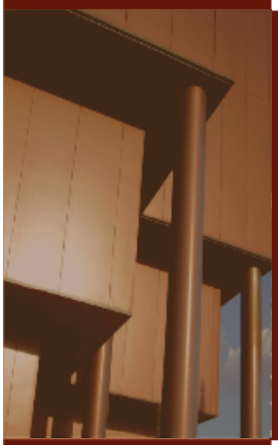
Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Programación de operaciones en flujo uniforme con fechas de comienzo

Autor: Guillermo Díaz – Toledo Álvarez

Tutor: José Manuel Framiñán Torres



Dpto. de Organización Industrial y Gestión de Empresas
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Programación de operaciones en flujo uniforme con fechas de comienzo

Autor:

Guillermo Díaz – Toledo Álvarez

Tutor:

José Manuel Framiñán Torres

Catedrático de Universidad

Dpto. de Organización Industrial y Gestión de Empresas

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado: Programación de operaciones en flujo uniforme con fechas de comienzo

Autor: Guillermo Díaz – Toledo Álvarez

Tutor: José Manuel Framiñán Torres

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

RESUMEN

En este Trabajo de Fin de Grado se aborda el problema de la programación de trabajos en flujo uniforme con fechas de comienzo, también conocido como flowshop scheduling with release dates. Se realizará una revisión de los conceptos fundamentales y de los distintos métodos aproximados para resolver estos problemas. Por último, se adaptarán y evaluarán los distintos métodos, analizando la influencia de los parámetros, la calidad de las soluciones obtenidas y los tiempos de ejecución.

ABSTRACT

This work aims to solve the flowshop scheduling problem with release dates. Firstly, a review of the main concepts and approximate methods to solve these problems will be conducted. Finally, the methods will be adapted and evaluated, analysing the influence of the parameters, the solutions' quality and the runtimes.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN Y OBJETIVOS DEL TRABAJO

La correcta programación de las operaciones supone, especialmente en el sector industrial, unas considerables reducciones de los plazos de entrega al igual que notables incrementos en la productividad y eficiencia de las empresas, que pueden ser necesarias para la supervivencia empresarial en un entorno de alta competitividad. Algunos ejemplos serían la producción de vehículos, el ensamblaje de productos tecnológicos u otros problemas no industriales como la gestión de las puertas de embarque en los aeropuertos, obras civiles o las mercancías de un puerto.

Uno de los problemas de programación de la producción más conocidos es la programación de trabajos en flujo uniforme, ya que es la configuración más extendida en la industria y en la mayoría de los casos en los que no se da, es esta una aproximación razonable. Dentro de este tipo de problemas es habitual que cada uno de los trabajos tenga una fecha de comienzo, a partir de la cual este está disponible para ser realizado, en muchos casos esto es debido a la necesidad de recibir las piezas necesarias de un proveedor.

1.1 Objetivos

Mediante la realización de este trabajo se persigue la identificación de los algoritmos más eficientes para la minimización del tiempo necesario para completar una serie de trabajos en flujo uniforme con fechas de comienzo distintas del instante inicial. Para ello, se adaptaron uno de los mejores algoritmos para el caso en el que las fechas de comienzo son todas el instante cero, propuesto por Víctor Fernández - Viagas y José Manuel Framiñán (Fernández – Viagas, Framiñán, 2014), y el algoritmo en el que este está basado, propuesto por Muhammad Nawaz, E. Emory Enscore Jr. y Inyong Ham (Nawaz, Enscore, Ham, 1983). Los resultados de estos se compararon con los de las recientes contribuciones de Danyu Bai y Lixin Tang (Bai, Tang, 2009) y Tao Ren, Meiting Guo, Lin Lin y Yunhui Miao (Ren et al., 2015).

Los métodos a comparar se implementaron en el lenguaje de programación C++, como se indica en la Sección 4.1. Posteriormente, Capítulo 5, se analizarán los resultados obtenidos, utilizando las herramientas del programa *Microsoft Excel*, comparando la eficiencia de los métodos antes mencionados. Para ello se compararán los tiempos de ejecución y las calidades de las soluciones obtenidas por cada uno de ellos en función de los distintos parámetros del problema.

1.2 Justificación

El ahorro de tiempo, y por tanto de costes, que aporta una adecuada programación de las operaciones hace que el estudio de este tipo de problemas sea de gran interés. El hecho de que muchos de estos problemas, y en concreto el que ocupa este trabajo (programación de trabajos en flujo uniforme con fechas de comienzo), sean denominados *NP-hard* y que por tanto, como se explicara en la Sección 2.3, la ingente cantidad de tiempo y recursos necesarios para la resolución exacta del problema hace que sea necesario desarrollar algoritmos que permitan encontrar resultados suficientemente próximos a la solución óptima del problema en un tiempo adecuado para el uso que se vaya a dar a ese resultado.

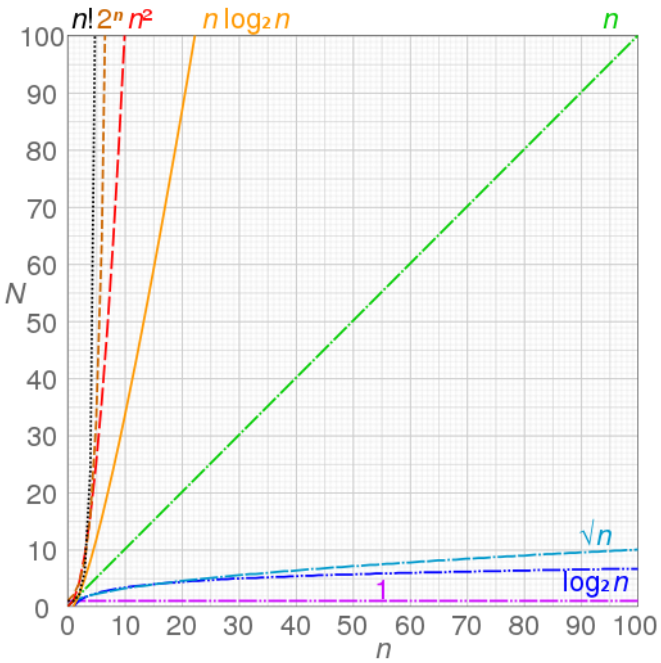


Figura 1. Relación entre el tiempo necesario para resolver un problema y el tamaño de este en función de su complejidad. [Fuente: Cmglee. commons.wikimedia.org]

1.3 Estructura

Tratando de cumplir con estos objetivos, el trabajo se ha estructurado en los 6 capítulos que se describen a continuación:

En el primer capítulo, el presente, se introduce el problema a estudiar, y se justifica su utilidad. Por último, se explica la estructura y los objetivos que se persiguen con la realización de este Trabajo de Fin de Grado.

El segundo capítulo presenta y describe el problema sobre el que versa el trabajo y se comentan sus particularidades y las razones por las que es interesante estudiarlo. Previamente, se introducirán los conceptos básicos y se realizará una breve reseña histórica de la disciplina de Programación de la Producción.

El capítulo 3 se dedica a presentar y explicar los principios en que se basan los distintos algoritmos/heurísticas a comparar, describiendo los pasos que siguen estos y las modificaciones realizadas a los mismos. También se indicarán las distintas ventajas e inconvenientes que presentan cada uno de ellos.

En el capítulo 4 se explica el *software* utilizado para la realización de los experimentos y el posterior procesado de los resultados. A continuación, se presentarán el conjunto de problemas con el que se probarán y compararán los algoritmos, se asignará y justificará el valor de los parámetros y se describirán los indicadores con los que se compararán los distintos algoritmos.

Mediante el quinto capítulo se procede a mostrar y analizar los resultados. Se comparan los resultados obtenidos por los distintos algoritmos al igual que los distintos tiempos de ejecución de cada uno de ellos, analizando la influencia de los parámetros en la calidad de las soluciones obtenidas por cada uno de los métodos.

Por último, en el sexto capítulo, se muestran las conclusiones más significativas que se han obtenido tras la realización de este trabajo.

2 DESCRIPCIÓN DEL PROBLEMA

En este apartado, como se ha comentado en la Sección 1.3, se describirá el problema sobre el que versa este trabajo y previamente se introducirán algunos conceptos importantes que sirven como base de este problema, y por tanto son convenientes para su comprensión.

2.1 Programación de la Producción

La programación de la producción, universalmente conocida como *scheduling* o en castellano secuenciación o programación de operaciones, se puede definir como el proceso de asignar en el tiempo los recursos disponibles con el objetivo de realizar un trabajo en el menor tiempo posible, al mínimo coste y con la mínima utilización de los recursos disponibles (Framiñan, Leisten, Ruíz, 2014).

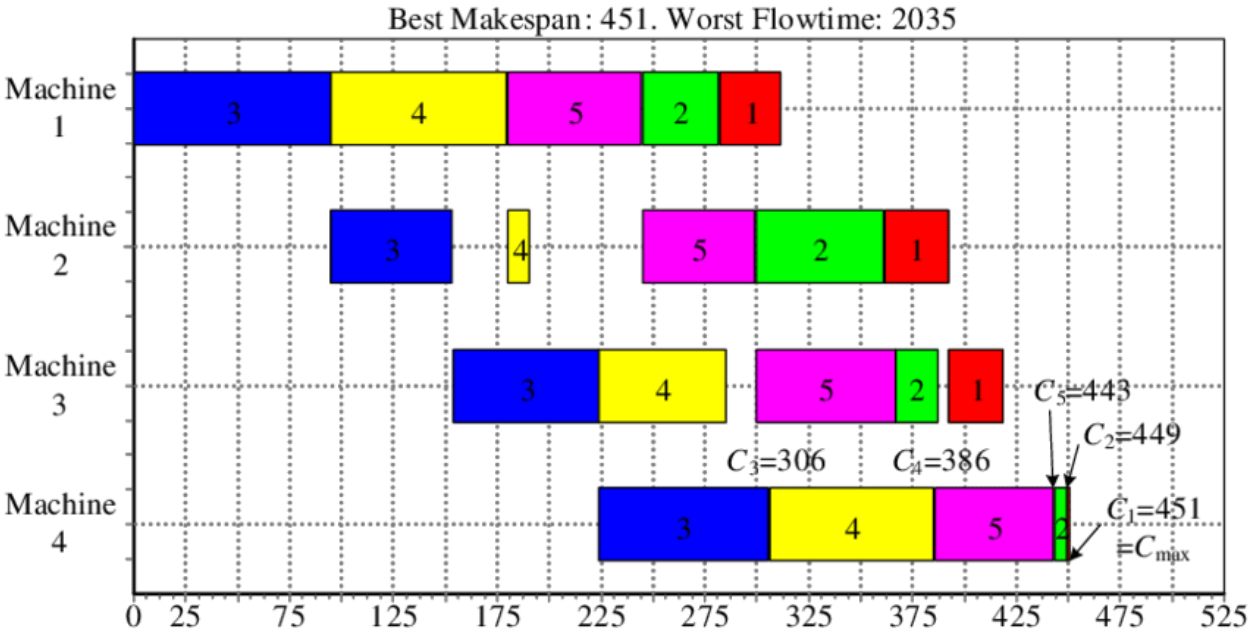


Figura 2. Diagrama de Gantt que muestra la secuencia de un proceso de tipo Flowshop. [Fuente: Ruiz, R., 2015]

Una de las primeras herramientas desarrolladas con el fin de programar y supervisar el desarrollo de trabajos es el diagrama de Gantt, que debe su nombre al ingeniero mecánico y consultor estratégico Henry Laurence Gantt quien en la década de 1910 modificó y popularizó el uso de este tipo de diagramas. Pero fue durante la Segunda Guerra Mundial cuando se produjeron los primeros avances significativos en el campo de la gestión de operaciones, tras el final de la guerra y con la posterior llegada de los primeros ordenadores modernos muchos de los científicos que trabajaron en este campo durante la misma continuaron desarrollando las herramientas y teorías que forman la base sobre la que se sustenta la Investigación Operativa, y concretamente la programación de operaciones, en la actualidad. El aumento del tamaño y la complejidad de los problemas obligó a desarrollar métodos aproximados, que como se verá en las Secciones 2.2 y 2.3 son la forma mas eficiente de resolver determinados problemas.

2.2 Algoritmos

“Algoritmo: Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.”

(Diccionario de la lengua española, Real Academia Española)

Se trata de una secuencia finita de operaciones bien definidas y ordenadas, con el fin de obtener una solución a un problema. Los algoritmos se pueden clasificar según la precisión de sus soluciones en:

- Exactos: Proporcionan la solución óptima o exacta del problema.
- Aproximados: Estos se caracterizan por una aproximación intuitiva al problema, que permite encontrar una solución en un menor tiempo, aunque renunciando a la exactitud, a encontrar todas las soluciones o a encontrar una solución óptima o más cercana a esta. Son de gran utilidad dado que la complejidad computacional de muchos problemas hace inviable el uso de algoritmos exactos para resolverlos, por lo que se usan este tipo de algoritmos para obtener soluciones útiles con unos recursos razonables.

También es posible clasificarlos según la aleatoriedad de sus soluciones:

- Deterministas: Para unas entradas dadas el algoritmo siempre devolverá la misma solución.
- No deterministas o aleatorios: Se caracterizan por que pueden obtener distintas soluciones para un mismo problema. Esto se debe a la introducción de algún factor de aleatoriedad, como puede ser el caso de que la solución dependa de un número generado al azar. Se usa habitualmente cuando la solución es aproximada.

2.3 Complejidad de los algoritmos

La forma mas habitual de clasificar los algoritmos según su complejidad es mediante el tiempo necesario para resolverlos en una máquina de Turing. La máquina de Turing es un dispositivo hipotético, definida por el matemático inglés Alan Turing en 1936, que ha resultado de gran utilidad para el desarrollo teórico de la teoría de la complejidad computacional. Según este criterio las clases de algoritmos de más interés son:

- Clase P: A esta categoría pertenecen todos los problemas que pueden ser un resueltos en tiempo polinomial por una máquina de Turing determinista.
- Clase NP: Esta categoría está formada por todos aquellos problemas que pueden ser resueltos en tiempo polinomial por una máquina de Turing no determinista, o lo que es lo mismo, todos los problemas cuya solución se puede verificar en tiempo polinomial por una máquina de Turing determinista.
- Clase NP-hard: Grosso modo pertenecen a esta categoría los problemas con una complejidad mayor o igual que los de la clase NP.
- Clase NP-completo: Esta categoría la forman los problemas que siendo de clase NP lo son también de la clase NP-hard.

