos que se presenta en el capítulo 6 (FACOP).

En la siguiente sección donde se comparan las distintas metaheurísticas se ha utilizado el análisis de la varianza (ANOVA) siguiendo las mismas pautas marcadas en 4.5.1, se han comprobado los supuestos ANOVA que salen correctos, pero este contenido se ha desplazado a los apéndices en la sección A.4 para mayor claridad.

5.7.1 Resultados en las instancias de Taillard

La tabla 5.8 muestra la Desviación Porcentual Relativa (RPD) promedio para cada metaheurística en el conjunto de instancias Taillard. Los promedios en la tabla están agregados en base al tamaño de la instancia representado por el número de trabajos (n) y el número de máquinas (m). En ella se muestra que ILS_g es el mejor del conjunto. También se presenta el gráfico de medias con los intervalos de confianza HSD de Tukey al 95% en la figura 5.7. De la figura se extrae que el mejor rendimiento entre los cuatro algoritmos corresponde al HGA_g. En contraposición, el algoritmo HGA_o ocupa una posición intermedia, mientras que el IG_g tiene el rendimiento más bajo.

El orden de mejor a peor es ahora HGA_g, ILS_g, HGA_o y, finalmente, IG_g. Se puede afirmar que las diferencias entre algoritmos son estadísticamente significativas y el rendimiento varía ligeramente respecto a los resultados en las instancias VRFS. Por decirlo así es como si los algoritmos de trayectoria *ILS* e *IG* retrocedieran una posición respecto a los dos *HGA*.

n	m	HGA_g	HGA_o	IG_g	ILS_g
20	5	2,05	2,94	1,58	1,45
	10	1,62	4,07	1,71	1,18
	20	1,31	4,34	1,20	0,97
Promedio		1,66	3,78	1,49	1,20
50	5	0,56	1,06	2,54	1,60
	10	1,07	1,85	1,78	1,08
	20	1,31	2,28	1,67	1,32
Promedio		0,98	1,73	1,99	1,33
100	5	0,13	0,19	3,79	2,32
	10	0,31	0,42	2,31	1,37
	20	0,36	0,40	1,36	1,06
Promedio		0,27	0,34	2,49	1,59
200	10	0,24	0,25	3,44	2,47
	20	1,68	1,68	1,64	1,88
Promedio		0,96	0,97	2,54	2,17
500	20	0,76	0,76	0,55	0,83
Promedio Total		0,95	1,69	1,96	1,46

Tabla 5.8: Promedios RPD para cada metaheurística por grupos de instancias ($n \times m$) para el conjunto de instancias de Taillard.

En la figura 5.8, se observa la interacción entre los diferentes algoritmos y el tiempo de ejecución de los mismos en relación con la desviación porcentual relativa. En esta figura, los cuatro algoritmos muestran tendencias de mejora al incrementar el tiempo de cómputo, aunque a ritmos diferentes.

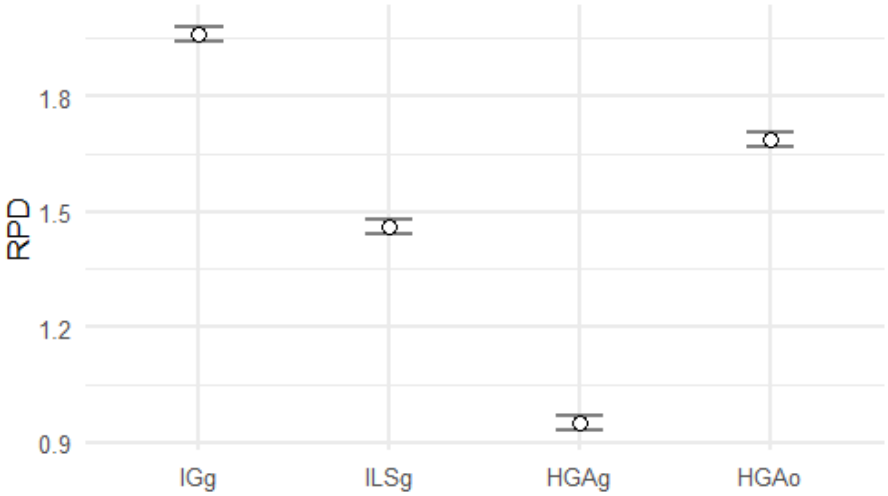


Figura 5.7: Gráfico de medias con intervalos de confianza Tukey 95% para los algoritmos en el conjunto las instancias de Taillard

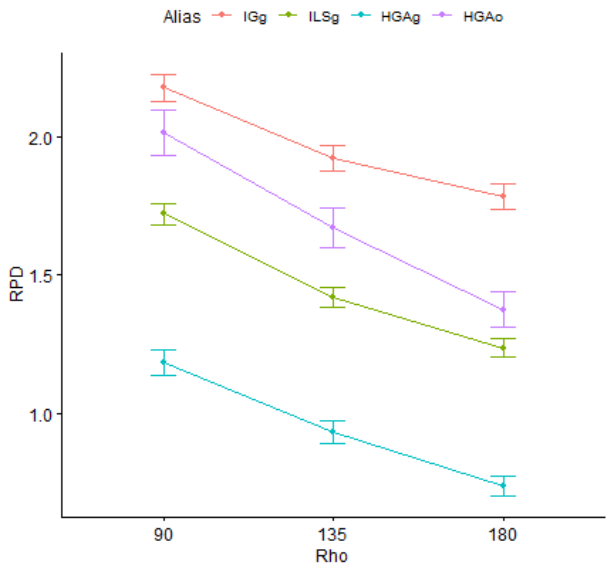


Figura 5.8: Interacción entre los algoritmos y ρ para las instancias de Taillard.

5.7.2 Resultados en las instancias VRF pequeñas

La tabla 5.9 muestra la Desviación Porcentual Relativa (RPD) promedio para cada metaheurística. Los datos en la tabla están agrupados por el tamaño de la instancia representado por n y m para el conjunto de instancias VRFS. En ella se ve que el ILS_g tiene los mejores resultados, pero le sigue de cerca el HGA_g, a más distancia se encuentran los otros dos algoritmos.

Además, se incluye el gráfico de medias que incluye los intervalos de confianza HSD de Tukey al 95% en la figura 5.9.

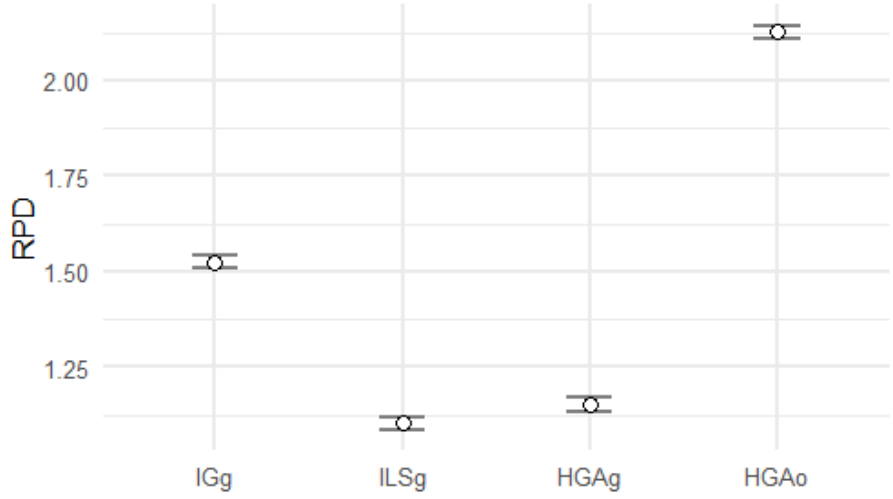


Figura 5.9: Gráfico de medias con intervalos de confianza Tukey 95% para los algoritmos en el conjunto de instancias VRFS

De acuerdo a la figura el rendimiento peor entre los cuatro algoritmos corresponde a HGA_o. Como se explicó anteriormente el algoritmo HGA_o utiliza la calibración original, lo que explica la distancia entre el valor HGA_o y HGA_g. Esto no es de extrañar pues HGA_o no ha sido calibrado para estas instancias y en su calibración original realizada en base a iteraciones empuja a la selección de componentes muy eficaces pero poco eficientes, es decir, operadores que mejoran la solución pero probablemente son costosos en tiempo. Entre los otros tres algoritmos, las distancias no son tan grandes: ILS_g resulta ser el mejor algoritmo, seguido del algoritmo HGA_g y finalmente encontramos el algoritmo IG_g.

En la figura 5.10 se muestra la interacción existente entre los diferentes algoritmos y el tiempo de ejecución de los mismos (ρ) sobre la desviación porcentual relativa. Todos los algoritmos muestran mejoras respecto a si mismos al incrementar el tiempo de cómputo. Sin embargo, podemos observar que los algoritmos ILS_g y HGA_g parten de puntos similares en $\rho = 90$ pero el ILS mejora más rápidamente en relación al tiempo, por lo que consigue ganar distancia respecto al HGA_g al aumentar el tiempo hasta $\rho = 180$. El algoritmo IG_g también parece mejorar más rápido, sin embargo, solo con más experimentación y valores de ρ más grandes se podría determinar que resultado esperar.

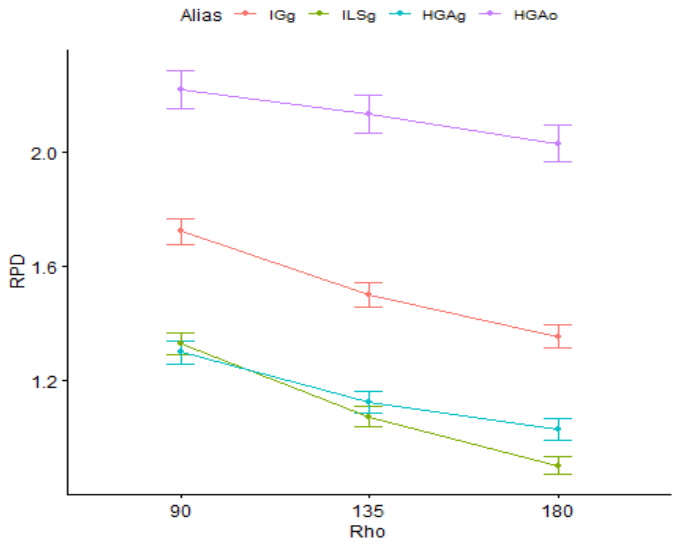


Figura 5.10: Interacción entre los algoritmos y ρ para el conjunto VRFS

n	m	HGA_g	HGA_o	IG_g	ILS_g
10	5	1,19	0,39	0,17	0,21
	10	0,74	0,61	0,27	0,18
	15	0,31	0,29	0,03	0,07
	20	0,10	0,03	0,00	0,06
Promedio		0,58	0,33	0,12	0,13
20	5	2,16	2,80	1,61	1,46
	10	1,82	3,84	1,60	1,22
	15	1,60	4,10	1,57	1,09
	20	1,39	4,38	1,14	1,03
Promedio		1,74	3,78	1,48	1,20
30	5	1,51	2,91	1,76	1,32
	10	1,47	3,26	1,83	1,28
	15	1,82	4,08	2,04	1,48
	20	1,83	3,76	1,79	1,27
Promedio		1,66	3,50	1,85	1,34
40	5	0,69	1,33	1,93	1,46
	10	1,16	1,93	1,84	1,11
	15	1,19	2,63	1,72	1,22
	20	1,60	2,97	1,83	1,49
Promedio		1,16	2,21	1,83	1,32
50	5	0,91	1,59	2,68	1,60
	10	0,95	1,70	1,90	1,33
	15	1,08	1,88	1,58	1,17
	20	1,22	1,85	1,58	1,07
Promedio		1,04	1,75	1,93	1,29
60	5	0,31	0,56	2,80	1,81
	10	0,68	1,11	1,91	1,14
	15	0,87	1,48	1,68	1,16
	20	1,02	1,55	1,35	1,21
Promedio		0,72	1,17	1,93	1,33
Promedio Total		1,15	2,13	1,53	1,10

Tabla 5.9: Promedios RPD para cada metaheurística por grupos de instancias ($n \times m$) para el conjunto de instancias VRFS.

5.7.3 Resultados en las instancias de VRF grandes

En la tabla 5.10 vemos que los resultados de HGA_o y HGA_g son los peores y son muy parecidos, existiendo solo muy pequeñas diferencias. En este caso la metaheurística IG_g obtiene mejores resultados que el ILS_g.

En la figura 5.11, se presenta el gráfico de medias que incluye los intervalos de confianza HSD de Tukey al 95%. Según este gráfico el algoritmo con el mejor rendimiento es el IG_g, seguido de cerca por el ILS_g. A una distancia considerable se encuentran el HGA_o y HGA_g, cuyos resultados son prácticamente idénticos y al encontrar solapamiento no existe diferencia estadísticamente significativa.

De forma similar a los casos anteriores las diferencias entre algoritmos son estadísticamente significativas, con la excepción de los algoritmos HGA_o y HGA_g, que no presentan diferencias significativas entre ellos. Esto implica que los algoritmos IG_g y ILS_g presentan un rendimiento superior en este conjunto de instancias VRFL.

Para analizar la interacción entre los diferentes algoritmos y el tiempo de ejecución de los mismos respecto a la desviación porcentual relativa se proporciona la figura 5.12. En ella, se aprecia que los algoritmos muestran mejoras con el incremento del tiempo de cómputo, siendo mayores en ambos *HGA* pero probablemente sea debido a que está mas alejada de la mejor solución. Sin embargo, estas mejoras son pequeñas, probablemente, porque el tiempo del que disponen las metaheurísticas para problemas tan grandes es demasiado pequeño. Esto se argumenta en la siguiente sección sobre el análisis del conjunto completo de instancias. En cuanto al algoritmo ILS_g, aunque su rendimiento es inferior al del IG_g, la diferencia entre el ILS_g y el HGA_g es mayor.