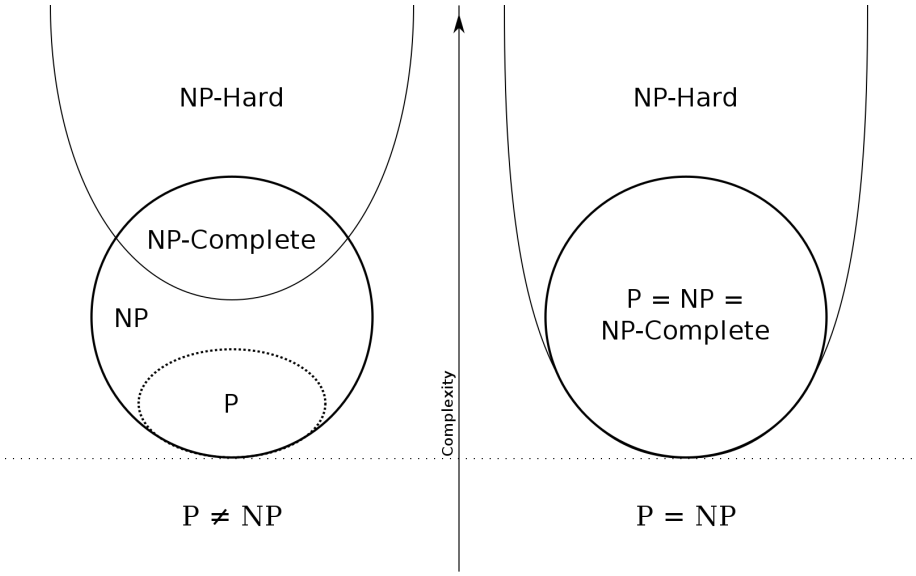


Figura 3. Diagrama de Euler de las relaciones de las clases de algoritmos, suponiendo que P es igual a NP y que P es distinto de NP. [Fuente: Behnam Esfahbod. commons.wikimedia.org]

Pero esta clasificación sólo es útil en el caso que P sea distinto a NP, cuestión que, Stephen Cook (Cook, 1971) y Leonid Levin (Levin, 1973) definieron formalmente. En sus trabajos y en el de Richard Karp (Karp, 1972) se prueba que algunos problemas relevantes son NP-completos. Es conocido que las soluciones de los problemas de clase NP se pueden verificar en tiempos polinomiales, la pregunta es por tanto si es posible obtener estas también en tiempos polinomiales, lo que implicaría que P sea igual a NP y por tanto los problemas hasta ahora considerados difíciles, es decir los NP-completos, serían resolubles en tiempos polinomiales con los algoritmos adecuados. A día de hoy no se ha conseguido ningún algoritmo capaz de resolver ninguno de los problemas de esta clase en tiempo polinómico.

2.4 Notación y conceptos

En este apartado, se va a definir la notación que se va a usar en este trabajo, que a grandes rasgos es la habitualmente utilizada en esta disciplina.

Dos elementos básicos en este tipo de problemas son los trabajos y las máquinas, se denotan como J_j *(j = 1, ..., n)* a los *n* trabajos que se han de procesar en las *m* máquinas, a las que se designará M_i *(i = 1, ..., m)*. También son fundamentales los tiempos de proceso, que se denotan como t_{ij} al tiempo necesario para realizar el trabajo *j* en la máquina *i* y p_{ij} al tiempo necesario para realizar el trabajo *j* de una determinada secuencia en la máquina i. Un elemento

característico de este tipo problemas es la fecha de comienzo (*release date)* de cada uno de los trabajos r_j, que indica el instante a partir del cual ese trabajo está disponible para ser realizado.

Una vez definida una secuencia hay distintos tiempos de esta, como los siguientes, que son de gran utilidad:

- C_{ij} (*Completion time*): Indica el instante en el que la máquina i finaliza el j-ésimo trabajo.
- e_{ij} (*Earliest completion time*): Indica el menor instante posible en el que la máquina i finaliza el j-ésimo trabajo.
- q_{ij} (*Tail*): Indica el menor tiempo posible desde que la máquina i empieza a procesar el j-ésimo trabajo hasta que se finalizan todos los trabajos, es decir, la cola de producción.
- f_{il} (*Completion time*): Indica el instante en el que la máquina i finaliza el l-ésimo trabajo.
- it_i (*Idle time*): Indica el tiempo que la máquina *i* no se utiliza desde que se empieza a procesar el primer trabajo en la primera máquina hasta que se finaliza el último trabajo en la i-ésima máquina.

Por último, se denota el tiempo de finalización de una secuencia (*Makespan*) como $C_{max}\ (l)$, siendo *l* la posición que ocupa el último trabajo insertado en la secuencia, y se indican las secuencias como π.

2.5 Entorno, restricciones y objetivo

La forma más utilizada para definir el entorno, las restricciones y el objetivo de un problema de programación de la producción es la propuesta por Ronald Graham, Eugene Lawler, Jan Karel Lenstra y Alexander Rinnooy Kan (Graham, Lawler, Lenstra, Rinnooy Kan, 1979), $\alpha \mid \beta \mid \gamma$, indicando α el entorno, β las restricciones y γ el objetivo.

El entorno está determinado por las máquinas y el recorrido que realizan por estas los trabajos. Estos son los distintos entornos:

- *Single Machine (1)*: El entorno lo compone una única máquina.
- *Parallel Identical Machines (Pm)*: El entorno está formado por *m* máquinas con tiempos de procesado idénticos.
- *Parallel Machines (Qm)*: El entorno está formado por *m* máquinas con tiempos de procesado proporcionales a las velocidades de cada una de las máquinas.

- *Parallel Unrelated Machines (Rm)*: El entorno está formado por *m* máquinas con distintos tiempos de procesado.
- *Flowshop (Fm)*: Entorno compuesto por *m* máquinas en el que los trabajos han de pasar por las distintas máquinas en un determinado orden, que es común a todos los trabajos. Es el entorno de este trabajo.
- *Jobshop (Jm)*: Entorno compuesto por *m* máquinas en el que los trabajos han de pasar por las distintas máquinas en una secuencia característica de cada uno de los trabajos.
- *Openshop (Om)*: Entorno compuesto por *m* máquinas en el que la secuencia que han de seguir los trabajos no está previamente definida.

Otra de las características necesarias a la hora de definir un problema son las restricciones. Algunas de las más comunes son las siguientes:

- Secuencia de permutación (*prmu*): Esta restricción solo puede ocurrir en los problemas en entornos *Flowshop*. Indica que el orden en el que los trabajos pasan por cada una de las máquinas ha de ser el mismo. Es una de las restricciones de este trabajo.
- *Release date* (r_j): Hace referencia al instante a partir del cual cada uno de los trabajos está disponible para ser realizado, es también el caso del que trata este trabajo. Esta restricción se puede dar de dos modos:
    - *Offline*: Es el caso que se da por defecto si no se indica lo contrario. Implica que las fechas de comienzo de cada uno de los trabajos son conocidas desde el inicio del problema.
    - *Online*: En este caso no se tiene conocimiento de cada uno de los trabajos hasta su fecha de comienzo, lo que implica que sus operaciones no se pueden programar hasta su fecha de comienzo ya que se desconocía su existencia.
- Relación de precedencia (*prec*): Indica los órdenes de prioridad entre los distintos trabajos. Esta restricción es típica de los procesos de montaje, ya que no es posible ensamblar una pieza hasta que no se han finalizado los trabajos de cada uno de los componentes que la forman.
- Interrupciones (*pmtn*): Permite interrupciones en el procesamiento de los trabajos.
- Tiempos de cambio o *setup* (*s*): Hace referencia al tiempo de preparación necesario en una máquina para poder realizar un trabajo.
- Máquina no ociosa (*no - idle*): Indica que una máquina no puede estar ociosa entre operaciones.

- Procesado en lotes (*batch*): Las operaciones han de realizarse por lotes.
- Almacenaje entre máquinas (*buffer*): Fija el número de trabajos que pueden estar esperando a la entrada de una máquina.
- Espera no permitida o *no-wait* (*nwt*): Indica que el trabajo deberá ser procesado sin esperas.
- Inoperatividad de máquinas (*brkdwn*): Imposibilidad de realizar operaciones en la máquina en un determinado intervalo temporal.
- Recirculación (*rcrc*): Indica que un trabajo debe ser procesado varias veces por la misma máquina.

Por último, para definir un problema de programación de la producción es necesario definir el objetivo del problema. Los más comunes son los siguientes:

- Finalización del último trabajo (C_{max}): Minimizar el instante en el que termina de ser procesado el último trabajo. Este objetivo, además de ser el más habitual, es el objetivo del problema sobre el que trata este trabajo.
- Tiempo total de finalización ($\sum C_j$): Aunque es similar al anterior, en este caso no se busca minimizar únicamente el tiempo de finalización del último trabajo sino el de cada uno de los distintos trabajos.
- *Flowtime* ($\sum F_j$): También similar al anterior, pero en este caso se descuenta el tiempo anterior a que cada uno de los trabajos estuviera disponible, es decir $F_j = C_j - r_j$.
- *Lateness* ($\sum L_j$): El objetivo es minimizar la suma de los retrasos de cada uno de los trabajos, pudiendo ser esta positiva o negativa. Otro caso habitual es minimizar el mayor de los retrasos (L_{max}).
- Tiempo total de retrasos ($\sum T_j$): El objetivo es minimizar la suma de los retrasos de cada uno de los trabajos, considerando este nulo si el trabajo se finaliza en plazo. Otro caso habitual es minimizar el mayor de los retrasos (T_{max}).
- Número de retrasos ($\sum U_j$): Minimizar el número de trabajos finalizados fuera de plazo.

2.6 Descripción del problema

Como se ha explicado en el Capítulo 1 el problema sobre el que trata este trabajo es la programación de trabajos en flujo uniforme, los trabajos han de pasar por las *m* máquinas en un determinado orden, que es común a todos los trabajos, con el objetivo de minimizar el *makespan*, es decir, el instante en el que se finaliza el procesado del último trabajo, en el que los trabajos tienen fechas de comienzo distintas de cero. O en forma reducida y según la notación anteriormente expuesta, Secciones 2.4 y 2.5, $Fm \mid r_j;\ prmu \mid C_{max}$. La solución obtenida se define por la secuencia que indica el orden óptimo en el que han de ejecutarse los trabajos y por el tiempo necesario para finalizar el procesado de estos, *makespan*.

A lo largo del Capítulo 3, además de explicar los distintos métodos a comparar en este Trabajo de Fin de Grado, se definirán formalmente los procedimientos utilizados por cada uno de estos métodos al igual que el modo en el que en cada uno de ellos se calcula el objetivo, minimización del *makespan*, dedicando la Sección 3.7 a detallar las modificaciones realizadas a las ecuaciones propuestas por Taillard, Sección 3.2, para calcular el *makespan* en problemas con fechas de comienzo.

Este mismo problema, aunque sin considerar fechas de comienzo, es sin lugar a dudas uno de los más estudiados en la Investigación Operativa, en gran parte debido a sus similitudes con los problemas que pueden aparecer en un entorno industrial real, que no hacen sino aumentar al considerar las fechas de comienzo de cada uno de los trabajos. El hecho de que el objetivo sea minimizar el *makespan* implica que se maximiza la utilidad de las máquinas y por tanto se minimizan los costes unitarios fijos. Además, la distribución más habitual de las máquinas es de tipo *flowshop* y en muchos de los entornos *jobshop* el recorrido de la mayoría de los trabajos es del tipo *flowshop*. Por último, es muy habitual que no todos los trabajos están disponibles para ser procesados desde un primer momento por diversas razones, como puede ser la necesidad de productos semiterminados o materias primas.

Si bien es cierto que la existencia de las anteriores condiciones es necesaria para que el problema sea de gran utilidad en entornos industriales reales, estas hacen que el problema sea de tipo *NP – hard*, como se explicó anteriormente, y por tanto inviable para un determinado número de trabajos y máquinas no demasiado grandes, encontrar las soluciones exactas a estos problemas. Por ello es de gran interés el desarrollo de heurísticas que permitan obtener soluciones aproximadas a estos problemas en tiempos razonables, posibilitando su uso en entornos reales y reduciendo los costes necesarios para aplicar este tipo de soluciones.

Un tipo habitual de problemas reales con características muy similares o idénticas al problema $Fm \mid r_j;\ prmu \mid C_{max}$ es la fabricación de productos electrónicos, en las que este recorre distintas máquinas, o puestos de trabajo, en los que se van ensamblando los distintos componentes, los cuales han de ser montados en un orden predeterminado. Dado que no es posible comenzar el montaje hasta que no se dispone de los componentes a integrar en la primera máquina, como puede ser el chasis del producto, se han de considerar unas fechas de comienzo impuestas por las fechas de llegada de los componentes iniciales.

3 DESCRIPCIÓN DE LOS ALGORITMOS

En este capítulo se expondrán los distintos algoritmos que se van a comparar en la resolución del problema expuesto en el Capítulo 2. Para ello, en los siguientes apartados se realizará una descripción de los pasos que realizan estos algoritmos para obtener las soluciones de los problemas, se señalarán las características por las que estos destacan y se describirá sucintamente los conceptos en los que están basados. Por último se adaptarán las ecuaciones de la aceleración de Taillard a los problemas con fechas de comienzo.

3.1 NEH

Esta heurística, diseñada en 1982 por Muhammad Nawaz, E. Emory Enscore Jr. y Inyong Ham (Nawaz, Enscore, Ham, 1983), a los que debe su nombre, es la base de los que a día de hoy se consideran los mejores algoritmos para la resolución de los problemas de programación de la producción en flujo uniforme con el objetivo de minimizar el *makespan* (*Fm* | *prmu* | C_{max}). La complejidad de esta heurística es $O(n^3m)$, ya que la complejidad de evaluar el *makespan* de una determinada secuencia es $O(nm)$ y por tanto la de evaluar cada una de las posiciones que puede ocupar un trabajo en esa secuencia es $O(n^2m)$. Como se explicará posteriormente, en la Sección 3.2, aplicando la aceleración de Taillard la complejidad de evaluar cada una de las posiciones que puede ocupar un trabajo en una secuencia se reduce a $O(nm)$, reduciendo por tanto la complejidad total de la heurística a $O(n^2m)$.

La idea en la que se basa la efectividad de esta heurística es que los trabajos con un mayor tiempo total de procesado deberán realizarse antes que aquellos con menores tiempos totales de procesado. Con el fin de aprovechar esta idea el algoritmo ha de seguir los siguientes pasos:

- Primero: Calcular el tiempo total de procesado de cada uno de los trabajos, siendo este $T_j = \sum_{i=1}^{m} t_{i,j}$.
- Segundo: Ordenar los trabajos en orden descendente según su tiempo total de procesado.
- Tercero: Seleccionar los dos trabajos con mayor tiempo total de procesado y calcular la secuencia óptima para estos dos trabajos según su *makespan*, la posición relativa de estos trabajos se mantendrá fija. Asignar a *i* el valor 3, $i = 3$.

- Cuarto: Seleccionar el trabajo que ocupa la j-ésima posición de la lista obtenida en el segundo paso y calcular la secuencia óptima de insertar este trabajo en la secuencia parcial obtenida en el paso anterior, sin alterar el orden relativo de los trabajos que ya forman parte de la secuencia parcial.
- Quinto: Mientras que *i* sea menor que *n* calcular *i* como $i = i + 1$ y volver al cuarto paso.