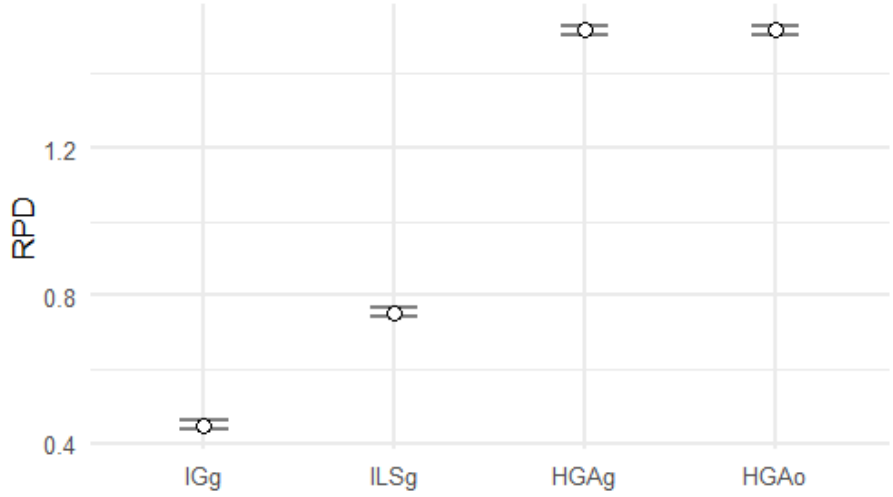


Figura 5.11: Gráfico de medias con intervalos de confianza Tukey 95% para los algoritmos en el conjunto las instancias VRFL

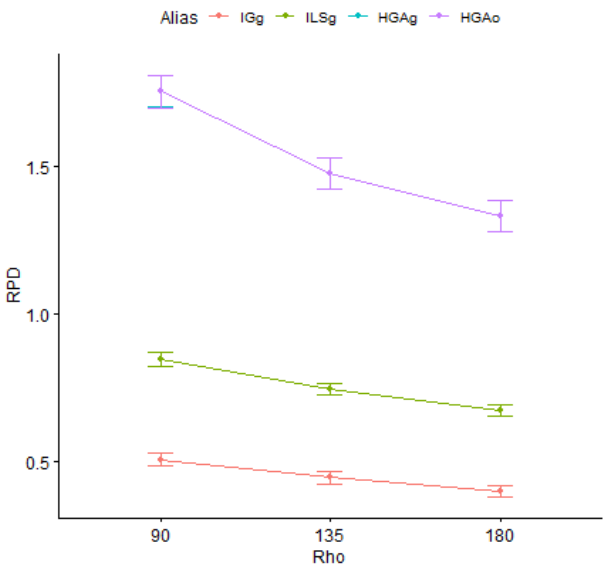


Figura 5.12: Interacción entre los algoritmos y ρ para las instancias VRFL.

n	m	HGA_g	HGA_o	IG_g	ILS_g
100	20	0,39	0,43	1,13	1,09
	40	0,64	0,59	0,93	0,77
	60	1,55	1,57	0,79	0,76
Promedio		0,86	0,86	0,95	0,88
200	20	1,43	1,42	1,65	1,80
	40	3,36	3,35	0,43	0,99
	60	3,86	3,85	0,42	0,97
Promedio		2,88	2,88	0,83	1,25
300	20	0,98	0,99	0,70	1,09
	40	2,27	2,28	0,27	0,80
	60	2,46	2,47	0,22	0,69
Promedio		1,90	1,91	0,40	0,86
400	20	0,82	0,81	0,52	0,85
	40	1,68	1,71	0,26	0,71
	60	2,20	2,21	0,20	0,65
Promedio		1,57	1,58	0,32	0,74
500	20	0,83	0,82	0,58	0,85
	40	1,53	1,54	0,23	0,63
	60	1,98	1,99	0,17	0,53
Promedio		1,45	1,45	0,33	0,67
600	20	0,82	0,82	0,43	0,68
	40	1,30	1,29	0,15	0,45
	60	1,79	1,77	0,18	0,55
Promedio		1,30	1,29	0,25	0,56
700	20	0,71	0,73	0,47	0,74
	40	1,28	1,27	0,19	0,50
	60	1,53	1,53	0,14	0,44
Promedio		1,17	1,18	0,27	0,56
800	20	0,64	0,64	0,44	0,71
	40	1,05	1,05	0,13	0,42
	60	1,35	1,35	0,14	0,40
Promedio		1,01	1,01	0,24	0,51
Promedio Total		1,52	1,52	0,45	0,75

Tabla 5.10: Promedios RPD para cada metaheurística por grupos de instancias ($n \times m$) para el conjunto de instancias VRFL.

5.7.4 Resultados en el conjunto de todas las instancias

Ahora se analiza los resultados de todas las instancias en conjunto: Taillard, VRFS y VRFL. La tabla para todas las instancias no sería otra que las tres tablas ya presentadas para cada set de instancias: 5.8, 5.9 y 5.10.

Revisando el gráfico de medias que se muestra en la figura 5.13 se observa que el mejor rendimiento en este conjunto total es el algoritmo ILS_g, seguido por el HGA_g. A continuación, se sitúa el IG_g, y finalmente el algoritmo con el rendimiento más bajo es el HGA_o. Estos resultados proporcionan una visión global sobre el rendimiento de los cuatro algoritmos en todo el conjunto de instancias. Cabe destacar que a pesar de las diferencias en los resultados obtenidos en los conjuntos individuales de Taillard, VRFS y VRFL, en el conjunto total las diferencias entre algoritmos son claras y significativas. Esto es porque aunque en el set de instancias de Taillard HGA_g sea algo mejor que el ILS_g, el rendimiento del ILS_g es consistentemente mejor en los demás sets.

Especialmente interesantes son las interacciones de los algoritmos con los factores de instancia. La primera interacción es la que se observa en la figura 5.15a, es un gráfico de medias para evaluar la interacción entre la disponibilidad de recursos y los algoritmos. Estos resultados reflejan que todos los algoritmos se ven afectados por ese factor de igual manera. Sin embargo, en la figura 5.15b se puede observar cómo los algoritmos reaccionan de forma diferente a la escasez de recursos. Los algoritmos genéticos son más resistentes a la escasez de recursos mientras que los algoritmos de traza se ven más perjudicados. Las interacciones entre los algoritmos y la n no son claros porque están mezclados los sets VRF y Taillard, pero no en todos los casos, por lo que la figura 5.14b no es muy clara, pero si se aprecia como HGA_o es la peor y ILS_g es la metaheurística más estable en ofrecer buenos rendimientos. En la figura 5.14a podemos ver la interacción de las metaheurísticas y el número de máquinas. De nuevo genera algo de ruido el hecho de que en los tres sets de instancias existe un m 20, pero aun así en la imagen se aprecia que IG_g mejora su rendimiento respecto a los demás con muchas máquinas mientras que los algoritmos HGA_o y HGA_g empeoran su rendimiento. Por su parte el ILS_g se mantiene estable siendo siempre uno de los mejores algoritmos.

Por último, la interacción con ρ de la figura 5.16, muestra la interacción entre los diferentes algoritmos y el tiempo de ejecución de los mismos (representado por ρ) en relación con la Desviación Porcentual Relativa (RPD). En este gráfico, todos los algoritmos muestran mejoras al incrementar el tiempo de cómputo y en este caso a ritmos muy similares. Para conocer cuando estos algoritmos empiezan a estancarse en óptimos locales tendríamos que ejecutarlos mucho más tiempo.

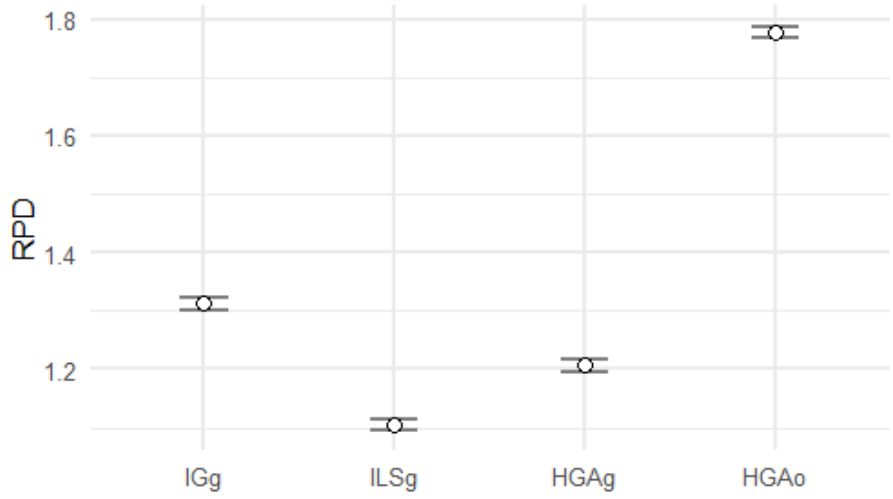
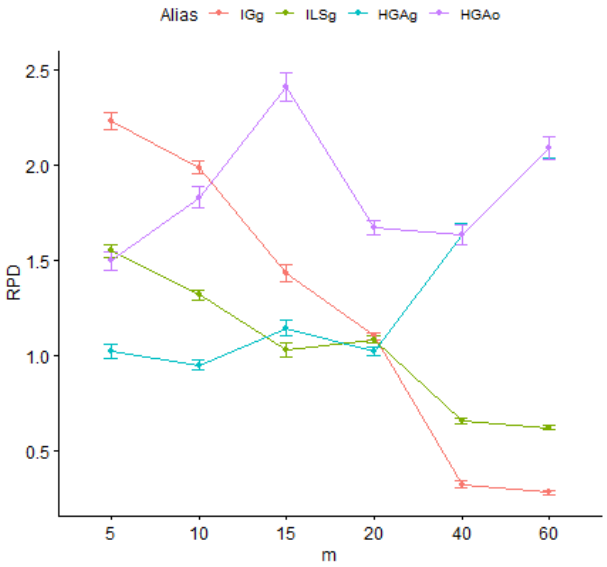
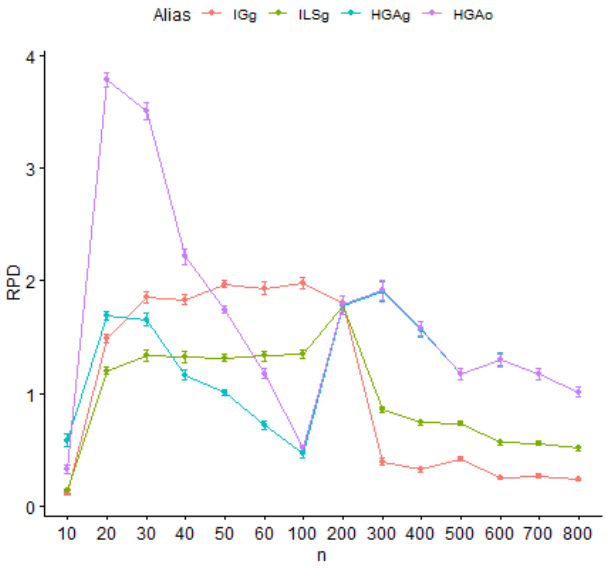


Figura 5.13: Gráfico de medias con intervalos de confianza Tukey 95% para los algoritmos en el conjunto de todas las instancias

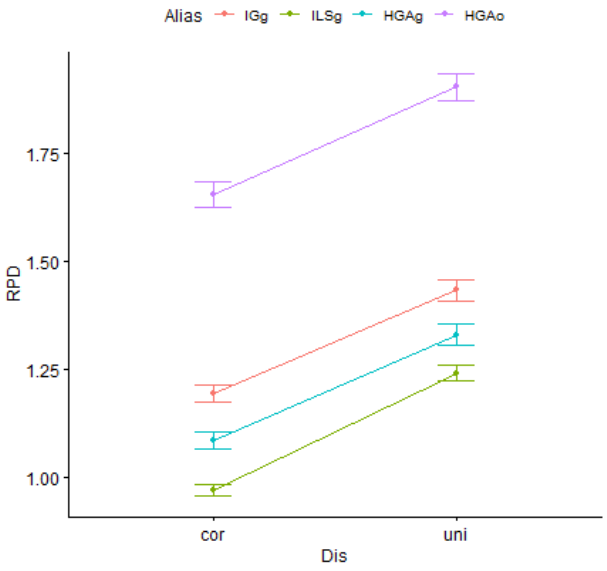


(a) Interacción entre los algoritmos y el número de máquinas (m)

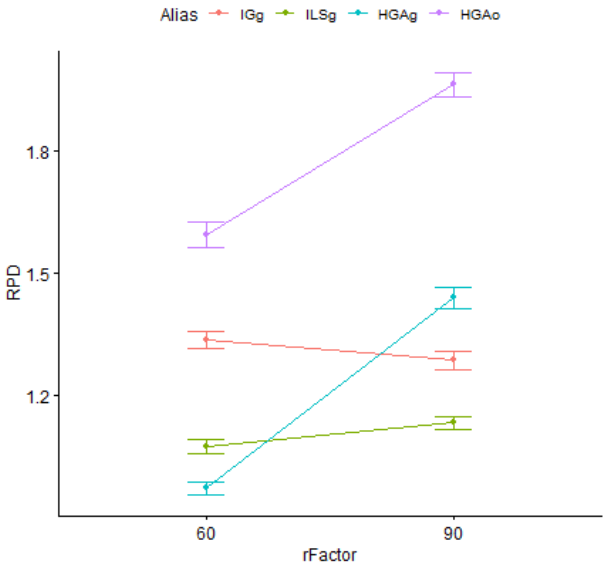


(b) Interacción entre los algoritmos y el número de trabajos (n)

Figura 5.14: Gráficos de las interacciones entre los algoritmos y algunos factores de instancia usando todas las instancias



(a) Interacción entre los algoritmos y la distribución de los recursos (*rFactor*)



(b) Interacción entre los algoritmos y el factor de disponibilidad de recursos (*rFactor*)

Figura 5.15: Gráficos de las interacciones entre los algoritmos y algunos factores de instancia usando todas las instancias

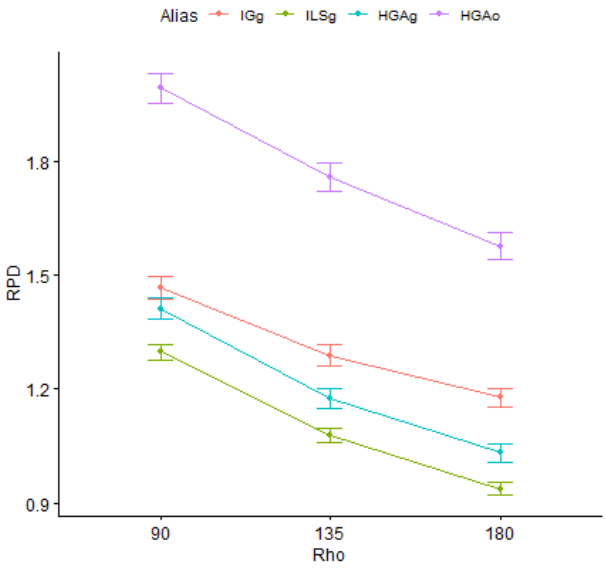


Figura 5.16: Gráfico de la interacción entre los algoritmos y ρ para el conjunto de todas las instancias

5.8 Conclusiones del capítulo

Este capítulo se plantea como la continuación en la búsqueda de métodos eficaces y eficientes para resolver el problema PFSP. Ya que en el capítulo anterior se mostraban las limitaciones de las heurísticas y modelos para resolver el problema del taller de flujo de permutación con recursos adicionales (PFSPR). Esto conduce a la conclusión lógica de que los algoritmos metaheurísticos pueden conducir a mejores soluciones.

En base a la discusión de la revisión bibliográfica planteada en el capítulo 3 se proponen tres algoritmos metaheurísticos especialmente prometedores como son: ILS, IG y HGA. La elección se basa en que estos algoritmos han sido probados con mucho éxito en problemas similares como se ha demostrado en investigaciones anteriores del área. De estos algoritmos metaheurísticos se proporciona una descripción detallada, destacando los componentes y métodos de los que están formados.

Posteriormente, se prepara la calibración de las metaheurísticas para el problema abordado.

Tras el inciso para cada algoritmo se identifican y discuten sus parámetros, su significado y el rango de valores que se exploran. Los resultados de las calibraciones se presentan, respaldados por tablas ANOVA, gráficos de medias e interacciones, para facilitar la comprensión y la evaluación. Simultáneamente, se comparan estos resultados con la calibración sugerida por los autores originales de los algoritmos.

Finalmente, tras la calibración exhaustiva de los tres algoritmos, se lleva a cabo un experimento para examinar y contrastar su rendimiento una vez ya han sido calibrados. Se explica la metodología adoptada para la experimentación, seguida por un análisis detallado de los resultados. Inicialmente este análisis se divide según el conjunto específico de instancias utilizadas, para seguidamente considerar todos los resultados juntos. El algoritmo ILS_g resulta ser el mejor algoritmo del conjunto ya que mantiene siempre unos buenos resultados independientemente del set de instancias que se utilizan o de cualquier otro factor de instancia. Por ello aparece como el mejor algoritmo cuando se analizan todas las instancias juntas. Sin embargo, los algoritmos genéticos HGA_g obtienen buenos resultados en el set de instancias de Taillard y por su pare el algoritmo IG_g obtiene buenos resultados en los sets de instancias grandes. Así que, aunque el ILS_g sea mejor por ser consistente obteniendo muy buenas soluciones, si se dispone de capacidad de cómputo, podría considerarse la alternativa de ejecutar los tres algoritmos en paralelo para obtener aún un mejor resultado.

Por lo tanto en este capítulo se ilustra tanto la metodología como los resultados de la aplicación de estas tres metaheurísticas en el contexto del PFSPR, proporcionando una base sólida para futuras investigaciones en este campo.