El hecho de que esta heurística sea una de las mejores para la resolución del problema sin fechas de comienzo (*Fm* | *prmu* | C_{max}) permite asumir que los resultados de adaptar esta heurística al problema con fechas de comienzo (*Fm* | r_j; *prmu* | C_{max}) han de ser favorables.

3.2 Taillard

Como se ha mencionado anteriormente, Sección 3.1, una de las causas de la eficiencia de los algoritmos basados en la heurística NEH es el uso de la aceleración de Taillard, desarrollada por Éric D. Taillard en 1990 (Taillard, 1990). Esta se basa en aprovechar el hecho de que al insertar un trabajo en una secuencia parcial se ha de calcular el *makespan* que se obtiene en cada una de las distintas combinaciones, por lo tanto, facilita el cálculo de este usando unas variables comunes a todas las posiciones de inserción en esa secuencia parcial. Las ecuaciones que se utilizan para calcular estas variables auxiliares y finalmente el *makespan* al insertar el k-ésimo trabajo son las siguientes:

- e_{ij} (*Earliest completion time*):
  $e_{0,j} = 0, e_{i,0} = 0 \quad e_{i,j} = max\left(e_{i,j-1}, e_{i-1,j}\right) + t_{i,j}$ *(i = 1, ..., m) (j = 1, ..., k - 1)*

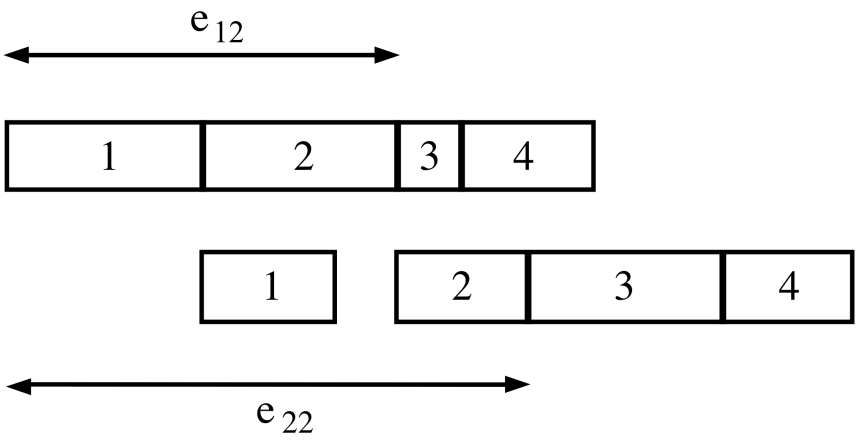


Figura 4. *Earliest completion time* del trabajo 2 en esta secuencia parcial. [Fuente: Taillard E., 1990]

- q_{ij} (*Tail*):

  $q_{i,k} = 0, e_{m+1,j} = 0$

  $q_{i,j} = max(q_{i+1,j}, q_{i,j+1}) + t_{i,j}$ *(i = m, ..., 1) (j = k - 1, ..., 1)*

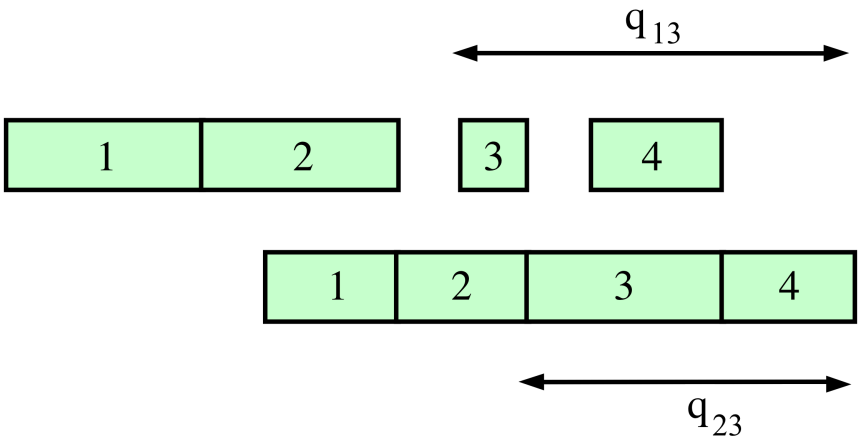


Figura 5. *Tail* del trabajo 3 en esta secuencia parcial. [Fuente: Taillard E., 1990]

- f_{ij} (*Completion time*):

  $f_{0,i} = 0$ $f_{i,j} = max(f_{i-1,j}, e_{i,j-1}) + t_{i,k}$ *(i = 1, ..., m) (j = 1, ..., k)*

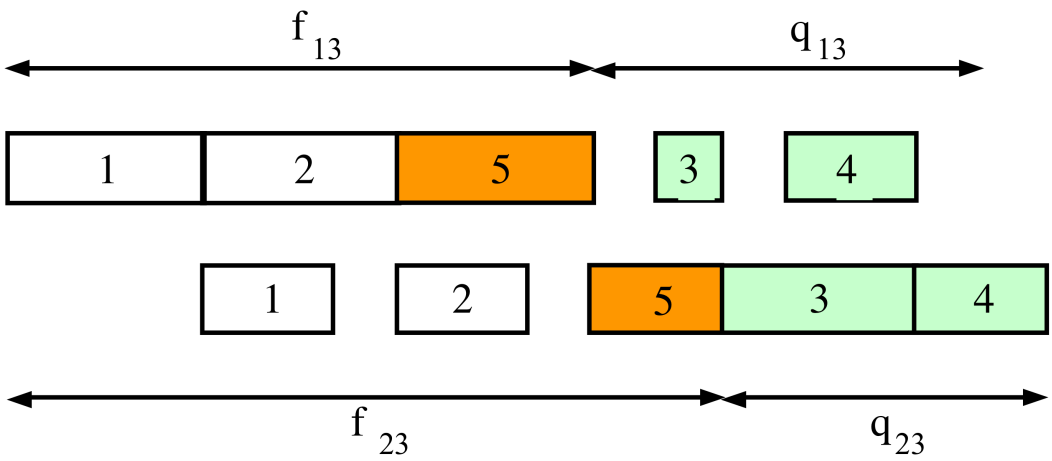


Figura 6. *Completion time* del trabajo 3 y *makespan* de esta secuencia. [Fuente: Taillard E., 1990]

- $C_{max}(j)$ (*Makespan*):

  $C_{max}(j) = \max_i(f_{i,j} + q_{i,j})$ *(i = 1, ..., m) (j = 1, ..., k)*

Para las pruebas que se han realizado con el fin de comparar los distintos algoritmos se han usado un conjunto de problemas, también desarrollados por Éric Taillard (Taillard, 1993), que destacan por ser unos de los mas usados en las pruebas de algoritmos para la resolución de este tipo de problemas. Posteriormente a estos se les han añadido fechas de comienzo, como se explicará en el siguiente capítulo, Capítulo 4.

3.3 IG_{RIS}

Este algoritmo, desarrollado en 2008 por Quan-Ke Pan, Mehmet Fatih Tasgetiren y Yun-Chia Liang (Pan, Tasgetiren, Liang, 2008), destaca por ser uno de los más eficientes en la resolución de problemas de programación de la producción en flujo uniforme con el objetivo de minimizar el *makespan* (*Fm* | *prmu* | C_{max}). Este método está basado en IG_RS_{LS}, desarrollado por Rubén Ruiz y Thomas Stützle (Ruiz, Stützle, 2007), al que se mejoró la fase de búsqueda local mediante el uso de *referenced insertion scheme*, técnica que se explicará posteriormente. Ambos a su vez usan como base un algoritmo *iterated greedy*, voraz iterativo, que empieza generando una solución inicial, mediante la heurística NEH mencionada anteriormente, para después realizar una búsqueda local y a continuación iniciar un bucle en el que pasa por una fase de destrucción, en la que se eliminan *d* trabajos de la solución obtenida, una fase de construcción, en la que los *d* trabajos previamente retirados vuelven a ser insertados en la

secuencia, y otra fase de búsqueda local para por último aplicar un criterio de aceptación. Como criterio de finalización de este bucle se usa un límite de tiempo proporcional al número de máquinas y trabajos del problema, $n \cdot \left(\frac{m}{2}\right) \cdot t_s$, siendo t_s un parámetro del algoritmo. A continuación, Figura 7, se muestra un esquema general del algoritmo.

```
procedure Iterated_Greedy
   π0 := GenerateInitialSolution;
   π  := LocalSearch(π0)                %optional
   while (Not Termination)
      π^D := Destruction(π)
      π*  := Construction(π^D, π^R)
      π*  := LocalSearch(π*)            %optional
      π   := AcceptanceCriterion(π, π*)
   endwhile
endprocedure
```

Figura 7. Esquema algoritmo *iterated greedy*. [Fuente: Pan Q-K, Tasgetiren M, Liang Y-C., 2008]

Parte clave de estos algoritmos son las fases de destrucción y construcción. En la fase de destrucción *d* trabajos aleatoriamente seleccionados se retiran de la solución π. En la siguiente fase, construcción, cada uno de estos trabajos se reinserta en la secuencia obtenida mediante el procedimiento de la heurística NEH.

```
Procedure DC(π)
   for i:=1 to d do
      πD=remove one job from at random from π and insert it in πR
   endfor
   for i:=1 to d do
      π=best permutation obtained by inserting job πR(i) in all possible positions of πD.
   endfor
   return π
```

Figura 8. Esquema de las fases de destrucción y construcción. [Fuente: Pan Q-K, Tasgetiren M, Liang Y-C., 2008]

El procedimiento *referenced insertion scheme* consiste en encontrar para cada uno de los trabajos que forman una secuencia su posición óptima dentro de esta secuencia y si esta es distinta de la posición que anteriormente ocupaban reiniciar el procedimiento con la nueva

secuencia. El proceso finaliza una vez se han comprobado, con éxito, todos los trabajos. A continuación, Figura 9, se muestra un diagrama de este procedimiento.

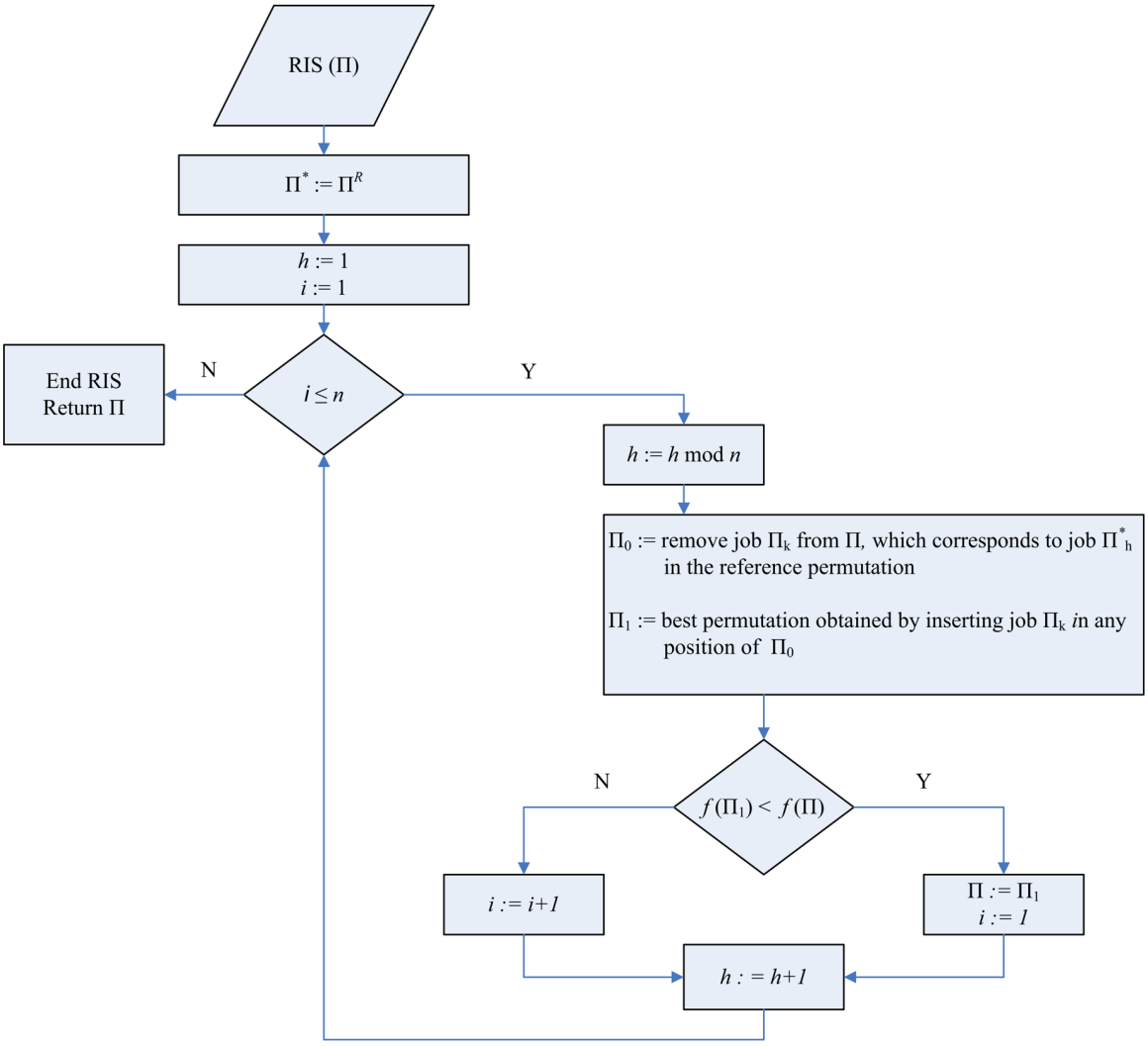


Figura 9. *Referenced insertion scheme*. [Fuente: Pan Q-K, Tasgetiren M, Liang Y-C., 2008]

Con el fin de evitar un estancamiento de las soluciones este algoritmo incluye un criterio de aceptación del tipo recocido simulado, para el cual se define la siguiente temperatura, donde τ es un parámetro: $T = \frac{\sum_{j=1}^{n}\sum_{k=1}^{m} p_{k,j}}{n \cdot m \cdot 10} \cdot \tau$. En el siguiente diagrama, Figura 10, se muestra el funcionamiento del algoritmo IG_{RIS}.