6. Plataforma para el desarrollo de algoritmos

Como se ha explicado a lo largo del capítulo 1 y con más detalle en la revisión bibliográfica del capítulo 3, en la actualidad del mundo empresarial e industrial se dan cambios a ritmos vertiginosos que obligan a las empresas a mantener grandes esfuerzos en automatización y sistemas de producción más eficientes, inmediatos y ágiles. En este punto es donde sistemas software especializados se utilizan para cubrir este papel, sin embargo, la optimización de procesos para problemas realistas como los planteados en esta tesis es muy compleja. Por una parte, se ha de estudiar el problema para realizar una modelización de este, por lo que además se necesita personal especializado con conocimientos técnicos muy específicos, incluso se necesita tener informatizado parte del proceso para tener acceso a los datos de producción y, por último, se ha de crear una herramienta software específica para cada problema. Todo este proceso está al alcance de grandes empresas que invierten gran cantidad de dinero en ello, pero no lo está tanto para la pequeña y mediana empresa que, además, no tiene personal especializado. Existe un problema añadido del que hemos hablado en la capítulo 1, el GAP, esa gran distancia que separa los trabajos académicos de la realidad industrial y que actualmente está recibiendo mucho interés. En este papel entran las empresas TIC (Tecnologías de la información y las comunicaciones) de investigación, que pueden trabajar de intermediarios entre los estudios científicos o académicos y la realidad industrial, contando con personal especializado y plataformas software ágiles que puedan enfrentar diferentes escenarios en diferentes empresas. Para afrontar estos problemas se necesitan paquetes software fáciles de usar, ágiles y específicos. En general estos han de ser capaces de ofrecer entornos que permitan modelar problemas de la realidad, obtener la información de estos problemas, el desarrollo de nuevos algoritmos, el uso de algoritmos ya desarrollados o parte de ellos, un sistema de prueba o test de los algoritmos y un sistema que devuelva la solución al problema.

Desde la perspectiva del investigador, enfrentarse a estos problemas también supone un gran reto y no está exento de necesidades que el software que normalmente se utiliza para investigación no suele resolver, y se lo ha de diseñar y desarrollar el propio investigador. Así, un investigador puede encontrarse con la necesidad de desarrollar todo un grupo de aplicaciones o una aplicación grande que gestione un grupo de tareas tediosas y repetitivas que siempre son necesarias. Entre estas estarían la lectura de instancias, la escritura de resultados, la lectura de los mismos, la agrupación de los resultados, la gestión de las experimentaciones que puede incluir la gestión de la ejecución de diferentes algoritmos, diferentes parámetros, etc. Como vemos la lista es larga.

6.1 FACOP

FACOP es el acrónimo de *Framework for Applied Combinatorial Optimization Problems* [1], es como su nombre indica un entorno de desarrollo para la optimización algorítmica de problemas combinatorios. Con el objetivo de facilitar el proceso de generación y testeo de algoritmos para la resolución de problemas, FACOP ofrece actualmente una suite de herramientas que puedan abstraer al usuario de todas las tareas tediosas que rodean el desarrollo de algoritmos.

El desarrollo de FACOP lo comenzó inicialmente el doctorando en un contrato predoctoral en la UPV en este trabajo contó con la dirección del grupo de sistemas de optimización aplicada (S.O.A.). Este software ha crecido mucho desde las primeras versiones que desarrollo el doctorando hasta la actualidad. Por la propia naturaleza de este *Framework* ese crecimiento supone que FACOP es ahora un gran conjunto de librerías ligeramente independientes, aunque en algunas ocasiones nos referimos también a las herramientas que utilizan FACOP. En los últimos años este software se ha utilizado para desarrollos de optimización realizados en el Instituto Tecnológico de Informática (I.T.I.) donde el doctorando trabaja a tiempo completo como desarrollador de esa misma herramienta desde hace años. Por ello, las librerías más actuales de FACOP, así como las herramientas se llevan a cabo y mejoran dentro de la empresa en el equipo LIDSOA.

FACOP es principalmente el conjunto de librerías desarrolladas en .Net, existe una librería principal conocida como FACOPLib y múltiples librerías con diferente funcionalidad. Existen librerías dedicadas a funciones específicas tan dispares como pueden ser: lectura de instancias u operadores genéticos. Bajo unos criterios que se explican en la siguiente sección, FACOP separa las funcionalidades en diferentes métodos permitiendo que se reutilice código, se eviten repeticiones innecesarias y se mantenga una organización clara.

Entre los objetivos de FACOP está la reducción del GAP de investigación. Por una parte, se centra en el investigador/desarrollador de algoritmos que obtiene una herramienta completa y testeada que simplifica y mejora el proceso de desarrollo de algoritmos de optimización en la investigación. Por otra parte, facilita la implementación y despliegue de esta librería en entornos empresariales reales. Esta librería se puede utilizar desde cualquier software que permita utilizar librerías dinámicas.

6.2 Características principales

6.2.1 Estructura FACOP e inyección de dependencias

FACOP distribuye el código en diferentes librerías, normalmente separadas por funcionalidad, así las librerías que leen ficheros están todas juntas, las librerías de algoritmos también, etc. Existe bastante libertad, pero hay recomendaciones de colocar las librerías agrupadas por su utilidad. Es la librería principal FACOPLib la que lee todas las librerías de FACOP disponibles utilizando un sistema de Factorías dinámicas que utiliza inyección de dependencias (*Dependency Injection* - DI) para generar nuevas instancias de clases. Esta capacidad que se consigue utilizando *Unity Dependency Injector* 5.11.10 permite a FACOP generar una nueva instancia de una clase desde una librería externa. En este patrón DI se utiliza *Refactoring* que permite no instanciar directamente las dependencias de una clase, aumentando la flexibilidad a la hora de inyectar dependencias. Esta capacidad es extremadamente útil, permite a FACOP utilizar librerías sin necesidad de haber sido compiladas juntas. Como se aprecia en la figura 6.1 un investigador puede incluir en una nueva librería nuevos componentes, simplemente agregarla al directorio de trabajo de FACOP y sin necesidad de hacer ningún tipo de instalación FACOP será capaz de utilizarla.

[1] Fuente: https://www.iti.es/facop-plataforma-para-la-optimizacion-de-procesos/

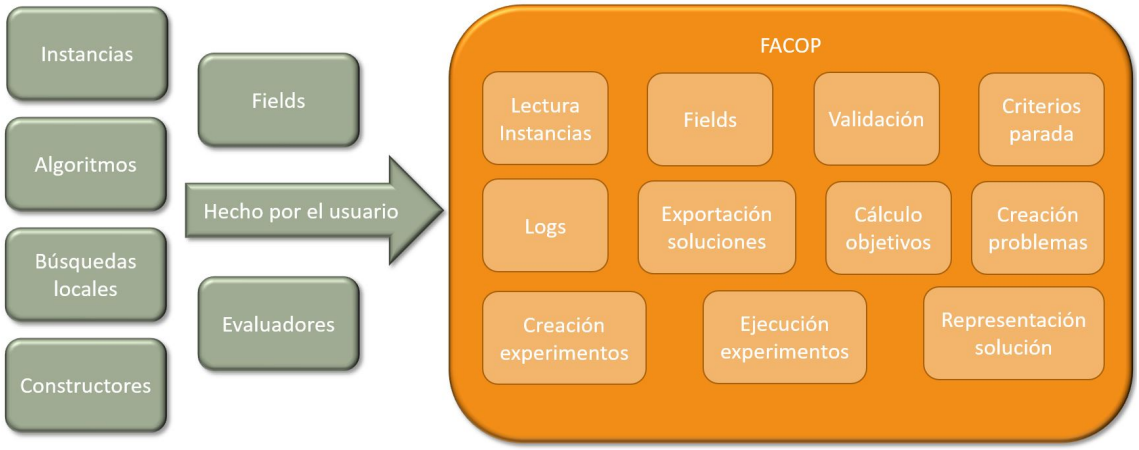


Figura 6.1: Librerías nuevas desarrolladas por el usuario (en gris) se añaden a FACOP solo copiándolas al directorio de trabajo.

6.2.2 Modularidad y reusabilidad de código

Para fomentar la reusabilidad de código, el uso controlado de los métodos, la lectura por inyección de dependencias y otras virtudes, las clases en FACOP siguen una orientación a objetos que utiliza al máximo la herencia entre clases, las clases abstractas y los interfaces, aprovechando estos últimos la herencia también. De esta forma, se han separado las funcionalidades de todo el framework para reutilizar el código.

En la revisión bibliográfica en la sección 3.4.2, se ofrece una explicación de cómo se pueden generar algoritmos por piezas con la técnica *AAD* (Automatic Algorithm Design). En ese tipo de trabajos, se intenta separar la funcionalidad interna de un algoritmo en métodos y clases que se puedan instanciar de forma independiente y mantengan esa parte de la funcionalidad. A esos métodos en algunos trabajos se les llama componentes (Alfaro-Fernández, Ruiz, Pagnozzi y Stützle 2020), y esto permite ver a un algoritmo como una plantilla que dentro tiene un conjunto de componentes. Una mejora sobre esto es considerar al algoritmo como tal también un componente, esto permite utilizar un algoritmo dentro de otro algoritmo. Esta estructura se inspira en la estructura de diseño automático de algoritmos que el doctorando desarrolló en su estancia predoctoral en IRIDIA, fruto de la cual se publicó el articulo Alfaro-Fernández, Ruiz, Pagnozzi y Stützle (2020).

Siguiendo esta práctica de trocear algoritmos y otras piezas clave en componentes, esto permite reutilizar código incluso desde algoritmos diferentes. Existen varios mecanismos para poder saber el papel que desempeña cada uno de los componentes y FACOP pueda identificar si un componente es compatible con otro o con un algoritmo. El primero es a través de la interfaz que implementa la clase, como se ve en la figura 6.2, los componentes del algoritmo son todos piezas inyectables, para saber si dos componentes son compatibles se utiliza la interfaz. También se aprecia que FACOP no contiene las clases algoritmo, constructor... estas clases siempre vienen definidas desde las librerías externas.

El segundo es validar que internamente funcionan de forma similar y son válidos para problemas similares. Así un algoritmo con una representación de la solución de una permutación no podrá conectarse con una búsqueda local usada en un algoritmo genético si este utiliza una representación de la solución que no sea una permutación o una clase descendiente de la misma.

6.3 Herramientas satélites y su propósito

La librería FACOPLib tiene mucha funcionalidad que se complementa con todas las librerías dinámicas que lee, pero es una librería, es decir, necesita de una aplicación que le llame y utilice esta funcionalidad. Se han desarrollado múltiples herramientas.

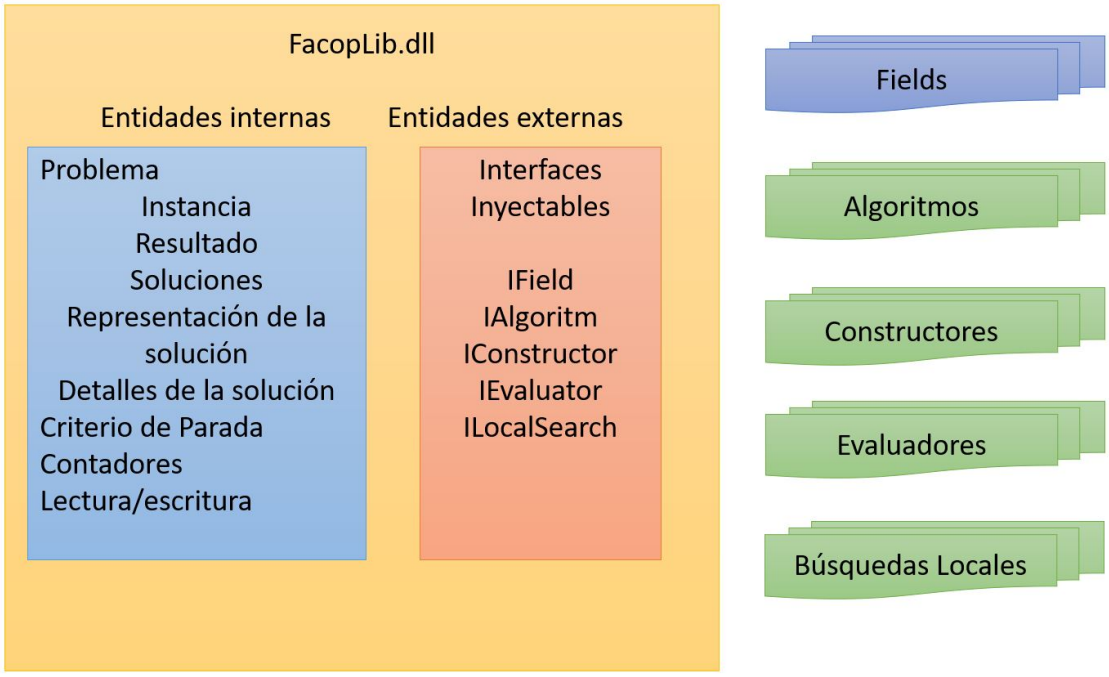


Figura 6.2: Algunas de las entidades de FACOP

6.3.1 ExecutionSolver

Es la herramienta más sencilla, su trabajo es resolver problemas utilizando FACOP, para ello necesita uno o varios ficheros de configuración. En los ficheros de configuración se especifica toda la información necesaria, como la instancia, los componentes que incluye el algoritmo, los criterios de parada, etc. Es una herramienta que se utiliza tanto para ejecuciones de un solo experimento como ejecuciones de gran cantidad de experimentos en un clúster de ordenadores.

6.3.2 Wizzard y Wizzard web

Wizzard y Wizzard Web, son asistentes paso a paso para la generación de ficheros de configuración FACOP, especialmente útiles para realizar diseños de experimentos donde se quieren utilizar diferentes valores de un parámetro en todo un conjunto de instancias. Como todo asistente guía al usuario paso a paso. Le pide al usuario las instancias y le ayuda a configurar algoritmos a partir de componentes. Los componentes elegidos son validados y se generan los ficheros de configuración de FACOP. Wizzard Web es una versión más actual que permite el uso multiplataforma desde un servidor.

6.3.3 UI

La herramienta UI es una aplicación con interfaz de usuario que permite realizar pruebas con los algoritmos, componentes o las librerías según se van desarrollando. Fields, Algoritmos, Soluciones, etc. Una función muy importante que no tienen otras herramientas es que FACOP permite descargar millones de ficheros de solución y generar las tablas que se necesitarían para cualquier estudio estadístico de un experimento. Permite también generar ficheros de configuración, resolver ficheros de configuración o crear problemas a mano y resolverlos. En la figura 6.3 se puede ver un ejemplo de uso de la interfaz de usuario de la herramienta UI. En la parte superior izquierda se encuentran las instancias a resolver, y justo debajo, los algoritmos disponibles y los parámetros a configurar. En la parte derecha se va mostrando la solución, que también se guarda en ficheros. Por último, en la parte inferior se puede observar un diagrama de Gantt para la solución del problema resuelto.

La limitación principal de esta herramienta es que no permite diseñar experimentos y que solo funciona en Windows.