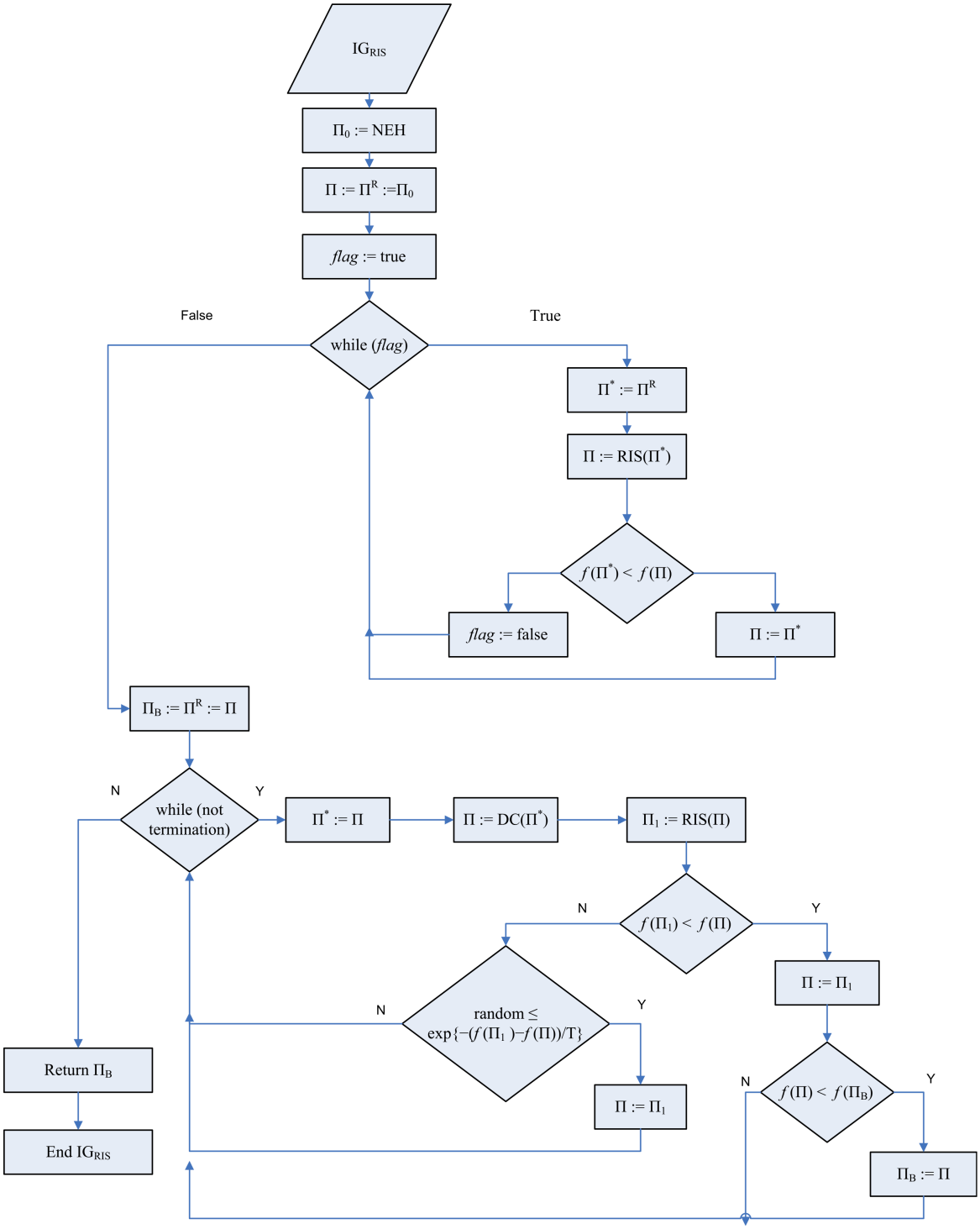


Figura 10. Diagrama del algoritmo *IG*$_{RIS}$. [Fuente: Pan Q-K, Tasgetiren M, Liang Y-C., 2008]

El hecho de que esta heurística, al igual que NEH, sea una de las mejores para la resolución del problema sin fechas de comienzo (*Fm* | *prmu* | C_{max}) permite asumir que los resultados de adaptar esta heurística al problema con fechas de comienzo (*Fm* | r_j; *prmu* | C_{max}) han de ser favorables.

3.4 *TB$_{FF}$*

Los procedimientos anteriormente expuestos provocan un gran número de empates entre secuencias, al insertar un trabajo en una secuencia se obtienen varias posiciones con idéntico *makespan*, y es sabido que la forma de romper estos empates influye significativamente en el rendimiento de estos algoritmos. También existen mecanismos para los casos de empate en el orden inicial de los trabajos, pero estos no son objeto de estudio en este trabajo. Aunque en la literatura existen diversos mecanismos de *tie-breaking*, como los propuestos por Xingye Dong, Houkuan Huang y Ping Chen (Dong, Huang, Chen, 2008) basado en equilibrar la carga de trabajo de las distintas máquinas y Pawel Jan Kalczynski y Jerzy Kamburowski (Kalczynski, Kamburowski, 2008) y (Kalczynski, Kamburowski, 20011), el propuesto por Víctor Fernández – Viagas y José M. Framiñan (Fernández – Viagas, Framiñán, 2014) (*TB$_{FF}$*) es el que obtiene los mejores resultados. Entre estos mecanismos no se ha considerado el propuesto por Imma Ribas, Ramón Companys y Xavier Tort – Martorell (Ribas, Companys, Tort – Martorell, 2010) debido a que su complejidad computacional es *O(n^{2}m)* o *O(nm^2)*, en función de si se aplica la aceleración de Taillard o no, frente a la de los anteriores mecanismos que es *O(nm)*.

El mecanismo *TB$_{FF}$* basa su funcionamiento en la reducción de los *idle time*, los cuales, aunque pueden definirse de las siguientes formas: considerando los tiempos anteriores al primer trabajo y posteriores al último, descontando ambos o descontando solo los posteriores al último trabajo, se opta por aplicar la última de estas definiciones. Por lo tanto, los tiempos ociosos de la i-ésima máquina se calcularán como $it_i = C_{i,n} - \sum_{j=1}^{n} p_{i,j}$ y el tiempo ocioso total como $it = \sum_{i=1}^{m} it_i$. De esta forma el incremento del tiempo ocioso total que provoca insertar un nuevo trabajo en una secuencia se calcula como $\Delta_{i,j} = \left(e_{i,j+1} - p_{i,j+1}\right) - e_{i,j}$, siendo por tanto $it = \sum_{i=1}^{m} \sum_{j=0}^{n-1} \Delta_{i,j}$. Para que el cálculo del tiempo ocioso total no incremente la complejidad del algoritmo este se calcula usando una estimación $it'(l) = \sum_{i=1}^{m} \left(g_{i,l-1} + h_{i,l} + \Delta'_{i,l-1} + \Delta'_{i,l}\right)$, siendo $g_{i,\,l-1}$ la suma de los tiempos ociosos provocados en la máquina *i* por los trabajos anteriores a la posición *l – 1*, $h_{i,\,l}$ es análogamente la suma de los tiempos ociosos provocados en la máquina *i* por los trabajos posteriores a la posición *l* y $\Delta'_{i,l-1}$ y $\Delta'_{i,l}$ son los tiempos ociosos producidos al insertar el nuevo trabajo en la posición *l*. Estas variables se calculan de la siguiente forma, siendo *r* el trabajo a insertar en la secuencia:

$$g_{i,l-1} = \sum_{j=0}^{l-2} \Delta_{i,j} = \sum_{j=1}^{l-1} \left[\left(e_{i,j} - p_{i,j}\right) - e_{i,j-1}\right]$$

$$h_{i,l} = \sum_{j=l}^{k-2} \Delta_{i,j} = \sum_{j=l}^{k-2} \left[\left(e_{i,j+1} - p_{i,j+1}\right) - e_{i,j}\right]$$

$$\Delta'_{i,l-1} = \left(f_{i,l} - t_{i,r}\right) - e_{i,l-1}$$

$\Delta'_{i,l} = max\{f'_{i-1,l} - f_{i,l}, 0\}$

$f'_{0,l} = 0, \qquad f'_{i,l} = max\{f_{i,l}, f'_{i-1,l}\} + p_{i,l}, \qquad i = 1, ..., m$

Siendo por tanto

$it'(l) = C + it''(l) = \sum_{i=1}^{m}(g_{i,l-1} + h_{i,l} + \Delta'_{i,l-1}) + \sum_{i=1}^{m}(f_{i,l} - e_{i,l} + p_{i,l} - t_{i,r} + \Delta'_{i,l})$

Quedando, ya que $\sum_{i=1}^{m} t_{i,r}$ es constante, la estimación del tiempo ocioso total como

$it''(l) = \sum_{i=1}^{m}(f_{i,l} - e_{i,l} + p_{i,l} + max\{f'_{i-1,l} - f_{i,l}, 0\})$

A continuación, Figura 11, se muestra el pseudocódigo de este mecanismo aplicado al NEH.

$\pi' \longleftarrow$ Sort in decreasing order of sum of processing times p_{ij};
$\pi \longleftarrow \pi'_1$;
for $k = 2$ **to** n **do**
  $r \longleftarrow \pi_k'$;
  Determine the values of e_{ij}, q_{ij} and f_{il} from Taillard's acceleration (see equations 2, 3, and 4);
  Determine minimal makespan resulting from inserting job r in all possible positions of π;
  $bp \longleftarrow$First position where the makespan is minimal;
  $tb \longleftarrow$ Number of positions with minimal makespan (i.e. number of ties);
  $ptb \longleftarrow$ Array (of length tb) with the positions where the makespan is minimal;
  it_{bp} is the idletime corresponding to the bp and set to a very large number;
  if $tb > 1$ **and** $k < n$ **then**
    for $l = 1$ **to** tb **do**
      $it'' \longleftarrow 0$;
      if $ptb[l] = k$ **then**
        for $i = 2$ **to** m **do**
          $it'' \longleftarrow it'' + f_{i,k} - e_{i,k-1} - t_{i,r}$;
        end
      else
        $f'_{1,ptb[l]} \longleftarrow f_{1,ptb[l]} + p_{1,ptb[l]}$;
        for $i = 2$ **to** m **do**
          $it'' \longleftarrow it'' + f_{i,ptb[l]} - e_{i,ptb[l]} + p_{i,ptb[l]} - t_{i,r} + \max\{0, f'_{i-1,ptb[l]} - f_{i,ptb[l]}\}$;
          $f'_{i,ptb[l]} \longleftarrow \max\{f'_{i-1,ptb[l]}, f_{i,ptb[l]}\} + p_{i,ptb[l]}$;
        end
      end
      if $it_{bp} > it''$ **then**
        $bp \longleftarrow ptb[l]$;
        $it_{bp} \longleftarrow it''$;
      end
    end
  end
  $\pi \longleftarrow$ Array obtained by inserting job r in position bp of π;
end

Figura 11. Pseudocódigo de NEH al aplicar *TB$_{FF}$*. [Fuente: Fernandez-Viagas, V. & Framinan, J.M., 2014]

3.5 *DSJF*

Este algoritmo, desarrollado por Danyu Bai y Lixin Tang (Bai, Tang, 2008), se caracteriza por su rapidez a la hora de resolver los problemas de programación de la producción en flujo uniforme con fechas de comienzo con el objetivo de minimizar el *makespan* (*Fm* | *online-*r_j*; prmu* | C_{max}). La rapidez de este algoritmo se debe a su baja complejidad computacional, provocada por la sencillez del procedimiento, y significativamente inferior a la del resto de métodos considerados en este trabajo, como se podrá ver posteriormente. Pese a su sencillez esta heurística es asintóticamente óptima para un número de trabajos, *n*, suficientemente grande. El funcionamiento de este algoritmo se basa en la idea de que, si no se consideran los tiempos ociosos, el *makespan* de una secuencia es la suma del tiempo total de proceso del primer trabajo, T_j, y la suma de los tiempos de proceso de los trabajos restantes en la última máquina. Por tanto, dado que la suma de los tiempos de proceso de los *n* trabajos en la última máquina es constante, el primer trabajo en ser procesado ha de ser aquel con menor tiempo de procesado en las primeras *m -1* máquinas. Con lo que, basado en esta idea, el funcionamiento de esta heurística es el siguiente:

- Primer paso: Procesar el trabajo disponible con menor tiempo de procesado en las primeras *m -1* máquinas, calculándose este tiempo como $\sum_{i=1}^{m-1} t_{i,j}$. En caso de no encontrar ningún trabajo disponible ir al tercer paso.
- Segundo paso: Cuando se finalice el trabajo en curso de la primera máquina ir al primer paso.
- Tercer paso: Dejar la primera máquina ociosa hasta que algún trabajo esté disponible, después ir al primer paso.
- Cuarto paso: Una vez se han programado todos los trabajos calcular el *makespan* resultante.

El hecho de que en este procedimiento se consideren *online release dates*, es decir no considerar los trabajos hasta que no se alcance su fecha de comienzo, se debe a que, habitualmente, en un problema real se desconocen las características, o incluso la existencia, de un trabajo hasta que este no está disponible para ser procesado.

3.6 *iMGS*

Este algoritmo, desarrollado por Tao Ren, Meiting Guo, Lin Lin y Yunhui Miao (Ren et al., 2015), consiste en una modificación del algoritmo *GS*, propuesto por Teófilo González y Sartaj Sahni, (González, Sahni, 1978), *MGS*, a la cual se le añade una fase de búsqueda local.

El algoritmo *MGS* consiste en lo siguientes pasos:

- Primer paso: Dividir las *m* máquinas en *m – 1* grupos *g*, siendo $g = \{i - 1,\ i\}$ con *i=2, ..., m*.
- Segundo paso: Para cada grupo de máquinas, cuando la máquina *i – 1* quede ociosa, procesar los trabajos disponibles según la regla de Johnson, es decir, procesar primero los trabajos con $p_{1,j} < p_{2,j}$ en orden no decreciente de $p_{1,j}$ y posteriormente procesar los trabajos restantes en orden no creciente de $p_{2,j}$. Si no hay ningún trabajo disponible ir, al tercer paso.
- Tercer paso: Dejar la primera máquina ociosa hasta que algún trabajo esté disponible, después ir al segundo paso.
- Cuarto paso: Una vez se han programado todos los trabajos calcular Z_g, *makespan* de la secuencia obtenida para el grupo *g*, y elegir la secuencia con el menor *makespan*, siendo este $C_{max} = \min_{1 \le g \le m-1} \{Z_g\}$.