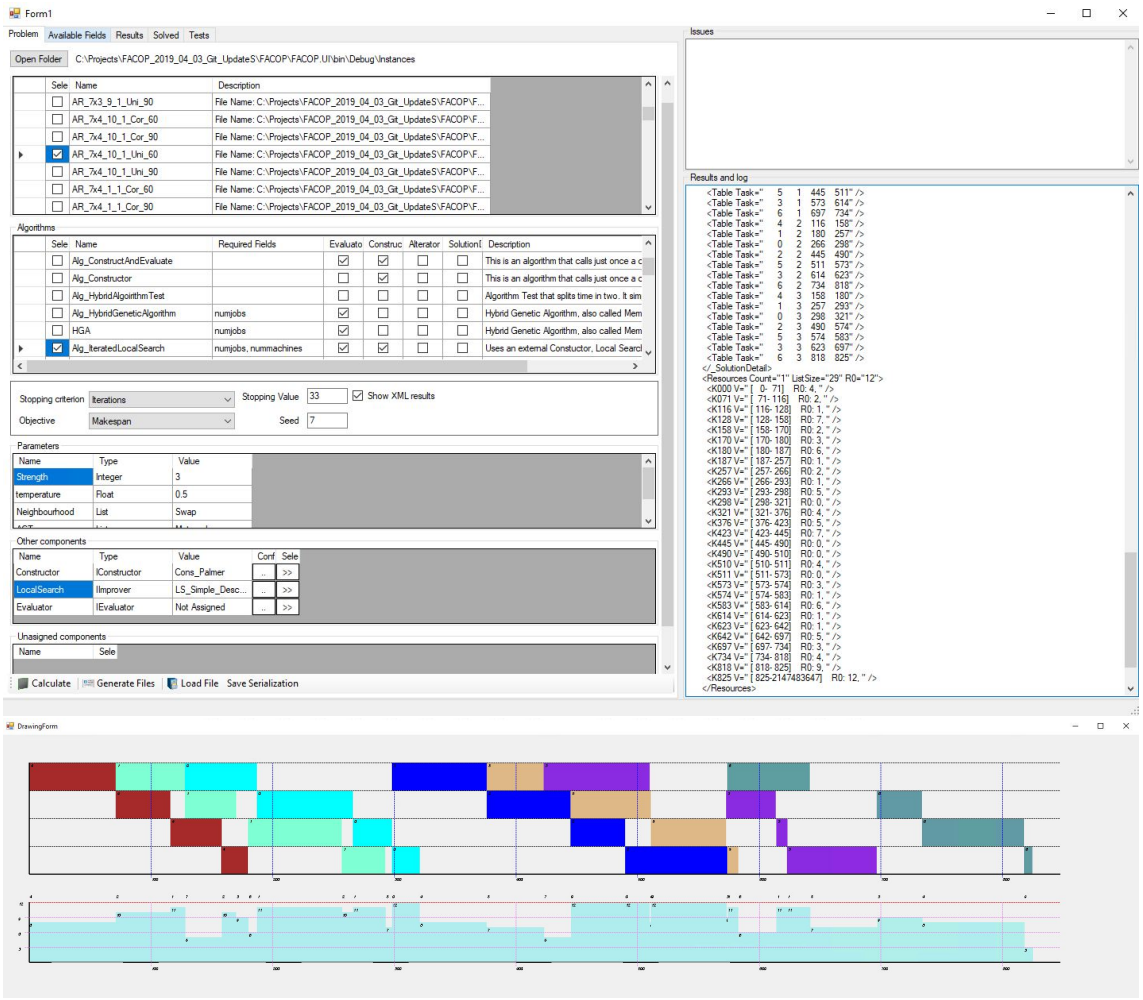


Figura 6.3: Interfaz de usuario de la herramienta UI y representación con un diagrama de Gantt de una solución.

6.3.4 Statistics

Esta herramienta permite analizar resultados de grandes experimentaciones, de forma autónoma y rápida. Dados unos resultados de una experimentación, se puede solicitar al servicio web statistics un análisis en base a una variable respuesta de uno o más factores. El sistema utiliza análisis de la varianza ANOVA para ofrecer gráficos de medias y significación estadística, seleccionando el mejor valor promedio y ofreciendo algo de información extra sobre el resultado del análisis. Cuando se solicita analizar más de un factor, z factores, realiza $z-1$ análisis de la varianza ANOVA y en cada ejecución realiza el informe y selecciona el valor con mejor promedio para el factor más significativo. Este sistema no soporta interacciones, pero es automático y no requiere supervisión.

6.4 Aplicaciones de FACOP en el mundo real

En esta sección se describen los casos de uso más importantes de la herramienta FACOP.

- El primer caso de uso que se describe, es la tesis en sí misma y todo el trabajo de investigación realizado en el estudio del problema PFSPR, así como las diferentes formas de resolverlo propuestas. Esta tesis incluye nuevos problemas y nuevos algoritmos que han

sido "troceados" en componentes siguiendo la misma política usada en Alfaro-Fernández, Ruiz, Pagnozzi y Stützle (2020), artículo sobre AAD que publicó el doctorando basado en una colaboración en el instituto IRIDIA donde aprendió y aplicó esta innovadora técnica utilizando las herramientas EMILI y IRace. Se han desarrollado he incluido en FACOP: el modelo MILP, las 3 adaptaciones de la NEH, las 24 heurísticas restantes programadas y los tres algoritmos propuestos. Se han diseñado experimentos masivos, algunos de ellos superando los 2 millones de tratamientos. Estos experimentos se han completado utilizando el generador de experimentos (Wizzard), que nos ha permitido diseñar las comparaciones de heurísticas y algoritmos, así como las calibraciones *full-factorial* de golpe. Se han realizado los experimentos en clústeres con cientos de máquinas virtuales de forma aleatorizada y automática. Se ha utilizado la aplicación que agrupa resultados (UI) para agruparlos y realizar los cálculos del RPD. Sin embargo, como el análisis estadístico que proporciona la herramienta estadística (Statistics) no soporta estudio de interacciones, se ha optado por realizar el análisis usando la herramienta "R Tools" incluida en Visual Studio (RTVS). Usando el motor "Microsoft R Open" 4.0.2, que es equivalente a R-4.0.2. Este caso de estudio demuestra la viabilidad de utilizar FACOP para investigación en problemas de optimización combinatoria.

- El segundo caso de uso que se plantea es el de un proyecto europeo, en concreto el proyecto AIDEAS, AI Driven Industrial Equipment Product Life Cycle Boosting Agility, Sustainability and Resilience en el que se está implantándose actualmente FACOP. El proyecto AIDEAS "desarrolla tecnologías de Inteligencia Artificial para ayudar durante el ciclo completo de manufactura, desde el diseño hasta el reciclado, como un instrumento estratégico para la sostenibilidad, agilidad y resiliencia en la industria de manufactura de maquinaria Europea" [2]. En este proyecto se utiliza FACOP para implantar el Procurement Optimizer, un optimizador de compras.

- El tercer caso de uso AI-PRISM, AI-Powered Human-Centred Robot Interactions for Smart Manufacturing. El tercer caso de uso que se plantea es otro proyecto Europeo en el que se está implantándose actualmente FACOP. En AI-PRISM se quiere "proveer de un ecosistema de colaboración y cooperación entre humanos y robots en el entorno de manufactura donde las tareas sean difíciles de automatizar y la velocidad y versatilidad sean esenciales" [3]. En este proyecto FACOP, ayuda en los procesos de scheduling para además permitir la re-utilización de componentes.

- Otros casos de uso: se ha trabajado en resolver problemas de contenedores en el puerto de Valencia, se ha trabajado optimizando la colocación de cajas en palés, se ha utilizado para la optimización de rutas de autobuses, etc.

Como se puede ver, este software ya está consolidado como una herramienta factible tanto para implantaciones en empresa, como para estudios académicos. Por estas razones podemos decir que FACOP ha colaborado a la reducción de la brecha o GAP entre la ciencia y la industria.

[2]Fuente: https://aideas-project.eu/
[3]Fuente: https://aiprism.eu/

7. Conclusiones, aportaciones y líneas futuras

Esta Tesis Doctoral aborda el problema del taller de flujo de permutación considerando recursos adicionales renovables. Este problema es una versión más realista del clásico problema de taller de flujo de permutación. Incluir recursos, además de las máquinas, tales como personal, herramientas, moldes... incorpora al problema una realidad muy habitual en al ámbito de la producción industrial y que es todo un reto desde el punto de vista académico. Diseñar e implementar herramientas que ayuden a tomar decisiones en un problema de producción que incorpora características reales, permite disminuir el GAP existente entre la academia y el entorno productivo y facilita la transferencia de investigación.

Tras introducir, en el capítulo 1, la importancia de los problemas de producción, así como la motivación y objetivos de este trabajo, el capítulo 2 muestra una clasificación y caracterización de los diferentes problemas de scheduling, que se han abordado en el ámbito de la producción, teniendo en cuenta los diferentes entornos productivos existentes. Para finalizar se caracteriza el problema objeto de esta tesis: el taller de flujo con permutación y recursos adicionales renovables con el objetivo de minimizar el makespan (PFSPR).

El capítulo 3 recoge una amplia revisión bibliográfica. Se han analizado trabajos no solo relacionados directamente con el problema del taller de flujo, sino también problemas de scheduling que incorporaban recursos renovables adicionales. El resultado de esta revisión es que no se han encontrado, hasta la fecha, artículos que hayan abordado el problema que se estudia en este trabajo. Por ello, la aportación principal de esta Tesis Doctoral es el estudio por primera vez de este problema y la propuesta y adaptación de métodos para su resolución.

El capítulo 4 muestra la definición formal del problema PFSPR y su modelización matemática a través de un modelo de programación lineal mixta (MILP). Los problemas de scheduling de entornos productivos que han trabajado con recursos mostraban, tal y como se ha recogido en la revisión bibliográfica, la dificultad que los modelos de programación matemática tenían a la hora de abordar instancias que no fuesen muy pequeñas. El modelo propuesto se ha probado en conjunto de instancias generadas exprofeso con la finalidad de analizar su escalabilidad. El modelo muestra la dificultad del problema ya que es incapaz de resolver óptimamente instancias de más de 7 trabajos y 3 máquinas. Esto justifica la necesidad de diseñar e implementar métodos aproximados (heurísticas y metaheurísticas) capaces de proporcionar buenas soluciones en un tiempo pequeño de computación para problemas más realistas, es decir, con mayor cantidad de trabajos y máquinas.

Para analizar el rendimiento de las heurísticas y metaheurísticas implementadas, se han propuesto adaptaciones de las instancias más populares para el problema del taller de flujo con permutación procedentes de Taillard (1993) y Vallada, Ruiz y J. M. Framinan (2015) ya que no

incorporaban recursos adicionales. Para introducir la información de los recursos se ha utilizado la misma metodología que Fanjul-Peyro, Perea y Ruiz (2017) para UPMR. Partiendo de estas instancias se han podido realizar múltiples experimentaciones para poder validar los diferentes métodos propuestos y analizar las diferencias en eficacia y eficiencia de estos métodos.

A diferencia de la evaluación del problema PFSP, la evaluación cuando se disponen de recursos adicionales ha de cambiar, una de las razones es que cuando se está evaluando una solución y la necesidad de recursos no se puede cumplir, algunas tareas tendrán que esperar. Esto nos condujo a realizar pruebas que determinó que si se secuencia un trabajo completo detrás de otro trabajo completo (secuenciando por trabajo), los bloqueos por recursos aparecen espaciados entre los trabajos, lo que de alguna forma mantiene el uso de recursos más constante. En contraposición, la secuenciación por máquina empeora los resultados porque no se valoran los recursos consumidos en el *schedule* cuando se secuencian las tareas en las primeras máquinas. Esto genera mayores retrasos al secuenciar las tareas de las últimas máquinas, por todo ello, es apoya la tesis de la necesidad de considerar los recursos en los problemas de scheduling.

Con el objetivo de disponer de reglas o heurísticas que sean rápidas en obtener una solución factible, se han adaptado heurísticas, que proporcionaban buenos resultados en la versión del problema sin recursos, al problema con recursos adicionales y se han creado nuevas. A estos métodos se les ha añadido políticas de desempate para intentar mejorarlos y, finalmente, todos estos métodos se han comparado entre sí. A través del análisis de la experimentación se concluye que la adaptación del algoritmo de Palmer es la mejor de estas heurísticas. Sin embargo, se constata que todas las heurísticas son capaces de encontrar la mejor solución conocida del grupo para algunas instancias, esto junto a la rapidez de cálculo invita a pensar que son complementarias. Se han propuesto algoritmos multipasada con agrupaciones de heurísticas con el objetivo de mejorar la eficacia sin empobrecer mucho la eficiencia. Estos experimentos también ofrecieron la conclusión que se pueden obtener un algoritmo multipasada como B10, que utiliza todas las reglas con sus mejores políticas de desempate, que ofrece un incremento en la eficiencia manteniendo unos tiempos de cómputo moderados.

Además de implementar las heurísticas basadas en reglas de despacho, se adapta la clásica NEH al problema de PFSPR diseñando tres versiones, tras analizar cómo afectaba el hecho de los recursos a la clásica NEH. Como estas adaptaciones de la NEH utilizan una ordenación generando una lista de trabajos pendientes, se han probado todas las heurísticas anteriormente propuestas para comprobar cuál es la mejor ordenación para estos métodos. Así en una extensa experimentación con estas 3 adaptaciones combinadas con 24 posibles ordenaciones se ha concluido que la NEH *cut and fill* es la más eficaz, pero no puede terminar cuando las instancias son muy grandes. La versión NEH con reparación aunque no es tan eficiente que la NEH *cut and fill* es mucho más rápida.

La adaptación NEH abortion es la que peores resultados proporciona. Por tanto, la NEH cut and fill y la NEH con reparación son métodos más costosos que las heurísticas basadas en reglas de despacho pero proporcionan mejores resultados. Además, puedes utilizarse como soluciones de partida en metaheurísitcas más sofisticadas.

Con el objetivo de incrementar la calidad de las soluciones proporcionadas por las heurísticas, el capítulo 5 muestra las tres metaheurísticas que se han implementado para este problema. Las metaheurísticas seleccionadas son las que han resultado ser las más eficientes en problemas de taller de flujo con permutación. Se han implementado dos metaheurísticas basadas en trayectorias: el Iterated Local Search (ILS) e Iterated Greedy (IG) y otra basada en poblaciones: el Hybrid Genetic Algorithm (HGA).

Una vez calibradas con el set que incluye todas las instancias de entrenamiento, se prepara una exhaustiva experimentación para comparar el rendimiento de las mismas. Se compararán las tres metaheurísticas con los valores de nuestra calibración ILS_g, IG_g y HGA_g. Pero también se incluye la versión HGA_o que utiliza los valores de la calibración original ya que la calibración original se hizo por iteraciones y resulta en valores bastante diferentes, por consistencia se añade HGA_o que

utiliza la calibración original.

Con estos 4 algoritmos se realiza una extensa experimentación para compararlos. La experimentación se estudia utilizando análisis de la varianza (ANOVA). Primero se estudian los datos separados por cada uno de los sets de instancias (Taillard, VRFS y VRFL) y finalmente se analizan como conjunto completo.

Aunque dependiendo del set de instancias el ILS o el HGA parecen funcionar en algunos casos uno mejor que otro, en conjunto el ILS parece ser una metaheurística eficiente a la hora de abordar cualquier tamaño de problema. El estudio muestra que la calidad de las soluciones proporcionadas por las metaheurísticas incrementa cuando ésta tiene un mayor tiempo de computación. Señalar que se ha aplicado el análisis de la varianza (ANOVA) para analizar si los resultados son estadísticamente significativos y se ha realizado el correspondiente análisis de residuos para comprobar si las hipótesis del ANOVA se cumplían en todo momento.

Con este trabajo original se ofrece la metodología y resultados de aplicar estas metaheurísticas adaptadas al problema PFSPR, proporcionando información y resultados que esperamos sean útiles para futuras investigaciones en el ámbito de scheduling para problemas con recursos adicionales.

El trabajo requerido para resolver problemas similares al presentado en esta tesis, tanto en el entorno académico como en el entorno empresarial, se requiere un enorme esfuerzo que se divide en múltiples tareas. Para ayudar en algunas de estas tareas se ha creado el entorno de desarrollo para algoritmos FACOP que se presenta en esta tesis. Esta librería puede ayudar a lo largo de todo este proceso de múltiples formas, asistiendo o automatizando múltiples tareas. En la entrada de datos, para los casos en los que las experimentaciones se realizan sobre multitud de instancias de un problema ayuda ofreciendo un sistema estandarizado y automatizado de lecturas de instancias, pero también se ha utilizado para la lectura directa de datos de cliente desde otras fuentes. Facilita la reutilización de código entre diferentes algoritmos, incluso diferentes problemas. Acceso a librerías conjuntas con representaciones de la solución, soluciones, resultados y muchas otras partes del problema completas y testadas. Facilita la generación de ficheros de configuración para experimentos masivos de comparación o calibración de algoritmos. Obtener sumarios de resultados en formato de tabla, análisis sencillos de los mismos, etc. Los trabajos que se presentan en esta tesis para abordar el problema PFSPR con un enfoque investigador constatan la validez de este software como herramienta para la investigación y al mismo tiempo los casos reales de implantación demuestran la validez del mismo como herramienta de transferencia científica.

7.0.1 Otras aportaciones

Además de las aportaciones evidentes de la tesis como son el propio trabajo presentado, así como la herramienta FACOP, ha habido más aportaciones que se han generado a lo largo de esta tesis. En sus comienzos el doctorando colaboró en la experimentación y los estudios estadísticos en el artículo:

- Q.-K. Pan, R. Ruiz y P. Alfaro-Fernández (2017). “Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows”. En: *Computers & Operations Research* 80, páginas 50-60. ISSN: 03050548

Además, el doctorando realizó una estancia pre-doctoral en el instituto IRIDIA (*Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*) en la "Université Libre de Bruxelles"(ULB) en Bruselas. Un instituto puntero en aplicaciones novedosas de la Inteligencia Artificial. Durante esta estancia trabajo con tecnologías como IRace y EMILI para desarrollar un sistema de diseño automático de algoritmos, con sus colaboradores diseño el sistema para enfrentar el problema del taller de flujo híbrido, y estudio los diferentes funcionamientos y algoritmos que se generaban al enfrentar diferentes versiones mono-objetivo del problema. Como resultado de este trabajo el doctorado presentó el trabajo en dos congresos y publicó un artículo:

- P. Alfaro-Fernández, R. Ruiz, F. Pagnozzi y T. Stützle (2020). "Automatic algorithm design for hybrid flowshop scheduling problems". En: *European Journal of Operational Research* 282.3, páginas 835-845
- P. Alfaro-Fernández, R. Ruiz, F. Pagnozzi y T. Stützle (2017b). "Exploring automatic algorithm design for the hybrid flowshop problem". En: *12th Metaheuristics International Conference*, páginas 201-203
- P. Alfaro-Fernández, R. Ruiz, F. Pagnozzi y T. Stützle (2017a). "Approaching Scheduling with Automatic Algorithm Design". En: *21st Triennial Conference of the International Federation of Operational Research Societies (IFORS 2017)*

El trabajo desarrollado en IRIDIA ha sido clave para mejorar el planteamiento de FACOP y hacerlo capaz de hacer cosas similares a menor escala. La parte inicial del trabajo de esta tesis ya se ha presentado en el siguiente congreso:

- P. Alfaro-Fernández, R. Ruiz, F. Pagnozzi y T. Stützle (2018). "Flow shop scheduling problem with additional resources". En: *29th European Conference on Operational Research (EURO 2018)*, páginas 201-203. ISBN: 978-84-09-02938-9

Por último, señalar que actualmente se está elaborando un artículo en una revista JCR que recoge las contribuciones de esta tesis.

7.0.2 Líneas futuras

Este trabajo puede ser extendido en múltiples direcciones para ampliar el grueso de la investigación científica. Una posible mejora sería el tratamiento de problemas multi-objetivo, ya que en muchas industrias la limitación de recursos renovables es una cota superior, por ejemplo, la energía eléctrica contratada que permite más máquinas trabajar simultáneamente, pero a un coste mayor y por tanto obtener la frontera de Pareto. Esto podría ayudar en la toma de decisiones y la rentabilidad económica de inversiones.

El trabajo actual es sobre el objetivo más estudiado (*makespan*), en muchos casos no es un objetivo factible para empresas reales que tienen fechas de llegada de materias primas y fechas de entrega prefijadas. Este tipo de restricciones no son tan estudiadas, pero son extremadamente comunes en la industria. Con la inclusión de fechas de liberación, pero aplicada a algunos recursos y otros no, dando lugar a un problema con recursos adicionales doblemente restrictivos como podrían ser trabajadores y materias primas al mismo tiempo. Siendo así un problema mucho más realista. Otra línea de investigación aún más realista sería plantear el problema del taller de flujo sin permutación, es decir, en el proceso productivo se pueden adelantar tareas rompiendo la permutación y permitiendo utilizar de forma más exhaustiva los recursos disponibles.

Finalmente, se abre todo un abanico de posibilidades en la mejora del software FACOP. Actualmente ya se está preparando un sistema de asistencia a compras optimizadas. Pero se podría finalizar las piezas que faltan para la implementación automatizada del proceso de diseño automático de algoritmos, reparar el sistema para uso multi-objetivo y muchas otras mejoras.

Bibliografía

Alfaro-Fernández, P., R. Ruiz, F. Pagnozzi y T. Stützle (2017a). "Approaching Scheduling with Automatic Algorithm Design". En: *21st Triennial Conference of the International Federation of Operational Research Societies (IFORS 2017)*.

Alfaro-Fernández, P., R. Ruiz, F. Pagnozzi y T. Stützle (2017b). "Exploring automatic algorithm design for the hybrid flowshop problem". En: *12th Metaheuristics International Conference*, páginas 201-203.

Alfaro-Fernández, P., R. Ruiz, F. Pagnozzi y T. Stützle (2018). "Flow shop scheduling problem with additional resources". En: *29th European Conference on Operational Research (EURO 2018)*, páginas 201-203. ISBN: 978-84-09-02938-9.

Alfaro-Fernández, P., R. Ruiz, F. Pagnozzi y T. Stützle (2020). "Automatic algorithm design for hybrid flowshop scheduling problems". En: *European Journal of Operational Research* 282.3, páginas 835-845.

Aranha, C., C. L. Camacho Villalón, F. Campelo, M. Dorigo, R. Ruiz, M. Sevaux, K. Sörensen y T. Stützle (2022). "Metaphor-based metaheuristics, a call for action: the elephant in the room". En: *Swarm Intelligence* 16.1, páginas 1-6.

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. John Wiley & Sons.

Bektaş, T., A. Hamzadayı y R. Ruiz (2020). "Benders decomposition for the mixed no-idle permutation flowshop scheduling problem". En: *Journal of Scheduling* 23, páginas 513-523.

Blazewicz, J., N. Brauner y G. Finke (2004). "Scheduling with Discrete Resource Constraints." En.

Blazewicz, J., K. Ecker, E. Pesch, G. Schmidt y J. Weglarz (2019). *Handbook on scheduling*. Springer.

Błażewicz, J., W. Kubiak, H. Röck y J. Szwarcfiter (sep. de 1987). "Minimizing mean flow-time with parallel processors and resource constraints". En: *Acta Informatica* 24.5, páginas 513-524. ISSN: 1432-0525.

Błażewicz, J., J. Lenstra y A. Kan (1983). "Scheduling subject to resource constraints: classification and complexity". En: *Discrete Applied Mathematics* 5.1, páginas 11-24. ISSN: 0166-218X.

Bonney, M. y S. Gundry (1976). "Solutions to the constrained flowshop sequencing problem". En: *Journal of the Operational Research Society* 27.4, páginas 869-883.

Brum, A., R. Ruiz y M. Ritt (2022). "Automatic generation of iterated greedy algorithms for the non-permutation flow shop scheduling problem with total completion time minimization". En: *Computers & Industrial Engineering* 163, página 107843. ISSN: 0360-8352.

Campbell, H. G., R. A. Dudek y M. L. Smith (1970). "A heuristic algorithm for the n job, m machine sequencing problem". En: *Management science* 16.10, B-630.

Chen, C.-L., V. S. Vempati y N. Aljaber (1995). "An application of genetic algorithms for flow shop problems". En: *European Journal of Operational Research* 80.2, páginas 389-396.

Ciavotta, M., G. Minella y R. Ruiz (2013). "Multi-objective sequence dependent setup times permutation flowshop: A new algorithm and a comprehensive study". En: *European Journal of Operational Research* 227.2, páginas 301-313. ISSN: 0377-2217.

Daniels, R. L., B. J. Hoopes y J. B. Mazzola (1996). "Scheduling parallel manufacturing cells with resource flexibility". En: *Management science* 42.9, páginas 1260-1276.

Daniels, R. L. y J. B. Mazzola (1993). "A tabu-search heuristic for the flexible-resource flow shop scheduling problem". En: *Annals of Operations Research* 41.3, páginas 207-230.

Daniels, R. L. y J. B. Mazzola (1994). "Flow shop scheduling with resource flexibility". En: *Operations Research* 42.3, páginas 504-522.

Dong, X., P. Chen, H. Huang y M. Nowak (2013). "A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time". En: *Computers & Operations Research* 40.2, páginas 627-632.

Dudek, R. A. y O. F. Teuton Jr (1964). "Development of m-stage decision rule for scheduling n jobs through m machines". En: *Operations Research* 12.3, páginas 471-497.

Edis, E. B. y C. Oguz (2012). "Parallel machine scheduling with flexible resources". En: *Computers & Industrial Engineering* 63.2, páginas 433-447. ISSN: 0360-8352.

Edis, E. B., C. Oguz e I. Ozkarahan (2013). "Parallel machine scheduling with additional resources: Notation, classification, models and solution methods". En: *European Journal of Operational Research* 230.3, páginas 449-463. ISSN: 0377-2217.

Edis, E. B. e I. Ozkarahan (2011). "A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions". En: *Engineering Optimization* 43.2, páginas 135-157.

Edis, E. B. e I. Ozkarahan (feb. de 2012). "Solution approaches for a real-life resource-constrained parallel machine scheduling problem". En: *The International Journal of Advanced Manufacturing Technology* 58.9, páginas 1141-1153. ISSN: 1433-3015.

Fanjul-Peyro, L. (2020). "Models and an exact method for the Unrelated Parallel Machine scheduling problem with setups and resources". En: *Expert Systems with Applications: X* 5, página 100022. ISSN: 2590-1885.

Fanjul-Peyro, L., F. Perea y R. Ruiz (2017). "Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources". En: *European Journal of Operational Research* 260.2, páginas 482-493. ISSN: 0377-2217.

Fanjul-Peyro, L. y R. Ruiz (2010). "Iterated greedy local search methods for unrelated parallel machine scheduling". En: *European Journal of Operational Research* 207.1, páginas 55-69. ISSN: 0377-2217.

Fernandez-Viagas, V., P. Perez-Gonzalez y J. M. Framinan (2019). "Efficiency of the solution representations for the hybrid flow shop scheduling problem with makespan objective". En: *Computers & Operations Research* 109, páginas 77-88.

Ferone, D., S. Hatami, E. M. González-Neira, A. A. Juan y P. Festa (2020). "A biased-randomized iterated local search for the distributed assembly permutation flow-shop problem". En: *International Transactions in Operational Research* 27.3, páginas 1368-1391.

Figielska, Ewa (2008). "A new heuristic for scheduling the two-stage flowshop with additional resources". En: *Computers & Industrial Engineering* 54.4, páginas 750-763.

Figielska, Ewa (2009). "A genetic algorithm and a simulated annealing algorithm combined with column generation technique for solving the problem of scheduling in the hybrid flowshop with additional resources". En: *Computers & Industrial Engineering* 56.1, páginas 142-151.

Figielska, Ewa (2010). "Heuristic algorithms for preemptive scheduling in a two-stage hybrid flowshop with additional renewable resources at each stage". En: *Computers & Industrial Engineering* 59.4, páginas 509-519.

Figielska, Ewa (2014). "A heuristic for scheduling in a two-stage hybrid flowshop with renewable resources shared among the stages". En: *European Journal of Operational Research* 236.2, páginas 433-444.

Figielska, Ewa (2018). "Scheduling in a two-stage flowshop with parallel unrelated machines at each stage and shared resources". En: *Computers & Industrial Engineering* 126, páginas 435-450.

Framinan, J., R. Leisten y C. Rajendran (2003). "Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem". En: *International Journal of Production Research* 41.1, páginas 121-148.

Framinan, J. M., R. Leisten y R. Ruiz García (2014). *Manufacturing scheduling systems: an integrated view on models, methods and tools*. Springer Science & Business Media.

French, S. (1982). "An Introduction to the Mathematics of the Job Shop". En: *Ellis Horwood Ltd*.

Garey, M. R. y D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*.

Garey, M. R., D. S. Johnson y R. Sethi (1976). "The complexity of flowshop and jobshop scheduling". En: *Mathematics of operations research* 1.2, páginas 117-129.

Glover, F. (1986). "Future paths for integer programming and links to artificial intelligence". En: *Computers & Operations Research* 13.5. Applications of Integer Programming, páginas 533-549. ISSN: 0305-0548.

Gmys, J., M. Mezmaz, N. Melab y D. Tuyttens (2020). "A computationally efficient Branch-and-Bound algorithm for the permutation flow-shop scheduling problem". En: *European Journal of Operational Research* 284.3, páginas 814-833. ISSN: 0377-2217.

Gonzalez-Neira, E. M., D. Ferone, S. Hatami y A. A. Juan (2017). "A biased-randomized simheuristic for the distributed assembly permutation flowshop problem with stochastic processing times". En: *Simulation Modelling Practice and Theory* 79, páginas 23-36.

Graham, R., E. Lawler, J. Lenstra y A. Kan (1979). "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey". En: *Annals of Discrete Mathematics* 5, páginas 287-326. ISSN: 0167-5060.

Graves, S. C. (1981). "A review of production scheduling". En: *Operations research* 29.4, páginas 646-675.

Gupta, J. N. (1971). "A functional heuristic algorithm for the flowshop scheduling problem". En: *Journal of the Operational Research Society* 22, páginas 39-47.

Holland, J. H. (1975). "Adaptation in Natural and Artificial Systems". En: *Ann Arbor*.

Ignall, E. y L. Schrage (1965). "Application of the branch and bound technique to some flow-shop scheduling problems". En: *Operations research* 13.3, páginas 400-412.

Janiak, A. (1988). "General flow-shop scheduling with resource constraints". En: *International Journal of Production Research* 26.6, páginas 1089-1103.

Janiak, A., W. Janiak y M. Lichtenstein (2007). "Resource management in machine scheduling problems: a survey". En: *Decision Making in Manufacturing and Services* 1.1-2, páginas 59-89.

Johnson, S. (1954). "Optimal Two- and Three-Stage Production Schedules with Setup Times Included". En: *Naval research logistics quarterly* 1.1, páginas 61-68.

Kalczynski, P. J. y J. Kamburowski (2008). "An improved NEH heuristic to minimize makespan in permutation flow shops". En: *Computers & Operations Research* 35.9, páginas 3001-3008.

Katragjini, K., E. Vallada y R. Ruiz (2013). "Flow shop rescheduling under different types of disruption". En: *International Journal of Production Research* 51.3, páginas 780-797.

Kellerer, H. y V. A. Strusevich (2008). "Scheduling parallel dedicated machines with the speeding-up resource". En: *Naval Research Logistics (NRL)* 55.5, páginas 377-389.

Laribi, I., F. Yalaoui y Z. Sari (2019). "Permutation flow shop scheduling problem under non-renewable resources constraints". En: *International Journal of Mathematical Modelling and Numerical Optimisation* 9.3, páginas 254-286.

Leu, S.-S. y S.-T. Hwang (2002). "GA-based resource-constrained flow-shop scheduling model for mixed precast production". En: *Automation in construction* 11.4, páginas 439-452.

Liang, Z., P. Zhong, M. Liu, C. Zhang y Z. Zhang (2022). "A computational efficient optimization of flow shop scheduling problems". En: *Scientific Reports* 12.1, página 845.

Lomnicki, Z. (1965). "A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem". En: *Journal of the operational research society* 16, páginas 89-100.

Lourenço, H. R., O. C. Martin y T. Stützle (2019). "Iterated local search: Framework and applications". En: *Handbook of metaheuristics*. Springer, páginas 129-168.

Manne, A. S. (1960). "On the job-shop scheduling problem". En: *Operations research* 8.2, páginas 219-223.

Minella, G., R. Ruiz y M. Ciavotta (2011). "Restarted Iterated Pareto Greedy algorithm for multi-objective flowshop scheduling problems". En: *Computers & Operations Research* 38.11, páginas 1521-1533. ISSN: 0305-0548.

Miyata, H. H. y M. S. Nagano (2022). "An iterated greedy algorithm for distributed blocking flow shop with setup times and maintenance operations to minimize makespan". En: *Computers & Industrial Engineering* 171, página 108366.

Montgomery, D. C. (2012). *Design and analysis of experiments*. Eigth. John Wiley & Sons.