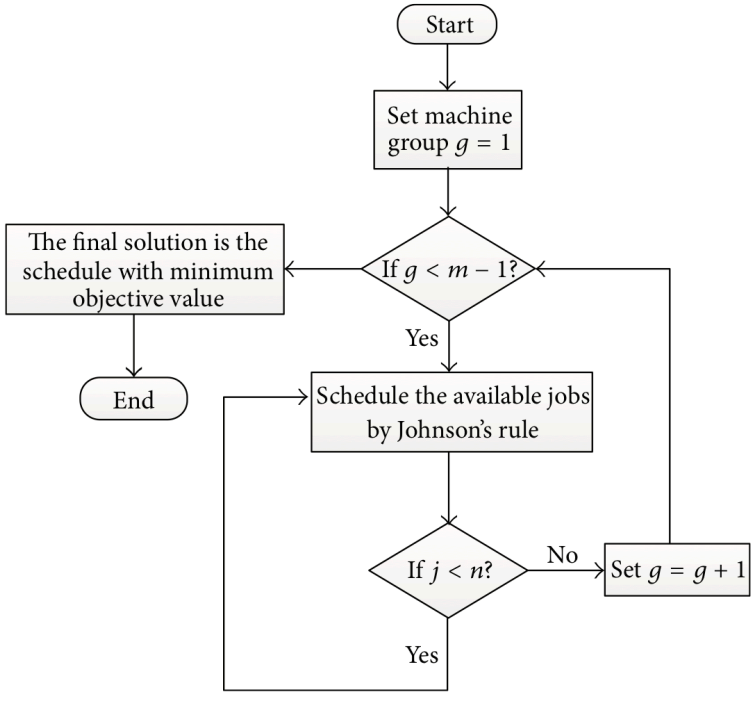


Figura 12. Diagrama del algoritmo *MGS*. [Fuente: Ren, T. et al., 2015]

Por último, el algoritmo *iMGS* finalmente consiste en los siguientes pasos:

- Primer paso: Generar la secuencia inicial usando el algoritmo *MGS* y calcular su *makespan.*
- Segundo paso: Asignar a las variables *g, x* y *k los* siguientes valores: *g = 1, x = 1* y *k = 0.*
- Tercer paso: Comparar el x-ésimo trabajo de la secuencia con el siguiente trabajo, si estos trabajos, *a* y *b*, cumplen las siguientes condiciones
  1. $r_a \leq r_b$,
  2. Una de las siguientes:
     i. $r_b - r_a \leq p^g_{1,a} - p^g_{1,b}$,
     ii. $r_b - r_a \leq p^g_{2,b} - p^g_{1,b}$,
     iii. $r_b - r_a \leq p^g_{1,a} - p^g_{2,a}$,
     iv. $r_b - r_a \leq p^g_{2,b} - p^g_{2,a}$,

  intercambiar la posición de los trabajos *a* y *b* y calcular el *makespan* de la nueva secuencia.
- Cuarto paso: Si el *makespan* de la secuencia obtenida en el tercer paso es menor que $Z_{k,g}$, calcular *k* como *k := k + 1* y encontrar la secuencia $\pi_{k+1,g}$ tal que $Z_{k+1,g} = min\{Z_{y,g} \mid y \in \Pi_{k,g}(x)\}$. Si $x \leq n$, *x := x + 1* y volver al tercer paso, de lo contrario, ir al quinto paso.
- Quinto paso: Si $g \leq m - 1$, *g := g + 1,* y volver al tercer paso.
- Sexto paso: La solución final es la última secuencia obtenida, $\pi_{n,m-1}$.

3.7 Adaptación de aceleración de Taillard

Dado que la aceleración de Taillard es un método diseñado para el cálculo del *makespan* en problemas de programación de la producción en flujo uniforme con el objetivo de minimizar el *makespan* sin fechas de comienzo (*Fm | prmu | C_{max}*) y que el objeto de estudio de este Trabajo de Fin de Grado son aquellos con fechas de comienzo (*Fm | r_j; prmu | C_{max}*), ha sido necesario modificar las ecuaciones utilizadas para calcular las variables auxiliares que determinan el *makespan*.

Al considerar las fechas de comienzo se han de modificar, del modo que se muestra a continuación, las ecuaciones que determinan los *earliest completion times* y *completion times* para la primera máquina:

- e_{ij} (*Earliest completion time*):

  $e_{1,0} = 0$ $\quad$ $e_{1,j} = max(e_{1,j-1}, r_j) + t_{1,j}$ $\quad$ *(j = 1, ..., k - 1)*

- f_{ij} (*Completion time*):

  $f_{1,j} = max(e_{i,j-1}, r_j) + t_{1,k}$ $\quad$ *(j = 1, ..., k)*

Una vez realizadas estas modificaciones es posible continuar usando la aceleración de Taillard para calcular el *makespan*, sin incrementar la complejidad computacional de los métodos expuestos en este capítulo.

4 APLICACIÓN DE LOS MÉTODOS

En este capítulo se explicará y justificará brevemente el *software* utilizado, posteriormente se presentará el conjunto de problemas con el que se probarán y compararán los algoritmos, se asignará y justificará el valor de los parámetros y se describirán los indicadores con los que se compararán los distintos algoritmos.

Dado que los problemas objeto de estudio de este trabajo incluyen fechas de comienzo y algunos de los métodos anteriormente mencionados no están diseñados para la resolución de problemas con fecha de comienzo, estos tuvieron que ser modificados, Sección 3.7, de modo que se considerarán las restricciones que imponen las fechas de comienzo. Por ello, tras incorporar el mecanismo TB_{FF} al algoritmo IG_{RIS}, obteniendo por tanto el algoritmo $IG_{RIS(TB_{FF})}$, este se modificó para adaptarlo al problema con fechas de comienzo. Con respecto a la heurística *NEH*, y tras incluir en esta la aceleración de Taillard y el mecanismo TB_{FF}, también se modificó para adaptarla al problema con fechas de comienzo, se hará referencia a este algoritmo como $NEH(TB_{FF})$.

4.1 Software

Los algoritmos se han implementado en C++, este lenguaje de programación fue diseñado por Bjarne Stroustrup en 1979 como una evolución de C, debido, además de a su popularidad y lo extendido de su uso, al hecho de que existen multitud de bibliotecas, que permiten simplificar la programación de los algoritmos y aportan funciones y rapidez en la ejecución de estos. El código utilizado para generar los problemas, que se explicarán en posteriores apartados, con los que probar los algoritmos está implementado en C.

La elección del programa *Code::Blocks* se debe a que en el mismo programa se integran el editor, el compilador y el depurador de código. Otra de las ventajas de este programa es el hecho de que sea multiplataforma, es decir, compatible con múltiples sistemas operativos.

La elección del programa *Microsoft Excel* para el análisis y procesado de los datos se debe a la flexibilidad de este a la hora de importar datos en distintos formatos, la simplicidad y gran abanico de posibilidades que ofrece a la hora de procesar los resultados obtenidos y por último a su gran utilidad a la hora de generar distintos tipos de tablas y gráficos con los que poder

mostrar los resultados obtenidos por cada uno de los algoritmos a comparar y las conclusiones obtenidas en la realización de este Trabajo de Fin de Grado.

4.2 Generación de los problemas

El rendimiento de los distintos algoritmos se ha comparado usando la batería de 120 problemas, anteriormente mencionada en la Sección 3.2, propuesta por Taillard (Taillard, 1993) y disponibles en la siguiente página web: http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html. Esta la conforman 12 grupos de 10 problemas con los siguientes números de trabajos y máquinas respectivamente: 20 y 5, 20 y 10, 20 y 20, 50 y 5, 50 y 10, 50 y 20, 100 y 5, 100 y 10, 100 y 20, 200 y 10, 200 y 20 y 500 y 20. Los tiempos de proceso para cada uno de los trabajos de esta batería de problemas han sido generados según una distribución uniforme en el intervalo [1, 99].

Dado que esta batería de problemas fue diseñada para problemas sin fecha de comienzo, se han generado 3 subproblemas para cada uno de los problemas anteriores. Estos subproblemas comparten los mismos tiempos de proceso que el problema del que parten, pero sus fechas de comienzo tienen características distintas. Estas fechas de comienzo se han generado, igual que las utilizadas por Danyu Bai y Lixin Tang (Bai, Tang, 2008), según una distribución uniforme en el intervalo [1, $R_t \cdot n$], siendo R_t un parámetro que regula la dispersión de las fechas de comienzo y que toma los valores de 0.5, 1 y 5 para cada uno de los 3 subproblemas que se generan a partir de cada problema.

A cada uno de estos problemas se les denotará, según sus características, como: $taNUM_{n,m,Rt}$, siendo *NUM* un número de 3 cifras asignado originalmente por Taillard.

4.3 Parámetros de los algoritmos

El funcionamiento del algoritmo $IG_{RIS}(TB_{FF})$ estará caracterizado por sus parámetros, d, τ y t_s. Rubén Ruiz y Thomas Stützle (Ruiz, Stützle, 2007), anteriormente mencionados en la Sección 3.3, realizaron un análisis factorial completo de los parámetros d y τ en el que determinaron que sus valores óptimos al combinarlos son $d = 4$ y $\tau = 0.4$, por lo que estos fueron los valores escogidos, aunque en este análisis concluyeron que también existen otras combinaciones de valores con resultados estadísticamente similares. El parámetro que determina el tiempo de parada, t_s, se fijó en 60 milisegundos, también el mismo que el propuesto por Ruiz y Stützle (Ruiz,

Stützle, 2007), pese a esto, los resultados no son comparables debido al uso de ordenadores con distintas características.

Como se ha mencionado anteriormente otro de los parámetros que influyen de manera determinante en el funcionamiento de los algoritmos con tiempos de finalización son las características de los ordenadores en los que se realizan las pruebas. Estas, aunque no afectan a la calidad de las soluciones obtenidas por el resto de los algoritmos, también influyen de manera determinante en los tiempos ejecución necesarios para obtener esas soluciones, de modo que, sí afectan finalmente al rendimiento de los algoritmos, si se considera este como una relación entre la calidad de las soluciones y el tiempo de ejecución necesario para obtener estas soluciones. Los resultados que posteriormente se mostraran han sido obtenidos con un ordenador con las siguientes características:

- Ordenador: MacBook Pro (15 pulgadas, mediados de 2012).
- Sistema operativo: macOS Sierra.
- Procesador: Intel Core i7-3615QM, 2.3 GHz.
- Memoria RAM: 8 GB 1600 MHz DDR3.

4.4 Indicadores

A la hora de determinar el rendimiento de cada uno de los algoritmos se valorarán sus tiempos de ejecución y la calidad de sus soluciones, para determinar esta última se usará el *Average Relative Percentage Deviation* (*ARPD*). El *ARPD* de cada algoritmo para cada grupo de problemas se calculará como la media de sus Relative *Percentage Deviation* (*RPD*), calculándose estos como $RPD = \left(\frac{Sol - BestSol}{BestSol}\right) \cdot 100$, donde *Sol* hace referencia al valor de la solución cuya calidad se desea medir y *BestSol* al valor de la mejor solución encontrada por el conjunto de métodos a comparar, expuestos en el Capítulo 3. Con respecto al tiempo de ejecución, estos se han medido en segundos y se han calculado las medias de cada algoritmo para cada grupo de problemas.

Con el fin mejorar de la precisión de este análisis y debido a que el algoritmo $IG_{RIS}(TB_{FF})$ no es determinista este se ha ejecutado 5 veces para cada uno de los problemas, de modo que la calidad de sus soluciones para los distintos problemas se ha determinado como la *ARPD* de las 5 soluciones obtenidas para cada uno de los problemas.

5 RESULTADOS

En este capítulo se presentarán los resultados obtenidos y por último estos se analizarán, con el objetivo de determinar si algún método es globalmente superior a los demás o destaca para problemas con unas determinadas características o en los que se buscan tiempos de ejecución extremadamente reducidos, para un determinado tamaño del problema.

5.1 Resultados

Cada uno de los 360 problemas, previamente explicados, se ha resuelto según los distintos métodos expuestos en el Capítulo 3, es decir NEH, *DSJF*, *iMGS*, *NEH(*TB_{FF}*)* y *IG*$_{RIS}$*(*TB_{FF}*)*. Con el fin de permitir una mejor visualización de los resultados, y dado que el interés de este análisis no es sobre un problema concreto sino sobre las características de estos, los resultados se han agregado por grupos de problemas, formados por problemas con las mismas características (número de trabajos, número de máquinas y dispersión de las fechas de comienzo). En las tablas que se muestran a continuación se presentan los resultados obtenidos, *ARPD* (Tabla 1) y tiempos de ejecución (Tabla 2).

n	m	R_t	*NEH*	*DSJF*	*iMGS*	*NEH(TB_{FF})*	*$IG_{RIS}(TB_{FF})$*
20	5	0,5	3,416	18,892	9,223	2,558	0,008
20	5	1	3,072	19,113	8,949	2,026	0,010
20	5	5	2,931	16,446	7,678	3,196	0,000
20	10	0,5	4,488	25,199	10,745	4,437	0,009
20	10	1	4,731	24,752	10,350	5,019	0,014
20	10	5	4,694	23,616	10,294	4,990	0,006
20	20	0,5	3,896	21,763	7,184	3,132	0,019
20	20	1	4,282	22,482	8,039	3,822	0,039
20	20	5	4,570	20,942	6,933	4,804	0,031
50	5	0,5	1,115	15,200	5,281	0,707	0,000
50	5	1	1,017	15,543	4,177	1,545	0,007
50	5	5	1,359	14,221	4,423	1,423	0,010
50	10	0,5	5,045	25,993	10,006	4,650	0,149
50	10	1	5,377	25,972	9,094	4,946	0,117
50	10	5	5,594	24,813	9,289	5,530	0,103
50	20	0,5	6,148	25,132	10,547	6,067	0,306
50	20	1	5,805	24,153	10,296	6,274	0,325
50	20	5	5,795	23,420	10,479	5,877	0,221
100	5	0,5	0,537	10,051	2,708	0,435	0,026
100	5	1	0,626	9,729	3,247	0,462	0,006
100	5	5	1,677	9,251	3,471	1,284	0,007
100	10	0,5	2,157	18,281	6,134	1,921	0,061
100	10	1	1,986	18,187	5,527	1,853	0,113
100	10	5	2,630	17,199	4,979	2,736	0,145
100	20	0,5	4,922	24,379	8,344	4,010	0,248
100	20	1	4,472	23,390	7,700	4,523	0,177
100	20	5	5,211	22,615	8,463	4,775	0,272
200	10	0,5	1,171	13,710	3,911	1,009	0,103
200	10	1	1,394	13,414	3,277	0,927	0,106
200	10	5	1,540	13,067	3,490	1,393	0,058
200	20	0,5	3,302	19,522	5,784	2,871	0,169
200	20	1	3,269	19,778	5,386	3,239	0,170
200	20	5	3,772	18,847	6,302	3,185	0,206
500	20	0,5	1,669	14,532	3,014	1,295	0,095
500	20	1	1,837	14,372	2,997	1,400	0,118
500	20	5	1,904	13,937	3,663	1,451	0,106
Promedio			3,261	18,942	6,705	3,049	0,099

Tabla 1. *ARPD* de cada método en función de las características del problema.

n	m	R_t	*NEH*	*DSJF*	*iMGS*	*NEH(TB_{FF})*	*$IG_{RIS}(TB_{FF})$*
20	5	0,5	0,0	0,0	0,0	0,0	3,0
20	5	1	0,0	0,0	0,0	0,0	3,0
20	5	5	0,0	0,0	0,0	0,0	3,0
20	10	0,5	0,0	0,0	0,0	0,0	6,0
20	10	1	0,0	0,0	0,0	0,0	6,0
20	10	5	0,0	0,0	0,0	0,0	6,0
20	20	0,5	0,0	0,0	0,0	0,0	12,0
20	20	1	0,0	0,0	0,0	0,0	12,0
20	20	5	0,0	0,0	0,0	0,0	12,0
50	5	0,5	0,0	0,0	0,0	0,0	7,5
50	5	1	0,0	0,0	0,0	0,0	7,5
50	5	5	0,0	0,0	0,0	0,0	7,5
50	10	0,5	0,0	0,0	0,0	0,0	15,0
50	10	1	0,0	0,0	0,0	0,0	15,0
50	10	5	0,0	0,0	0,0	0,0	15,0
50	20	0,5	0,0	0,0	0,0	0,0	30,0
50	20	1	0,0	0,0	0,0	0,0	30,0
50	20	5	0,0	0,0	0,0	0,0	30,0
100	5	0,5	0,0	0,0	0,0	0,0	15,0
100	5	1	0,0	0,0	0,0	0,0	15,0
100	5	5	0,0	0,0	0,0	0,0	15,0
100	10	0,5	0,0	0,0	0,0	0,0	30,0
100	10	1	0,0	0,0	0,0	0,0	30,0
100	10	5	0,0	0,0	0,0	0,0	30,0
100	20	0,5	0,0	0,0	0,1	0,0	60,0
100	20	1	0,0	0,0	0,5	0,0	60,0
100	20	5	0,0	0,0	0,3	0,0	60,0
200	10	0,5	0,0	0,0	0,4	0,0	60,0
200	10	1	0,0	0,0	0,6	0,0	60,0
200	10	5	0,0	0,0	0,1	0,0	60,0
200	20	0,5	0,0	0,0	2,1	0,0	120,0
200	20	1	0,0	0,0	1,2	0,0	120,0
200	20	5	0,0	0,0	0,4	0,0	120,0
500	20	0,5	0,0	0,0	18,8	0,1	300,0
500	20	1	0,0	0,0	10,6	0,1	300,0
500	20	5	0,0	0,0	2,5	0,1	300,0

Tabla 2. Tiempo medio de ejecución, en segundos, de cada método en función de las características del problema.