Murata, T., H. Ishibuchi y H. Tanaka (1996a). "Genetic algorithms for flowshop scheduling problems". En: *Computers & Industrial Engineering* 30.4, páginas 1061-1071.

Murata, T., H. Ishibuchi y H. Tanaka (1996b). "Multi-objective genetic algorithm and its applications to flowshop scheduling". En: *Computers & industrial engineering* 30.4, páginas 957-968.

Naderi, B. y R. Ruiz (2010). "The distributed permutation flowshop scheduling problem". En: *Computers & Operations Research* 37.4, páginas 754-768. ISSN: 0305-0548.

Naderi, B., R. Ruiz y V. Roshanaei (mar. de 2023). "Mixed-Integer Programming vs. Constraint Programming for Shop Scheduling Problems: New Results and Outlook". En: *INFORMS Journal on Computing*.

Nawaz, M., E. E. Enscore Jr e I. Ham (1983). "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem". En: *Omega* 11.1, páginas 91-95.

Nowicki, E. y C. Smutnicki (1996a). "A fast taboo search algorithm for the job shop problem". En: *Management science* 42.6, páginas 797-813.

Nowicki, E. y C. Smutnicki (1998). "The flow shop with parallel machines: A tabu search approach". En: *European Journal of Operational Research* 106.2-3, páginas 226-253.

Nowicki, E. y C. Smutnicki (1996b). "A fast tabu search algorithm for the permutation flow-shop problem". En: *European Journal of Operational Research* 91.1, páginas 160-175.

Onwubolu, G. y D. Davendra (2006). "Scheduling flow shops using differential evolution algorithm". En: *european Journal of Operational research* 171.2, páginas 674-692.

Osman, I. H. y C. Potts (1989). "Simulated annealing for permutation flow-shop scheduling". En: *Omega* 17.6, páginas 551-557.

Page, E. (1961). "An approach to the scheduling of jobs on machines". En: *Journal of the Royal Statistical Society: Series B (Methodological)* 23.2, páginas 484-492.

Pagnozzi, F. y T. Stützle (2019). "Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems". En: *European Journal of Operational Research* 276.2, páginas 409-421. ISSN: 0377-2217.

Palmer, D. (1965). "Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum". En: *Journal of the Operational Research Society* 16.1, páginas 101-107.

Pan, C.-H. (1997). "A study of integer programming formulations for scheduling problems". En: *International Journal of Systems Science* 28.1, páginas 33-41.

Pan, Q.-K., R. Ruiz y P. Alfaro-Fernández (2017). "Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows". En: *Computers & Operations Research* 80, páginas 50-60. ISSN: 03050548.

Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*. Fifth. Springer.

Pochet, Y. y L. A. Wolsey (2006). *Production planning by mixed integer programming*. Volumen 149. 2. Springer.

Rajendran, C. y H. Ziegler (2004). "Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs". En: *European Journal of Operational Research* 155.2, páginas 426-438.

Reeves, C. R. (1993). "Improving the efficiency of tabu search for machine sequencing problems". En: *Journal of the operational research society* 44.4, páginas 375-382.

Reeves, C. R. (1995). "A genetic algorithm for flowshop sequencing". En: *Computers & operations research* 22.1, páginas 5-13.

Reeves, C. R. y T. Yamada (1998). "Genetic algorithms, path relinking, and the flowshop sequencing problem". En: *Evolutionary computation* 6.1, páginas 45-60.

Reza Hejazi, S. y S. Saghafian (2005). "Flowshop-scheduling problems with makespan criterion: a review". En: *International Journal of Production Research* 43.14, páginas 2895-2929.

Ribas, I., R. Companys y X. Tort-Martorell (2011). "An iterated greedy algorithm for the flowshop scheduling problem with blocking". En: *Omega* 39.3, páginas 293-301. ISSN: 0305-0483.

Ribas, I., R. Leisten y J. M. Framiñan (ago. de 2010). "Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective". En: *Computers & Operations Research* 37.8, páginas 1439-1454. ISSN: 0305-0548.

Rinnooy Kan, A. H. (1976). "Machine scheduling problems: classification, complexity, and computations". PhD thesis. University of Amsterdam.

Rios-Mercado, R. Z. y J. F. Bard (1999). "A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times". En: *IIE transactions* 31.8, páginas 721-731.

Ronconi, D. P. y V. A. Armentano (2001). "Lower bounding schemes for flowshops with blocking in-process". En: *Journal of the Operational Research Society* 52.11, páginas 1289-1297.

Rossit, D. A., F. Tohmé y M. Frutos (2018). "The non-permutation flow-shop scheduling problem: a literature review". En: *Omega* 77, páginas 143-153.

Ruiz, R., C. Maroto y J. Alcaraz (2006). "Two new robust genetic algorithms for the flowshop scheduling problem". En: *Omega* 34.5, páginas 461-476.

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

Ruiz, R. y T. Stützle (2007). "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem". En: *European Journal of Operational Research* 177.3, páginas 2033-2049. ISSN: 0377-2217.

Ruiz, R. y T. Stützle (2008). "An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives". En: *European Journal of Operational Research* 187.3, páginas 1143-1159. ISSN: 0377-2217.

Ruiz, R. y J. A. Vázquez-Rodríguez (ago. de 2010). "The hybrid flow shop scheduling problem". En: *European Journal of Operational Research* 205.1, páginas 1-18. ISSN: 0377-2217.

Ruiz-Torres, A. J. y G. Centeno (2008). "Minimizing the number of late jobs for the permutation flowshop problem with secondary resources". En: *Computers & Operations Research* 35.4, páginas 1227-1249. ISSN: 0305-0548.

Sadjadi, S., M. Aryanezhad y M. Ziaee (2008). "The general flowshop scheduling problem: mathematical models". En: *J. of app. sci* 8.17, páginas 3032-3037.

Saravanan, M., A. Noorul Haq, A. Vivekraj y T. Prasad (2008). "Performance evaluation of the scatter search method for permutation flowshop sequencing problems". En: *International Journal of Advanced Manufacturing Technology* 37.11, páginas 1200-1208.

Słowiński, R. (1980). "Two approaches to problems of resource allocation among project activities—a comparative study". En: *Journal of the Operational Research Society* 31.8, páginas 711-723.

Sörensen, K. (ene. de 2015). "Metaheuristics-the metaphor exposed". En: *International Transactions in Operational Research* 22.1, páginas 3-18. ISSN: 0969-6016.

Sörensen, K. y F. Glover (2013). "Metaheuristics". En: *Encyclopedia of operations research and management science* 62, páginas 960-970.

Srikar, B. N. y S. Ghosh (1986). "A MILP model for the n-job, m-stage flowshop with sequence dependent set-up times". En: *International Journal of Production Research* 24.6, páginas 1459-1474.

Stafford, E. F. (1988). "On the development of a mixed-integer linear programming model for the flowshop sequencing problem". En: *Journal of the Operational Research Society* 39.12, páginas 1163-1174.

Stafford Jr, E. F. y F. T. Tseng (1990). "On the Srikar-Ghosh MILP model for the iVx M SDST flowshop problem". En: *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH* 28.10, páginas 1817-1830.

Stinson, J. P., E. W. Davis y B. M. Khumawala (1978). "Multiple resource–constrained scheduling using branch and bound". En: *AIIE Transactions* 10.3, páginas 252-259.

Stützle, T. (1998). "Applying iterated local search to the permutation flow shop problem". En: *Technical Report AIDA–98–04, FG Intellektik, FB Informatik, TU Darmstadt*.

Stützle, T. (2006). "Iterated local search for the quadratic assignment problem". En: *European journal of operational research* 174.3, páginas 1519-1539.

Taillard, E. (1990). "Some efficient heuristic methods for the flow shop sequencing problem". En: *European Journal of Operational Research* 47.1, páginas 65-74. ISSN: 0377-2217.

Taillard, E. (1993). "Benchmarks for basic scheduling problems". En: *European Journal of Operational Research* 64.2, páginas 278-285.

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*. Volumen 74. John Wiley & Sons.

Tasgetiren, M. F., Y.-C. Liang, M. Sevkli y G. Gencyilmaz (2007). "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem". En: *European journal of operational research* 177.3, páginas 1930-1947.

Tasgetiren, M. F., Q.-K. Pan, P. N. Suganthan y A. H. Chen (2011). "A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops". En: *Information sciences* 181.16, páginas 3459-3475.

Tomazella, C. P. y M. S. Nagano (2020). "A comprehensive review of Branch-and-Bound algorithms: Guidelines and directions for further research on the flowshop scheduling problem". En: *Expert Systems with Applications* 158, página 113556. ISSN: 0957-4174.

Tosun, Ö. (2014). "Using artificial bee colony algorithm for permutation flow shop scheduling problem under makespan criterion". En: *International Journal of Mathematical Modelling and Numerical Optimisation* 5.3, páginas 191-209.

Tseng, F. T., E. F. Stafford Jr y J. N. Gupta (2004). "An empirical analysis of integer programming formulations for the permutation flowshop". En: *Omega* 32.4, páginas 285-293.

Vallada, E. y R. Ruiz (2009). "Cooperative metaheuristics for the permutation flowshop scheduling problem". En: *European Journal of Operational Research* 193.2, páginas 365-376.

Vallada, E., R. Ruiz y J. M. Framinan (feb. de 2015). "New hard benchmark for flowshop scheduling problems minimising makespan". En: *European Journal of Operational Research* 240.3, páginas 666-677.

Vallada, E., R. Ruiz y G. Minella (2008). "Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics". En: *Computers & Operations Research* 35.4, páginas 1350-1373.

Vallada, E., F. Villa y L. Fanjul-Peyro (2019). "Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem". En: *Computers & Operations Research* 111, páginas 415-424. ISSN: 0305-0548.

Villa, F., E. Vallada y L. Fanjul-Peyro (2018). "Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource". En: *Expert Systems with Applications* 93, páginas 28-38. ISSN: 0957-4174.

Wagner, H. M. (1959). "An integer linear-programming model for machine scheduling". En: *Naval research logistics quarterly* 6.2, páginas 131-140.

Widmer, M. y A. Hertz (1989). "A new heuristic method for the flow shop sequencing problem". En: *European Journal of Operational Research* 41.2, páginas 186-193.

Wilson, J. M. (1989). "Alternative formulations of a flow-shop scheduling problem". En: *Journal of the Operational Research Society* 40.4, páginas 395-399.

Yepes-Borrero, J. C., F. Perea, R. Ruiz y F. Villa (2021). "Bi-objective parallel machine scheduling with additional resources during setups". En: *European Journal of Operational Research* 292.2, páginas 443-455. ISSN: 0377-2217.

Yepes-Borrero, J. C., F. Perea, F. Villa y E. Vallada (2023). "Flowshop with additional resources during setups: Mathematical models and a GRASP algorithm". En: *Computers & Operations Research* 154, página 106192. ISSN: 0305-0548.

Yepes-Borrero, J. C., F. Villa, F. Perea y J. P. Caballero-Villalobos (2020). "GRASP algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources". En: *Expert Systems with Applications* 141, página 112959. ISSN: 0957-4174.

Zhang, L. y J. Wu (2014). "A PSO-based hybrid metaheuristic for permutation flowshop scheduling problems". En: *The Scientific World Journal* 2014.

A. Apéndices

En los apéndices se van a colocar los materiales que por espacio o importancia sea preferible tenerlos separados.

A.1 Heurísticas constructivas

En la sección 4.2 se presentan hasta 20 posibles heurísticas constructivas, que se analizan en la sección 4.5.2, en ella se presentan los los promedios de RPD de las heurísticas con las instancias de entrenamiento. Aquí se presenta la tabla A.1 que complementa la anterior. Los resultados no difieren de forma significativa entre los dos sets.

Orden	*Heur*	Promedios de RPD				Tiempo Promedio
	Heurísticas Constructivas	**Todos los sets**	**Set VRF Small**	**Set Taillard**	**Set VRF Large**	**Todos los sets (ms)**
1	*Palmer*	1,63	2,90	1,62	0,38	434,92
2	*KK*1	2,66	3,90	2,99	1,10	439,08
3	LPT_f	2,74	4,11	3,00	1,10	763,98
4	LPT_t	2,74	4,07	3,02	1,11	768,92
5	$LAAr_t$	3,43	4,74	4,07	1,50	765,12
6	$LAAp_f$	3,45	4,72	4,10	1,53	761,32
7	$LAAp_t$	3,45	4,75	4,07	1,53	761,66
8	$LAAr_f$	3,46	4,73	4,09	1,54	765,38
9	*SAR*	3,51	5,22	3,93	1,40	766,23
10	SPT_t	3,65	5,84	3,74	1,37	781,04
11	*LAR*	3,68	5,60	3,96	1,46	759,79
12	SPT_f	3,71	5,94	3,83	1,37	761,67
13	SRC_f	3,87	6,02	4,18	1,41	768,55
14	SRC_t	3,94	6,19	4,18	1,44	761,19
15	LRC_f	3,96	5,80	4,38	1,71	759,84
16	LRC_t	3,97	5,74	4,47	1,70	766,50
17	$SAAp_f$	4,07	6,53	4,17	1,50	759,34
18	$SAAr_t$	4,07	6,53	4,19	1,49	758,18
19	$SAAp_t$	4,08	6,50	4,24	1,49	767,82
20	$SAAr_f$	4,08	6,51	4,24	1,50	758,93

Tabla A.1: Resultados RPD promedio de las heurísticas constructivas con instancias de test.

A.2 Heurísticas constructivas multipasada

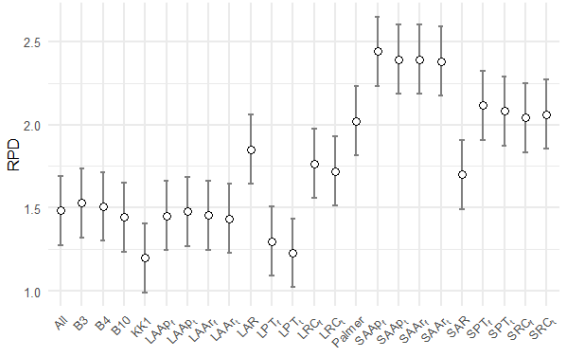
A continuación, se presentan los resultados del RPD promedio para los cuatro algoritmos multipasada en la tabla A.2 con las instancias de test. Los resultados de la tabla se pueden ver en las figura 4.4b

Orden	*Heur*	Promedios de RPD				Tiempo Promedio
	Heurísticas Multipasada	**Todos los sets**	**Set VRF Small**	**Set Taillard**	**Set VRF Large**	**Todos los sets (s)**
1	*All*	0,00	0,00	0,00	0,00	18,91
2	*B*10	0,11	0,12	0,15	0,05	6,86
3	*B*4	0,48	0,78	0,52	0,14	2,37
4	*B*3	0,60	0,98	0,65	0,16	1,61

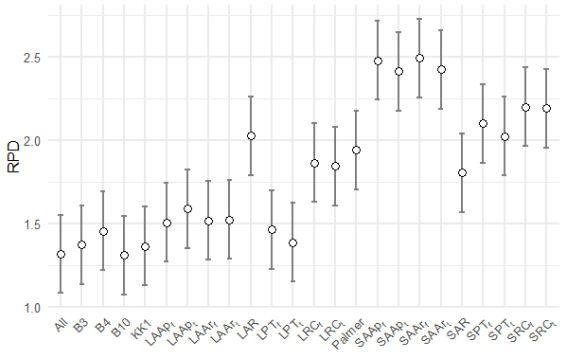
Tabla A.2: Resultados RPD promedio de las agrupaciones de heurísticas constructivas con instancias de test.