5.2 Análisis de los resultados

A lo largo de esta sección se realizará un análisis de los resultados obtenidos, atendiendo a la influencia de los distintos factores en estos. En la Figura 13, que se muestra a continuación, se presentan los resultados obtenidos para cada uno de los algoritmos en función de los parámetros característicos de los problemas, *n*, *m* y R_t.

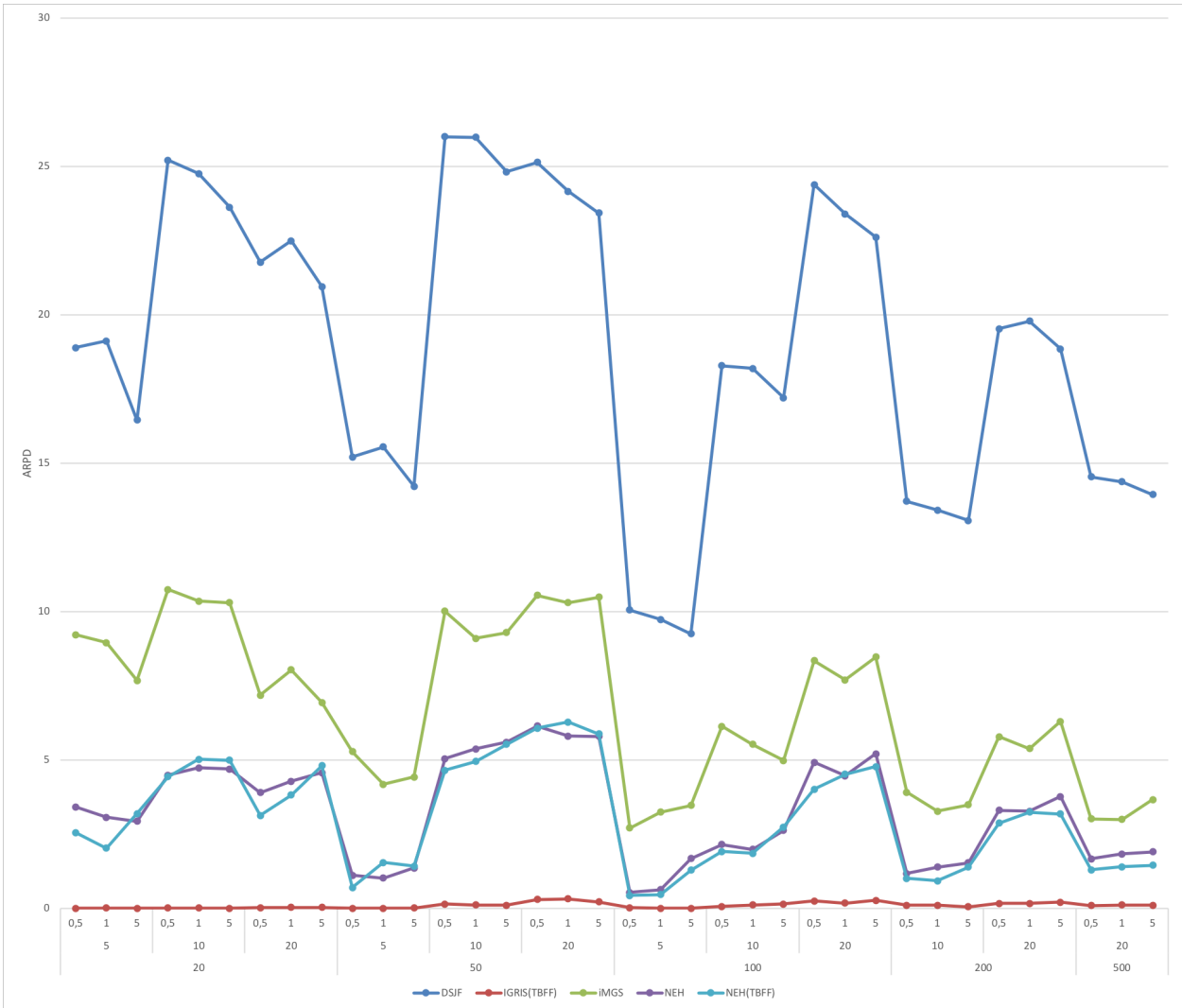


Figura 13. *ARPD* de cada método en función de las características de los problemas.

Como se puede apreciar en esta gráfica, Figura 13, el método $IG_{RIS}(TB_{FF})$ obtiene unos resultados claramente superiores a los del resto de algoritmos. Esta gráfica también muestra que el método con el que se obtienen los peores resultados es el método *DSJF*, seguido por el método *iMGS*. Aunque en esta gráfica, Figura 13, se puede apreciar una ligera superioridad de la calidad de las soluciones obtenidas mediante el método NEH frente a la calidad de las

soluciones obtenidas mediante el método *NEH(TB_{FF})*, esta diferencia no es tan amplia como la que separa los restantes métodos. Por ello se prestará especial atención a ambos métodos, profundizando en el análisis de la calidad de sus soluciones.

En la Tabla 1 y en la Figura 13 se han usado como indicadores de la calidad de las soluciones proporcionadas por los distintos métodos el *ARPD*, por lo que, debido a la naturaleza de este, la dispersión de los resultados obtenidos queda oculta. Para evitar esto, a continuación, Figuras 14-18, se muestran los *RPD* máximos, mínimos y medios (*ARPD*) de cada uno de los algoritmos en función de los parámetros característicos de los problemas, n, m y R_t.

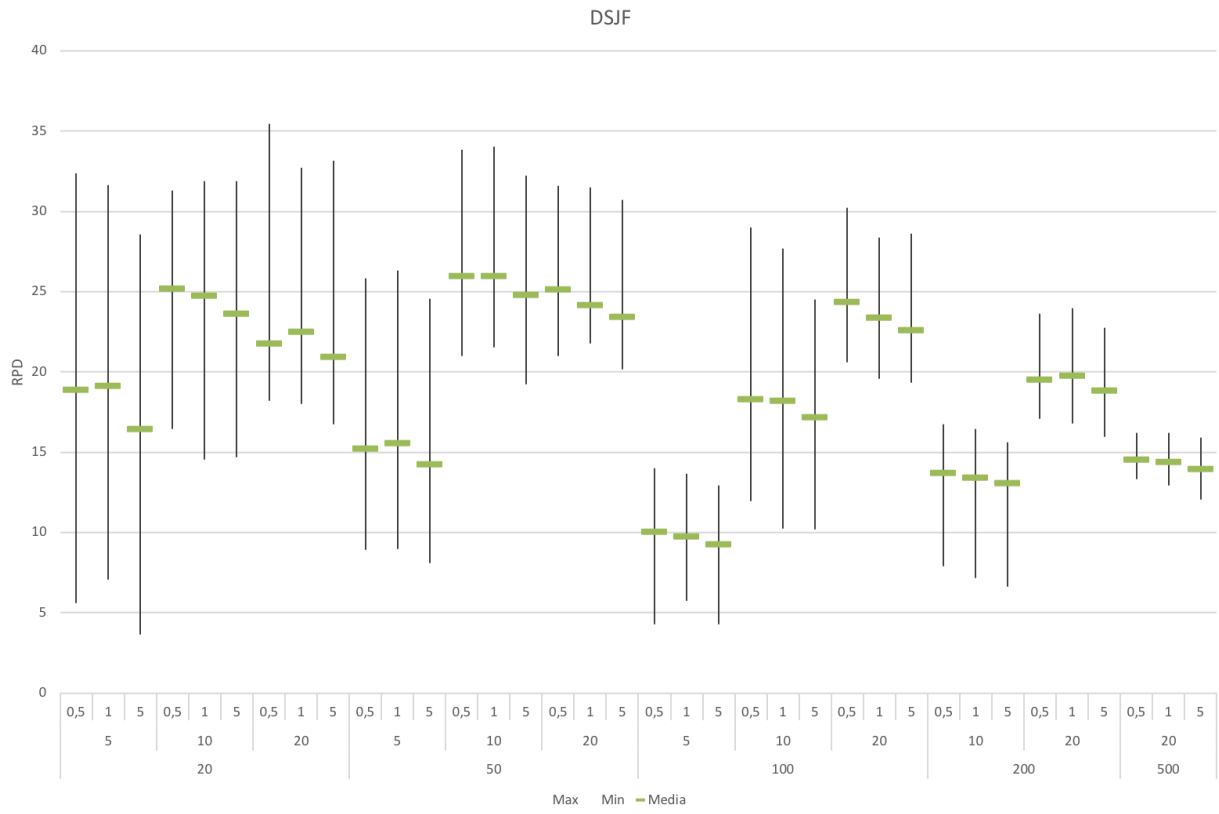


Figura 14. *RPD* del método *DSJF* en función de las características de los problemas.

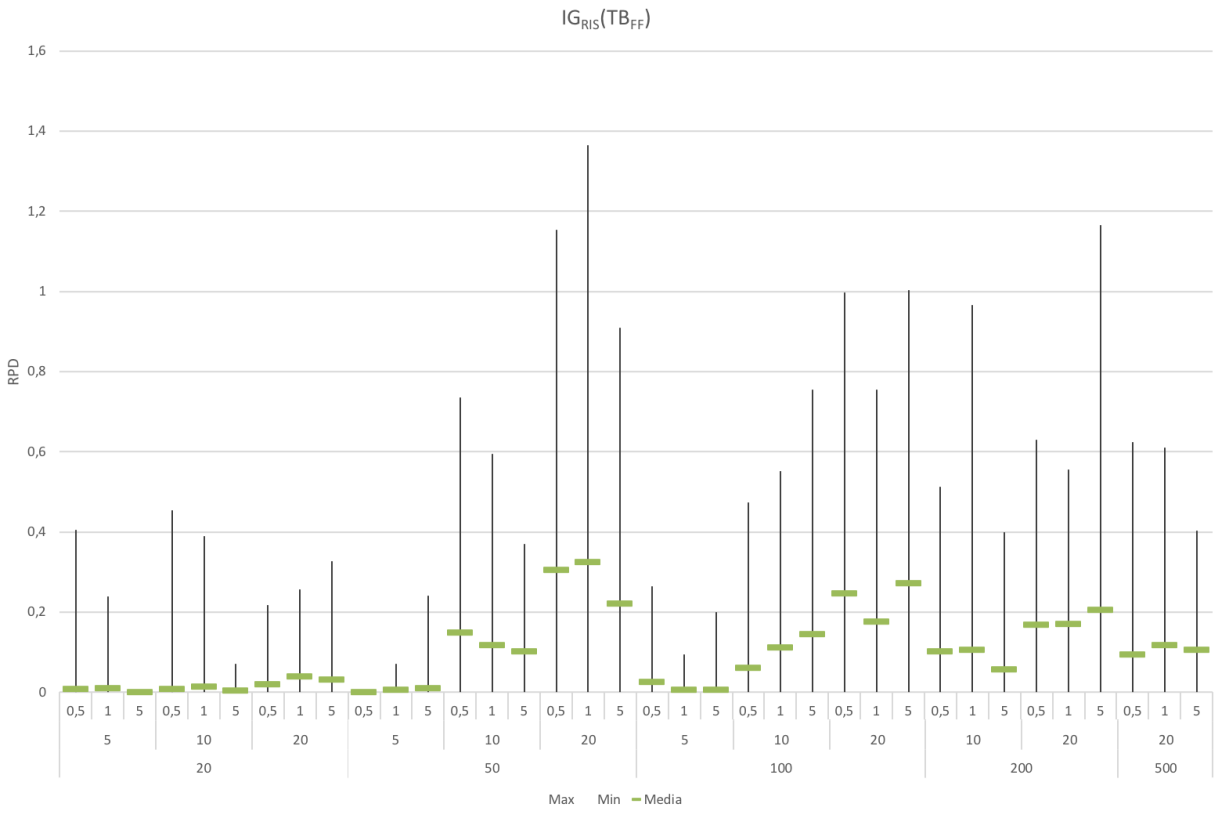


Figura 15. *RPD* del método *$IG_{RIS}(TB_{FF})$* en función de las características de los problemas.

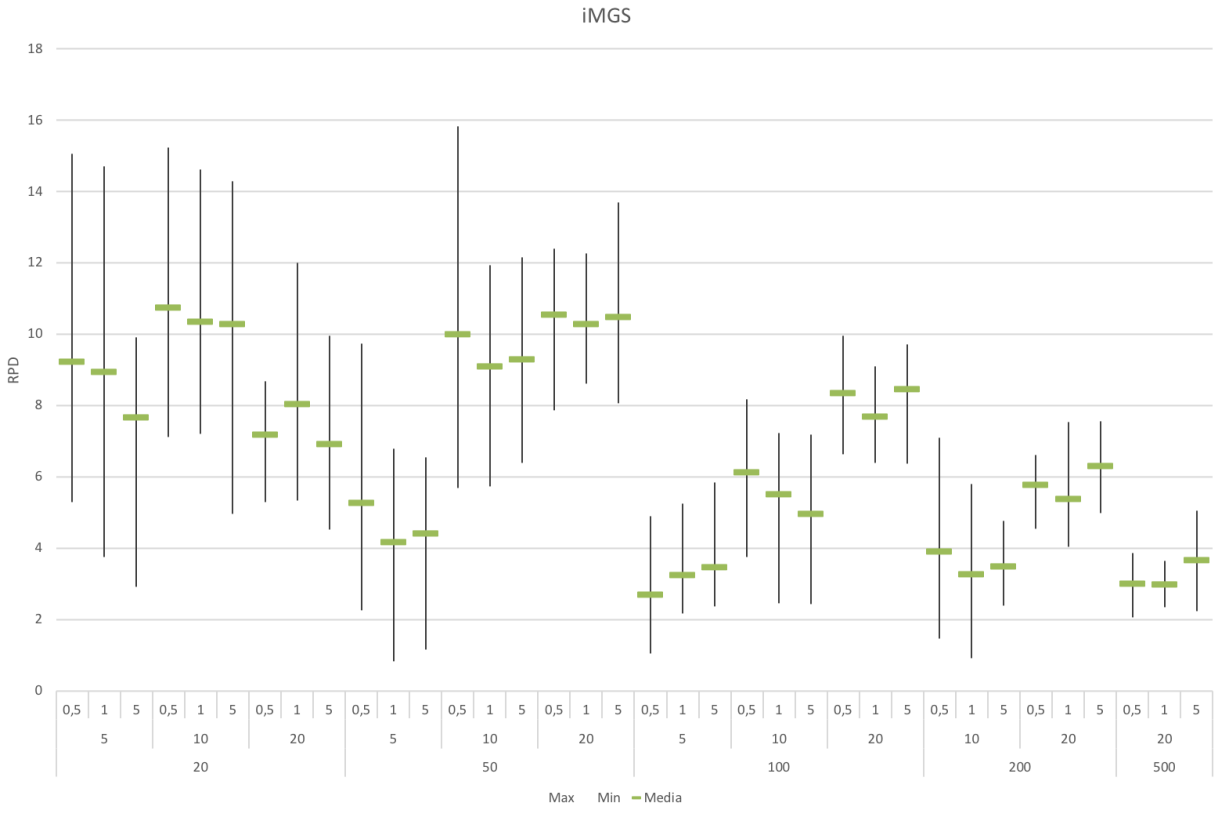


Figura 16. *RPD* del método *iMGS* en función de las características de los problemas.

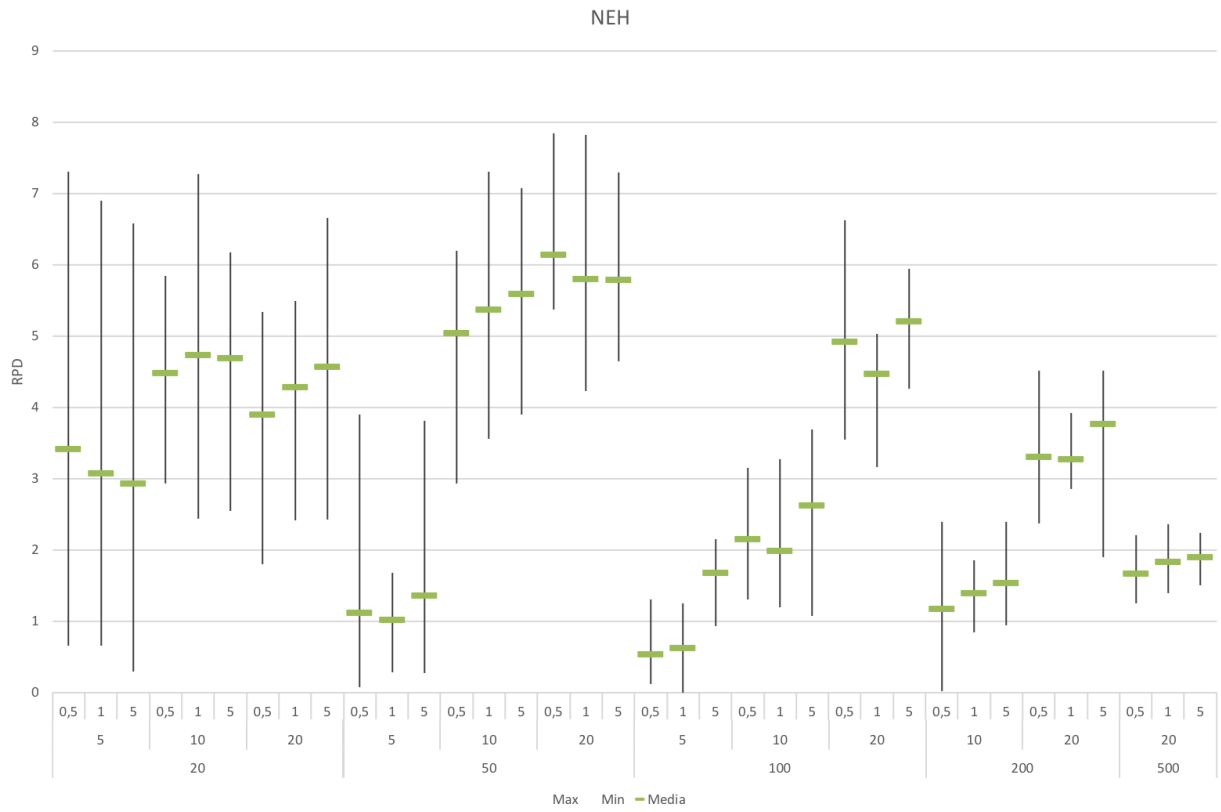


Figura 17. *RPD* del método NEH en función de las características de los problemas.

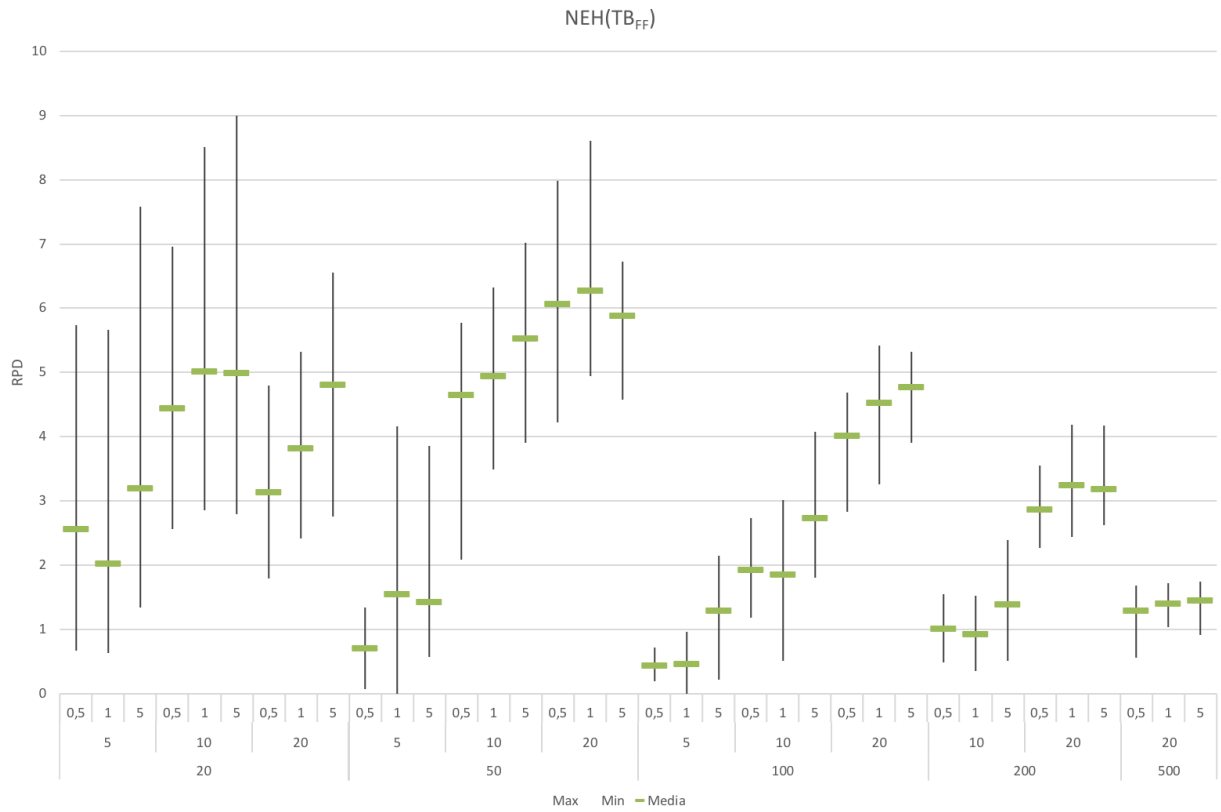


Figura 18. *RPD* del método *NEH(TB$_{FF}$)* en función de las características de los problemas.

Como se puede apreciar en los anteriores gráficos, Figura 14 y Figuras 16-18, existe una gran dispersión en la calidad de los resultados obtenidos por los métodos, *DSJF*, *iMGS*, NEH y *NEH(TB_{FF})*, esta es especialmente significativa para los problemas con un reducido número de trabajos. Por tanto, dado que las conclusiones obtenidas de la Figura 13 no consideran la dispersión de los resultados, a continuación, se comentarán el número de excepciones a estas conclusiones, con el fin de comprobar la validez de estas conclusiones.

- La calidad de las soluciones obtenidas mediante el método *$IG_{RIS}(TB_{FF})$* solo fue superada en una ocasión. Una de las cinco soluciones obtenidas para el problema *$ta094_{200,10,1}$* fue superada por la solución obtenida mediante el método *iMGS*. Por tanto, en el 99,94% de los casos el método *$IG_{RIS}(TB_{FF})$* obtuvo mejores soluciones que el resto de los métodos.
- El método *DSJF* obtuvo peores soluciones que el resto los métodos en el 99,44% de los problemas, es decir en todos salvo en dos. En los problemas *$ta002_{20,5,5}$* y *$ta063_{100,5,5}$* la calidad de las soluciones obtenidas por el método *DSJF* fue superior a la de las obtenidas por el método *iMGS*.
- El método *iMGS*, cuarto mejor de los métodos objeto de estudio de este trabajo según la calidad de sus soluciones, ocupo esta posición para todos los problemas salvo para las excepciones anteriormente explicadas y otros siete casos en los que obtuvo mejores soluciones que los métodos NEH o *NEH(TB_{FF})*.
  I. En los problemas *$ta023_{20,20,5}$*, *$ta029_{20,20,5}$* y *$ta094_{200,10,1}$* la calidad de las soluciones obtenidas mediante este método solo fue superada por la de las obtenidas mediante el método *$IG_{RIS}(TB_{FF})$*.
  II. En los problemas *$ta019_{20,\ 10,\ 5}$* y *$ta075_{100,\ 10,\ 5}$* la calidad de las soluciones obtenidas mediante este método solo fue superada por la de las obtenidas mediante los métodos *$IG_{RIS}(TB_{FF})$* y *NEH(TB_{FF})*.
  III. En los problemas *$ta056_{50,\ 20,\ 0,5}$* y *$ta103_{200,\ 20,\ 1}$* la calidad de las soluciones obtenidas mediante este método solo fue superada por la de las obtenidas mediante los métodos *$IG_{RIS}(TB_{FF})$* y NEH.

  Por tanto, ordenando los métodos estudiados en este trabajo según la calidad de sus soluciones, este método ocupa la segunda posición para el 0,83% de los problemas, la tercera posición para el 1,11% de los problemas, la cuarta posición para el 97,5% de los problemas y la quinta posición para el 0,56% de los problemas.

Estos resultados corroboran el orden, según la calidad de las soluciones, anteriormente inferido según los valores de los *ARPD* de los métodos *$IG_{RIS}(TB_{FF})$*, *iMGS* y *DSJF*.

Con respecto a los métodos NEH o *NEH(TB$_{FF}$)* ya se han expuesto los casos en los que la calidad de sus soluciones es superada por la del método *iMGS*. Debido a las no muy holgadas diferencias entre las soluciones obtenidas mediante estos métodos se va a realizar un análisis más detallado, que permita apreciar con mayor claridad la diferencia de calidad de las soluciones obtenidas por ambos.

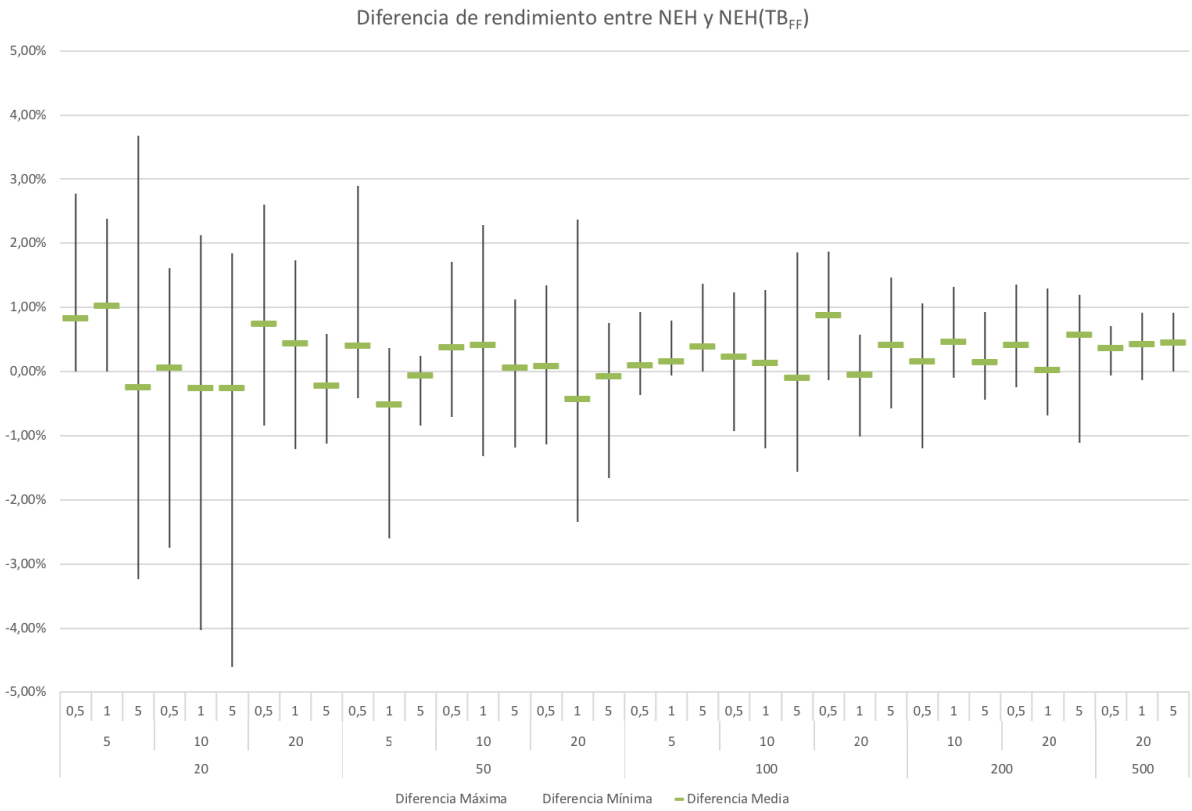


Figura 19. Diferencia de rendimiento máxima, mínima y media entre NEH y *NEH(TB$_{FF}$)* en función de las características de los problemas.