A.3 Heurísticas basadas en la NEH

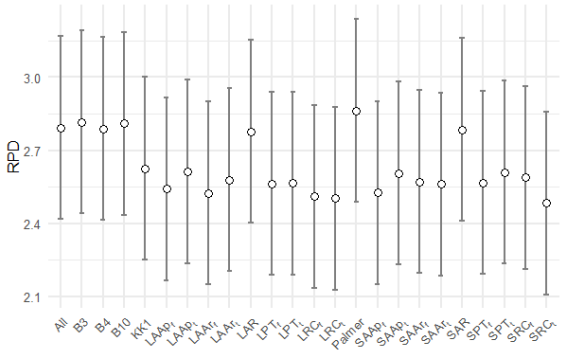
En la sección 4.4 se presentan tres heurísticas basadas en la NEH: NEH_{cf}, NEH_a y NEH_{nr}. Y se realiza la comparación de los resultados promedio de RPD. Sin embargo, aparecen solo los resultados de cada NEH con la "regla" de ordenación que mejores resultados da. Para comprobar los diferentes rendimientos de estas NEH con las reglas simultáneamente se presentan las tablas A.3 y A.4. Estos resultados separados por tipo de NEH se proporcionan en las figuras A.1, donde se aprecian los RPD promedio con intervalos de Tukey. Las lecturas son diferentes, para el método NEH_{nr} como se aprecia en la figura A.1c el tipo de ordenación demuestra no ser estadísticamente significativo. En las otras dos heurísticas NEH (figuras A.1a y A.1b) si hay diferencias significativas, pero no entre todas las reglas.



(a) NEH_{cf}



(b) NEH_a



(c) NEH_{nr}

Figura A.1: Promedios de RPD para las diferentes ordenaciones de la NEH con intervalos de confianza de Tukey.

	NEH con	Todas las instancias		VRF Small		Taillard		VRF Large	
Orden	Heurística	ARPD	Tiempo	ARPD	Tiempo	ARPD	Tiempo	ARPD	Tiempo
1	$\mathbf{NEHcf_{KK1}}$	**1,39**	339060,68	1,58	1350,79	**1,22**	64487,21	1,3	814057,31
2	$NEHcf_{LPT_t}$	1,42	339513,58	**1,5**	1343,3	1,36	64559,1	1,38	815161,1
3	$NEHcf_{LPT_f}$	1,49	339457,52	1,63	1392,6	1,52	64493	1,35	815004,7
4	$NEHcf_{LAAr_t}$	1,63	339179,61	1,71	1382,85	1,23	63972,83	1,75	814579,75
5	$NEHcf_{B10}$	1,64	339483,19	2,21	1398,81	1,74	64168,12	**1,00**	815225,1
6	$NEHcf_{LAAp_f}$	1,65	339258,24	1,69	1396,36	1,31	64429,46	1,77	814534,51
7	$NEHcf_{LAAr_f}$	1,65	338614,5	1,75	1345,9	1,34	64880,33	1,71	812750,2
8	$NEHcf_{LAAp_t}$	1,67	339104,44	1,75	1337,96	1,29	64829,56	1,79	814008,35
9	$NEHcf_{All}$	1,68	339002,65	2,2	1465,52	1,87	64854,96	1,05	813613,61
10	$NEHcf_{B4}$	1,7	339054,78	2,28	1337,93	1,9	64719,54	1,02	813939,24
11	$NEHcf_{B3}$	1,72	339171,16	2,31	1407,42	1,77	64616,1	1,11	814212,43
12	$NEHcf_{SARr_f}$	1,89	339328,35	2,2	1315,94	1,97	64767,17	1,55	814621,34
13	$NEHcf_{LRC_t}$	1,92	339555,84	1,8	1397,81	1,93	64381,71	2,03	815300,94
14	$NEHcf_{LRC_f}$	1,96	339340,32	2	1327,82	1,72	65037,25	2,05	814504,34
15	$NEHcf_{LARr_f}$	2,05	339142,56	2,52	1308,95	1,84	65262,6	1,68	813916,15
16	$\mathbf{NEHa_{B10}}$	**2,08**	338746,41	2,21	1400,95	2,1	64728,33	1,94	813100,91
17	$NEHa_{All}$	2,09	339249,34	2,2	1539,25	2,44	64644,1	**1,8**	814262,05
18	$NEHa_{B3}$	2,14	338736,76	2,31	1404,8	2,14	64542,21	1,98	813166
19	$NEHa_{KK1}$	2,14	339029,16	1,58	1326,89	1,87	64156,12	2,85	814167,96
20	$NEHa_{LPT_t}$	2,17	339187,7	**1,5**	1364,16	2,17	65510,56	2,83	813849,8
21	$NEHa_{B4}$	2,22	339043,41	2,28	1375,52	2,65	65062,35	1,96	813701,83
22	$NEHcf_{Palmer}$	2,22	339116,46	3,13	1346,26	2,34	64995,94	1,25	813946,93
23	$NEHa_{LPT_f}$	2,24	339245,69	1,63	1352,57	2,11	65392,77	2,93	814065,26
24	$NEHcf_{SRC_f}$	2,24	339875,54	2,96	1421,11	2,03	65724,5	1,62	815405,48
25	$NEHcf_{SRC_t}$	2,26	339002,3	2,96	1373,42	2,32	65181,46	1,52	813541,61
26	$NEHcf_{SPT_t}$	2,28	339576,89	2,77	1404,01	2,09	64703,06	1,87	815186,69
27	$NEHa_{LAAp_f}$	2,29	338849,75	1,69	1328,86	1,95	64433,08	3,06	813578,97
28	$NEHa_{LAAr_f}$	2,3	338877,2	1,75	1338,46	**1,76**	64568,48	3,13	813570,3
29	$NEHa_{LAAr_t}$	2,31	338943,68	1,71	1369,68	1,9	64703,96	3,11	813637,54
30	$NEHcf_{SPT_f}$	2,31	339088,27	2,96	1342,93	2,15	64753,5	1,75	814000,99
31	$NEHa_{LAAp_t}$	2,37	339734,31	1,75	1375,02	1,95	64465,48	3,2	815728,01
32	$NEHcf_{SAAr_t}$	2,58	339134,37	3,39	1347,71	2,59	64772,67	1,75	814101,88
33	$NEHa_{SARr_f}$	2,59	339029,32	2,2	1381,45	2,77	64500,25	2,88	813941,72
34	$NEHcf_{SAAp_t}$	2,59	339477,03	3,34	1387,31	2,7	65079,65	1,78	814765,45
35	$NEHcf_{SAAr_f}$	2,59	339158,33	3,43	1387,08	2,6	64770,67	1,74	814123,41
36	$NEHa_{LRC_t}$	2,63	339113,86	1,8	1407,47	2,92	64695,46	3,32	814029,46
37	$NEHcf_{SAAp_f}$	2,63	339615,89	3,5	1326,23	2,55	64810,77	1,81	815308,11
38	$NEHa_{LRC_f}$	2,65	338849,13	2	1339,88	2,53	65243,69	3,35	813161,11
39	$NEHa_{Palmer}$	2,71	338877,2	3,13	1350,32	2,89	64761,23	2,21	813462,07
40	$NEHa_{SPT_t}$	2,8	338902,54	2,77	1369,71	2,66	64855,29	2,9	813458,99
41	$NEHa_{LARr_f}$	2,81	339219	2,52	1406,29	2,37	64438,33	3,32	814422,04
42	$NEHa_{SPT_f}$	2,88	338639,77	2,96	1387,41	2,87	64548,08	2,81	812937,97
43	$NEHa_{SRC_t}$	2,97	339549,79	2,96	1374,73	2,94	64717,27	3	815141,11
44	$NEHa_{SRC_f}$	2,98	339158,43	2,96	1396,92	2,62	64660,4	3,19	814168,97
45	$NEHa_{SAAp_t}$	3,19	339157,17	3,34	1362,35	3,24	65189	3,02	813936,08
46	$NEHa_{SAAr_t}$	3,21	338843,76	3,39	1401,1	3,17	64859,54	3,04	813278,52
47	$NEHa_{SAAp_f}$	3,26	339405,85	3,5	1429,6	3,37	65655,48	2,96	814257,27
48	$NEHa_{SAAr_f}$	3,27	339617,79	3,43	1375,07	3,21	64971,85	3,15	815183,47

Tabla A.3: Tabla con los resultados promedio de la desviación porcentual relativa (RPD) para los tipos de NEH que evalúan usando recursos.

	NEH con	Todas las instancias		VRF Small		Taillard		VRF Large	
Orden	Heurística	ARPD	Tiempo	ARPD	Tiempo	ARPD	Tiempo	ARPD	Tiempo
1	$\mathbf{NEHnr_{SRC_t}}$	**7,1**	9815,24	**10,81**	28,08	10,66	989,35	1,61	24015,34
2	$NEHnr_{LAAr_f}$	7,14	10113,43	10,99	28,38	10,48	936,77	1,64	24786,82
3	$NEHnr_{LRC_t}$	7,15	10392,54	10,83	25,15	10,73	958,46	1,68	25476,97
4	$NEHnr_{LAAp_f}$	7,16	10050,32	11,01	25,5	10,49	983,33	1,65	24608,64
5	$NEHnr_{LRC_f}$	7,16	10407,64	10,87	27,08	10,76	983,29	1,64	25500,38
6	$NEHnr_{LPT_f}$	7,17	10223,59	11,11	26,94	10,24	953,98	1,7	25055,05
7	$NEHnr_{SAAp_f}$	7,17	10261,78	10,93	27,77	11,11	1002,25	**1,44**	25125,56
8	$NEHnr_{LPT_t}$	7,18	10357,05	11,15	25,31	10,28	890,23	1,65	25422,21
9	$NEHnr_{LAAr_t}$	7,2	10411,86	11,09	28,55	10,48	1021,65	1,67	25490,28
10	$NEHnr_{SAAr_t}$	7,2	10034,75	10,93	29,39	11,1	949,79	1,53	24582,6
11	$NEHnr_{SPT_f}$	7,2	10122,5	11,02	25,82	10,78	981,67	1,58	24789,58
12	$NEHnr_{SAAr_f}$	7,21	10286,85	11,04	25,29	10,96	1040,96	1,51	25171,36
13	$NEHnr_{SRC_f}$	7,21	10666,69	11,06	25,22	10,77	977,81	1,59	26152,59
14	$NEHnr_{KK1}$	7,24	9673,53	11,27	26,93	10,38	958,9	1,63	23677,46
15	$NEHnr_{LAAp_t}$	7,24	10334,67	11,12	25,46	10,52	915,62	1,71	25353,41
16	$NEHnr_{SAAp_t}$	7,25	10391,05	11,08	26,92	10,91	1013,62	1,59	25443,91
17	$NEHnr_{SPT_t}$	7,25	10423,2	10,95	25,8	11,24	1001,23	1,54	25531,57
18	$NEHnr_{B4}$	7,41	11794,14	11,01	38,1	11,17	1191,96	1,93	28851,26
19	$NEHnr_{LARr_f}$	7,41	10119,81	11,51	25,95	11,05	982,96	1,5	24782,09
20	$NEHnr_{SARr_f}$	7,41	10030,6	11,69	26,59	**10,15**	926,96	1,75	24586,42
21	$NEHnr_{All}$	7,42	29306,2	10,9	149,93	11,43	3534,38	1,93	71348,39
22	$NEHnr_{B10}$	7,43	16318,17	10,95	70,02	11,4	1720,6	1,93	39865,11
23	$NEHnr_{B3}$	7,44	10761,72	11,09	33,79	11,17	1060,17	1,93	26340,42
24	$NEHnr_{Palmer}$	7,5	10123,86	11,31	23,86	10,99	907,83	1,93	24831,88

Tabla A.4: Tabla con los resultados promedio de la desviación porcentual relativa (RPD) para la NEH que evalúa sin usar recursos con todas sus posibles ordenaciones.

A.4 Resultados metaheurísticas

En la sección 5.7.4 se presentan los resultados de la experimentación de las metaheurísticas propuestas con el conjunto de los tres sets de instancias. Estos resultados se completan añadiendo un análisis independiente para cada set de instancias en las secciones: 5.7.1, 5.7.2 y 5.7.3. Para el análisis de la varianza ANOVA se han estudiado y se cumplen todos los supuestos. A continuación, se facilitan figuras referentes a cada análisis ANOVA. Para el conjunto de los tres sets de instancias juntas podemos ver que se cumple la homocedasticidad en la Figura A.2, la normalidad en un histograma superpuesto con una normal en la Figura A.3, la dispersión en los factores Algoritmo y ρ A.5. Además, podemos ver que la réplica no es estadísticamente significativa en la Figura A.4.

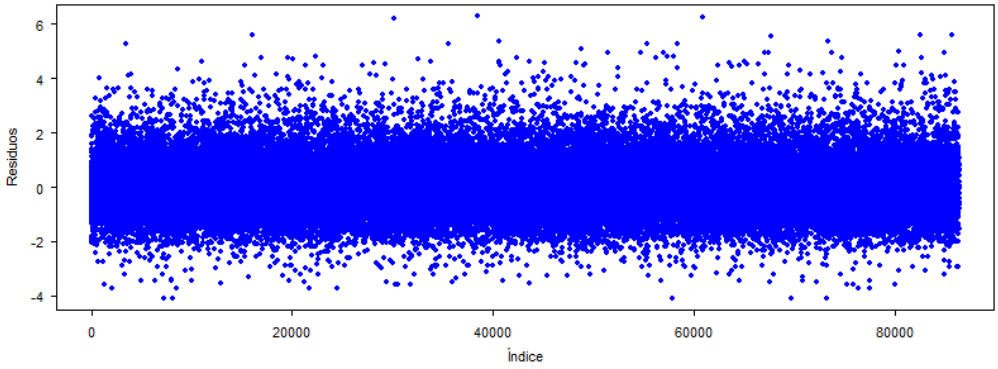


Figura A.2: Gráfico de homocedasticidad del ANOVA de test para el set All.

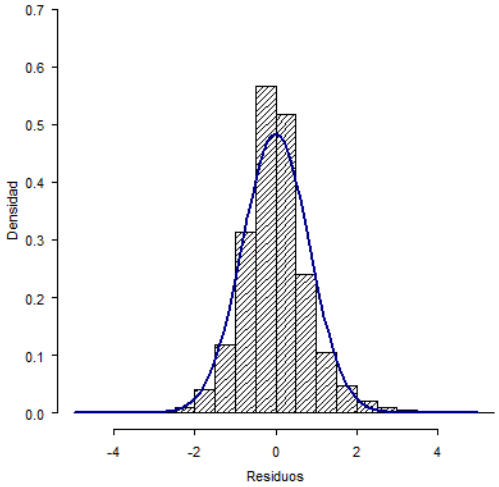


Figura A.3: Gráfico con histograma de residuos y normalidad (All).

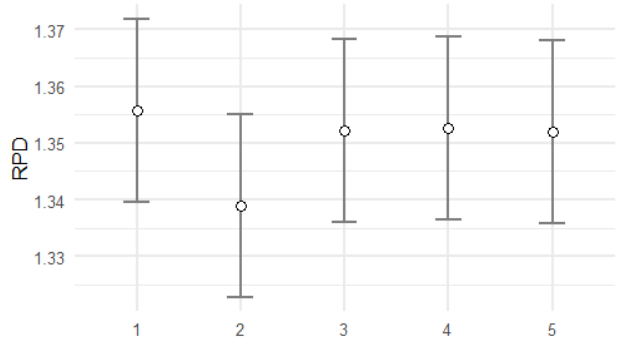
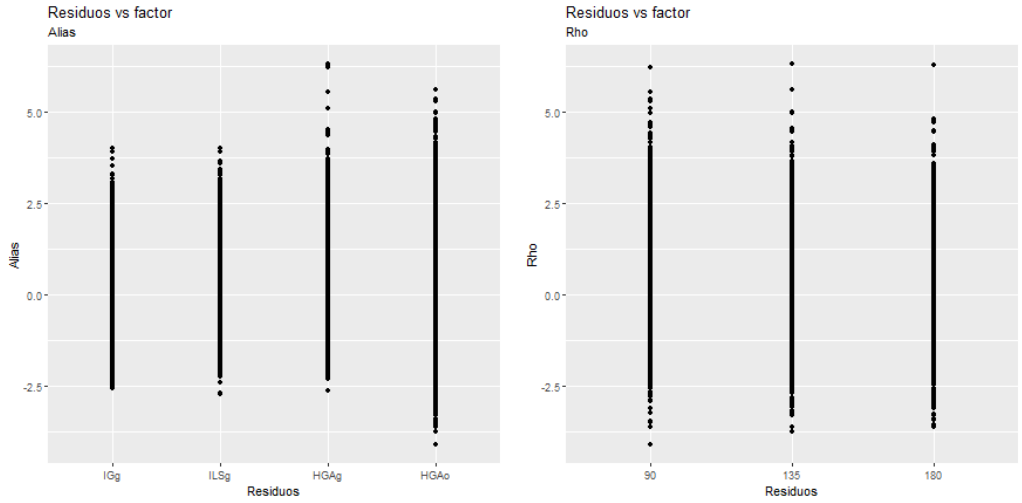


Figura A.4: Significación de las réplicas (All).



(a) Algoritmo (*Alias*) (b) Rho (ρ)

Figura A.5: Gráficos de residuos para los factores algoritmo y rho (All).

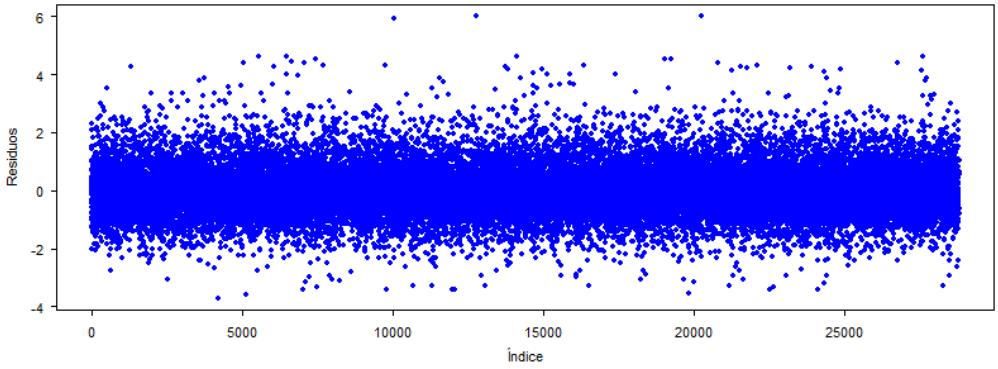


Figura A.6: Gráfico de homocedasticidad del ANOVA de test para el set VRFS.

Para el set de instancias VRFS se muestra el gráfico de homocedasticidad en la Figura A.6, la normalidad en la Figura A.7, la dispersión en los factores Algoritmo y ρ A.9. De nuevo la réplica no es estadísticamente significativa (Figura A.8).

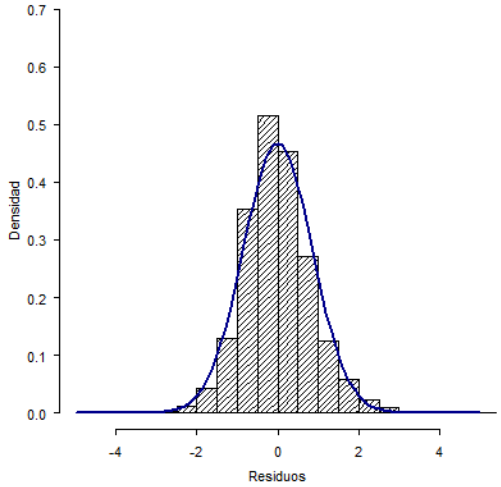


Figura A.7: Gráfico con histograma de residuos y normalidad para el set VRFS.

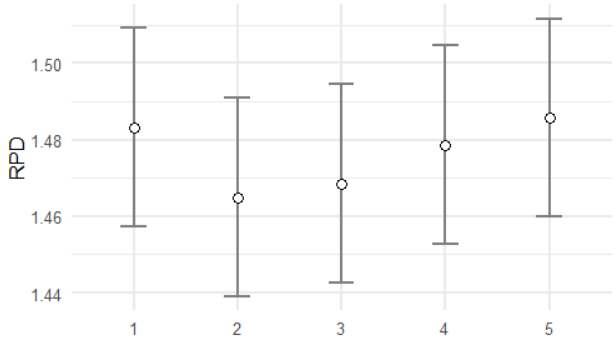
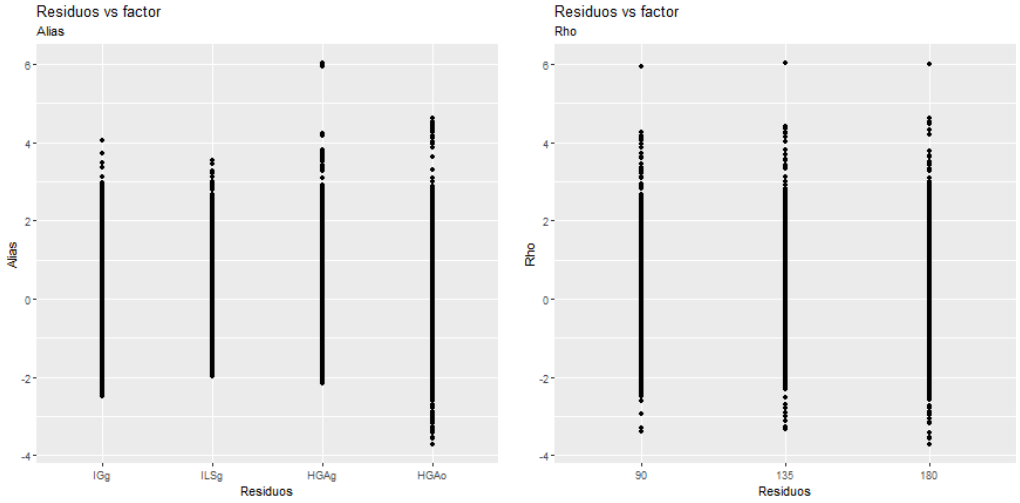


Figura A.8: Significación de las réplicas (VRFS).



(a) Algoritmo (*Alias*) (b) Rho (ρ)

Figura A.9: Gráficos de residuos para los factores algoritmo y rho (VRFS).

Las figuras de los supuestos ANOVA para el set de instancias de Taillard: la homocedasticidad en la Figura A.10, la normalidad en la Figura A.11, la dispersión en los factores Algoritmo y ρ (A.13). La réplica no es estadísticamente significativa (Figura A.12).

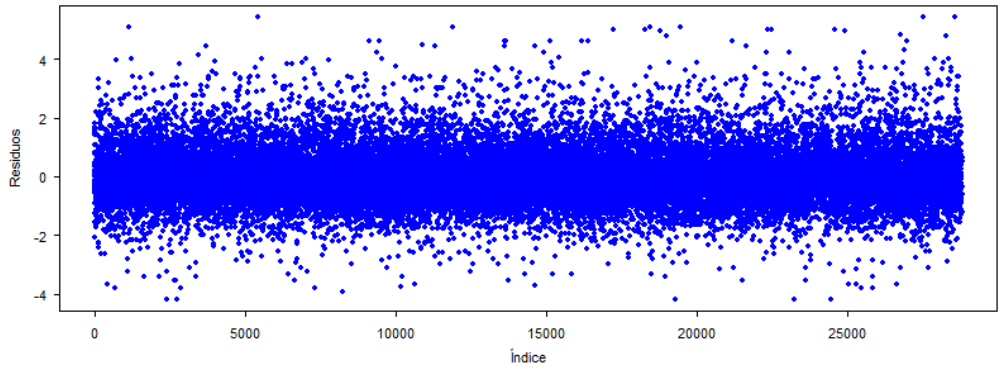


Figura A.10: Gráfico de homocedasticidad del ANOVA de test para el set Taillard.

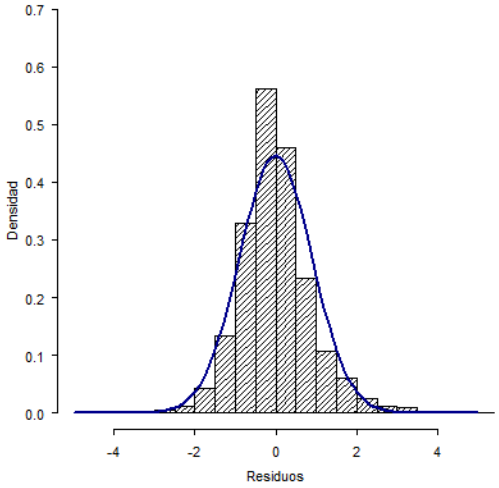


Figura A.11: Gráfico con histograma de residuos y normalidad para el set Taillard.

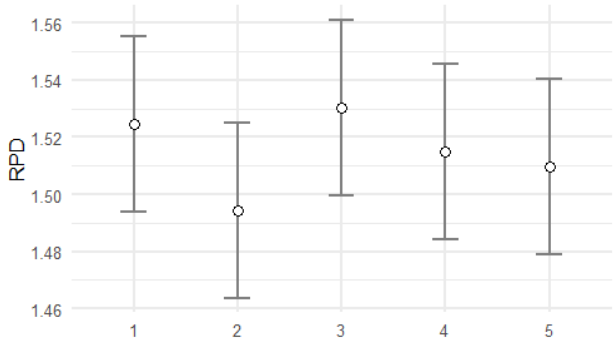
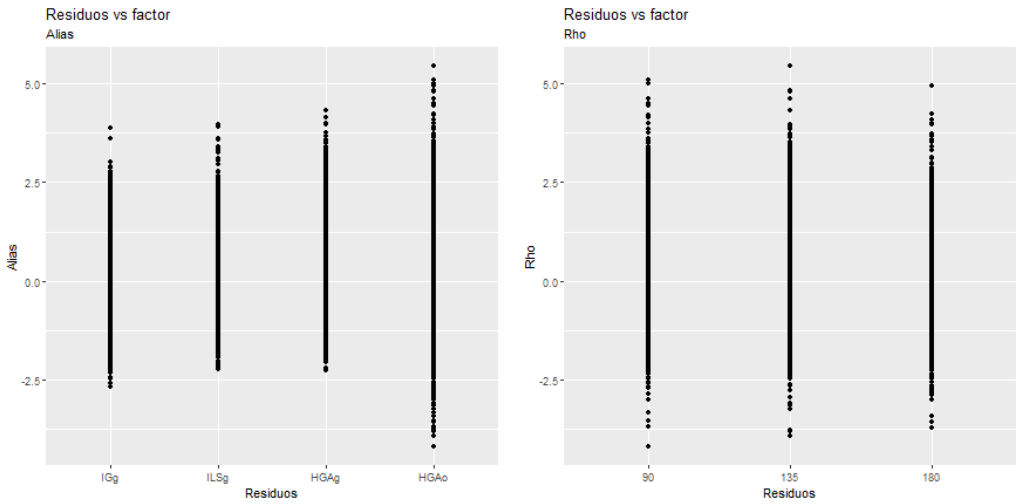


Figura A.12: Significación de las réplicas (Taillard).



(a) Algoritmo (*Alias*) (b) Rho (ρ)

Figura A.13: Gráficos de residuos para los factores algoritmo y rho (Taillard).

El gráfico de homocedasticidad para el conjunto de instancias VRFL se presenta en la Figura A.14, la normalidad se ilustra en la Figura A.15. La dispersión en los factores Algoritmo y ρ se presenta en la Figura A.17. Como se puede apreciar en la Figura A.16, la réplica no resulta ser estadísticamente significativa.

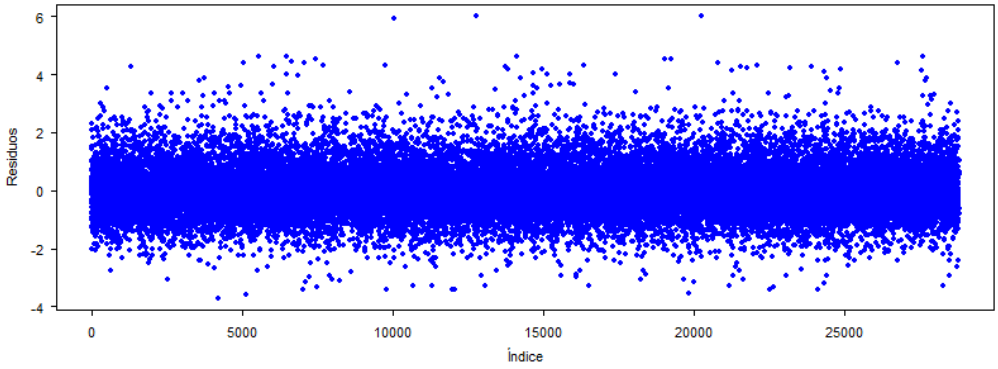


Figura A.14: Gráfico de homocedasticidad del ANOVA de test para el set VRFL.

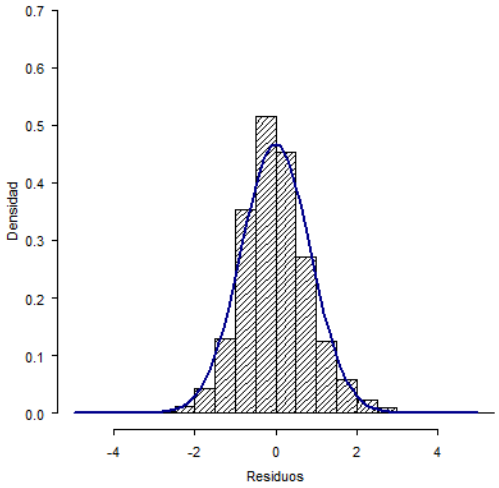


Figura A.15: Gráfico con histograma de residuos y normalidad para el set VRFL.

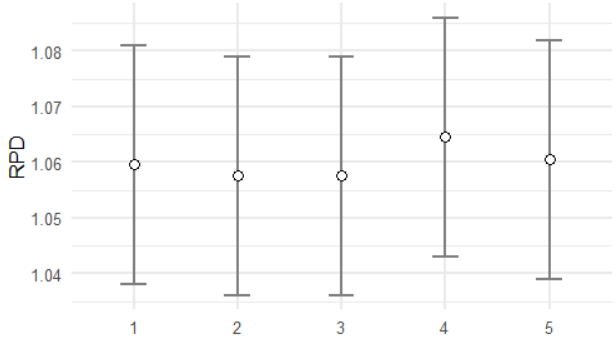
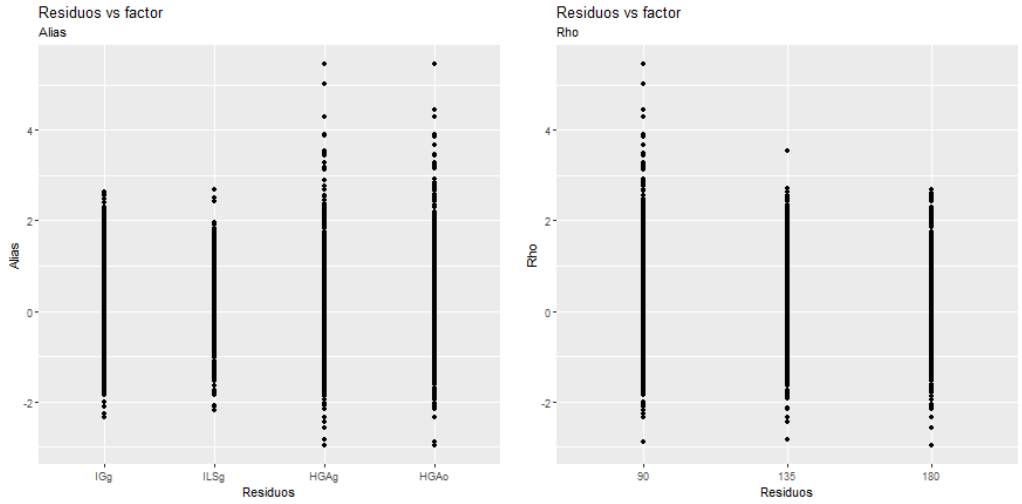


Figura A.16: Significación de las réplicas (VRFL).



(a) Algoritmo (*Alias*) (b) Rho (ρ)

Figura A.17: Gráficos de residuos para los factores algoritmo y rho (VRFL).