Como indica la Figura 19, pese a que la calidad media de las soluciones obtenidas mediante el método *NEH(TB$_{FF}$)* es superior, en un 0,21% de media, a la de las obtenidas mediante el método NEH, se pueden observar varios casos en los que, para los mismos parámetros del problema, las diferencias máximas y mínimas alcanzan valores superiores al 2%. En las gráficas que se exponen a continuación, Figuras 20-21, se muestran, además de la diferencia media de rendimiento, el porcentaje de problemas en los que cada uno de estos dos métodos obtiene una mejor solución que el otro, Figura 20, y la diferencia entre estos porcentajes, Figura 21.

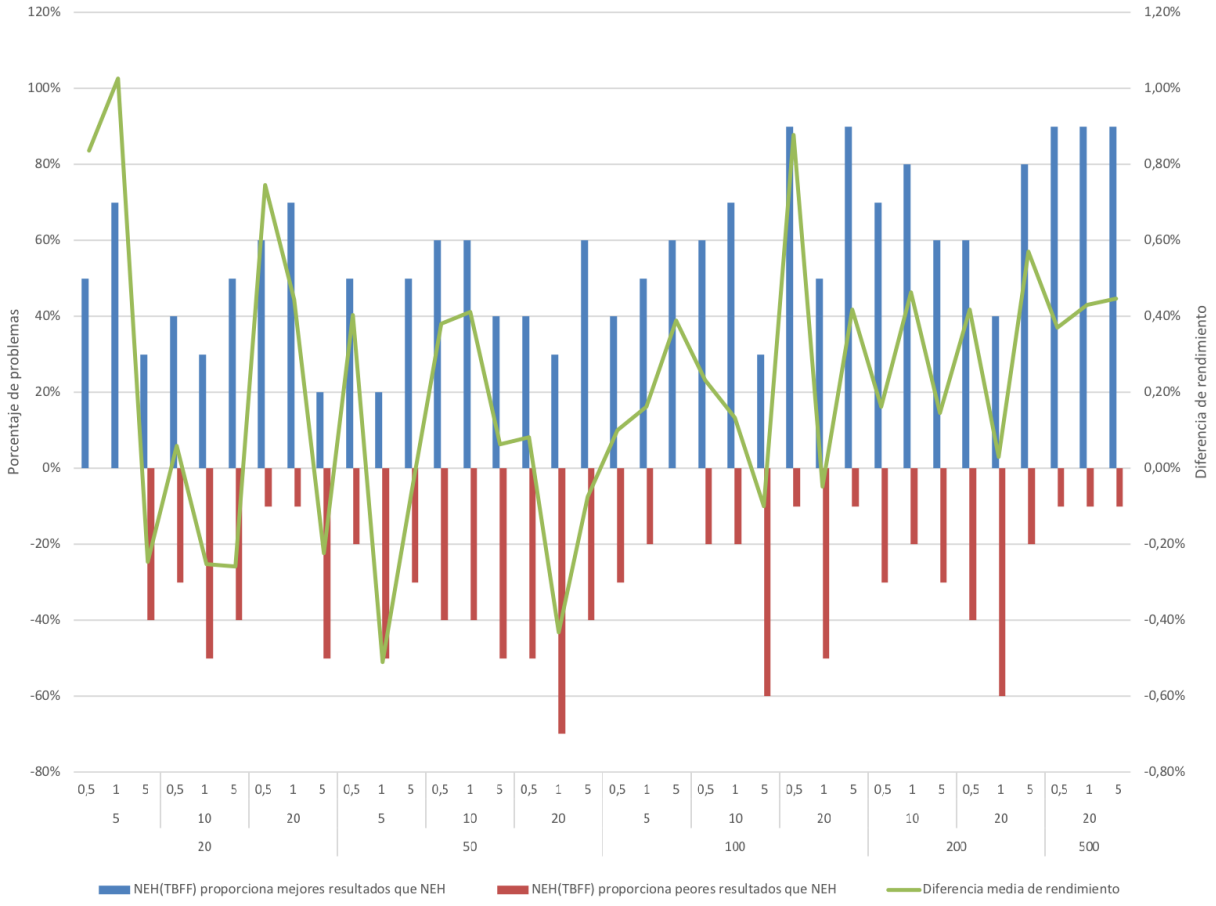


Figura 20. Diferencia de rendimiento media entre NEH y *NEH(TB_{FF})* y porcentaje de problemas en los que cada método es superior.

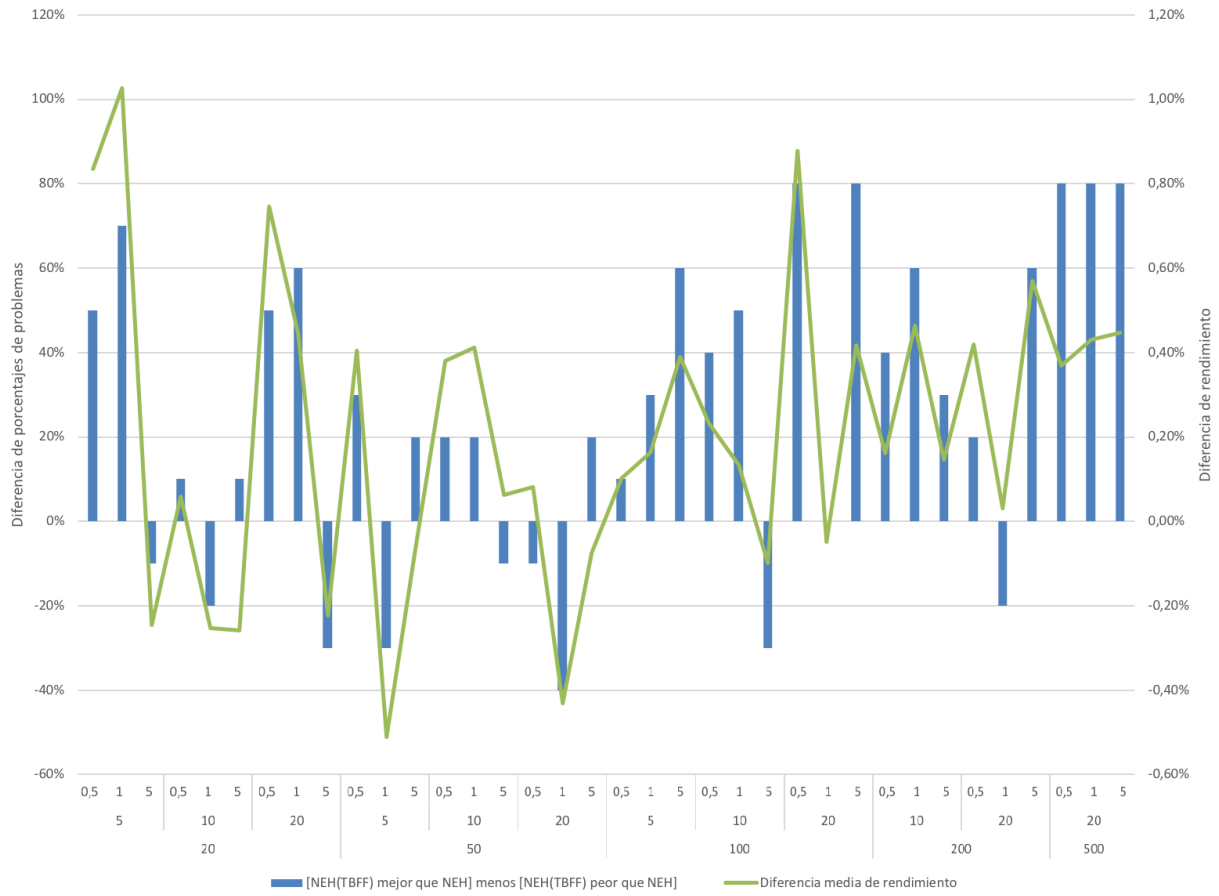


Figura 21. Diferencia de rendimiento media y diferencia entre el porcentaje de problemas en los que cada método es superior.

La calidad de las soluciones obtenidas por el método *NEH(TB_{FF})* es superior a las del método NEH para el 56,39% de los problemas, inferior para el 29,72% de los problemas e idéntica para el 13,89% de los problemas. Como se puede observar en las Figuras 20-21 la ventaja del método *NEH(TB_{FF})* es especialmente significativa para los problemas con un gran número de trabajos, concretamente para los problemas con 500 trabajos, donde obtiene mejores soluciones en un 90% de los casos con una ventaja media del 0,42%.

A continuación, Figuras 22-32, se analizará el efecto del parámetro R_t en la calidad de las soluciones obtenidas por cada uno de los métodos. Este parámetro es de especial interés, ya que, como se explicó en la Sección 4.2, es el encargado de modular de la dispersión de las fechas de comienzo de los trabajos, por tanto, a menores valores de este parámetro mayor es la similitud de estos problemas, *Fm* | r_j; *prmu* | C_{max}, a los problemas sin fecha de comienzo,

Fm | *prmu* | C_{max}. En la Figura 22 se muestran los *ARPD* de los distintos métodos en función del valor de R_t.

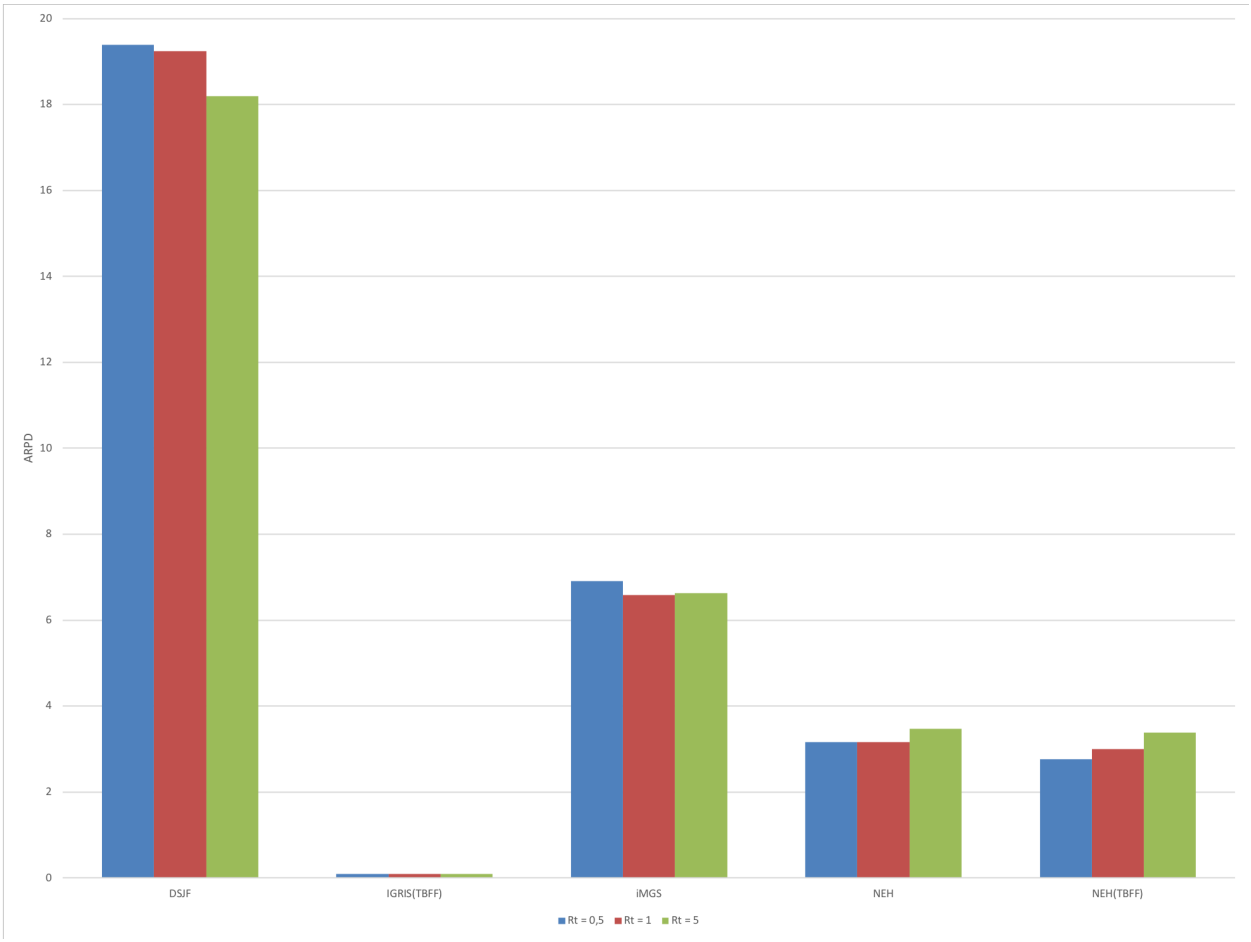


Figura 22. Efecto de R_t.

Como cabe esperar la calidad de los resultados obtenidos por el método *DSJF* aumenta conforme mayor es el valor de R_t, empeorando estos un 1% y un 0,88%, respectivamente para $R_t = 0,5$ y $R_t = 1$, con respecto a los resultados obtenidos para $R_t = 5$. Este mismo fenómeno, aunque de forma más leve, ocurre también con el método *iMGS*, empeorando estos un 0,26% para $R_t = 0,5$ aunque mejorando un 0,04% para $R_t = 1$, con respecto a los resultados obtenidos para $R_t = 5$, debido a que ambos métodos fueron diseñados específicamente para problemas con fechas de comienzo. Por otra parte, los métodos NEH, la calidad de sus resultados empeora un 0,31% para $R_t = 5$ con respecto a los resultados obtenidos para $R_t = 0,5$ y $R_t = 1$, y especialmente *NEH*(TB_{FF}), cuyos resultados empeoran un 0,61% para $R_t = 5$ y un 0,24% para $R_t = 1$ con respecto a los resultados obtenidos para $R_t = 0,5$, obtienen peores resultados conforme aumenta el valor de R_t. Por último, la calidad de las soluciones obtenidas por el método

$IG_{RIS}(TB_{FF})$ parece no verse afectada por los distintos valores del parámetro R_t, ya que su influencia media es menor del 0,4%.

Dado que en el anterior análisis no se ha tenido en consideración cómo puede variar el efecto del parámetro R_t en función de los parámetros restantes, en las Figuras 23-27 se muestra el efecto de este en la calidad de las soluciones obtenidas por cada uno de los métodos en función del número de trabajos y máquinas de cada uno de los problemas.

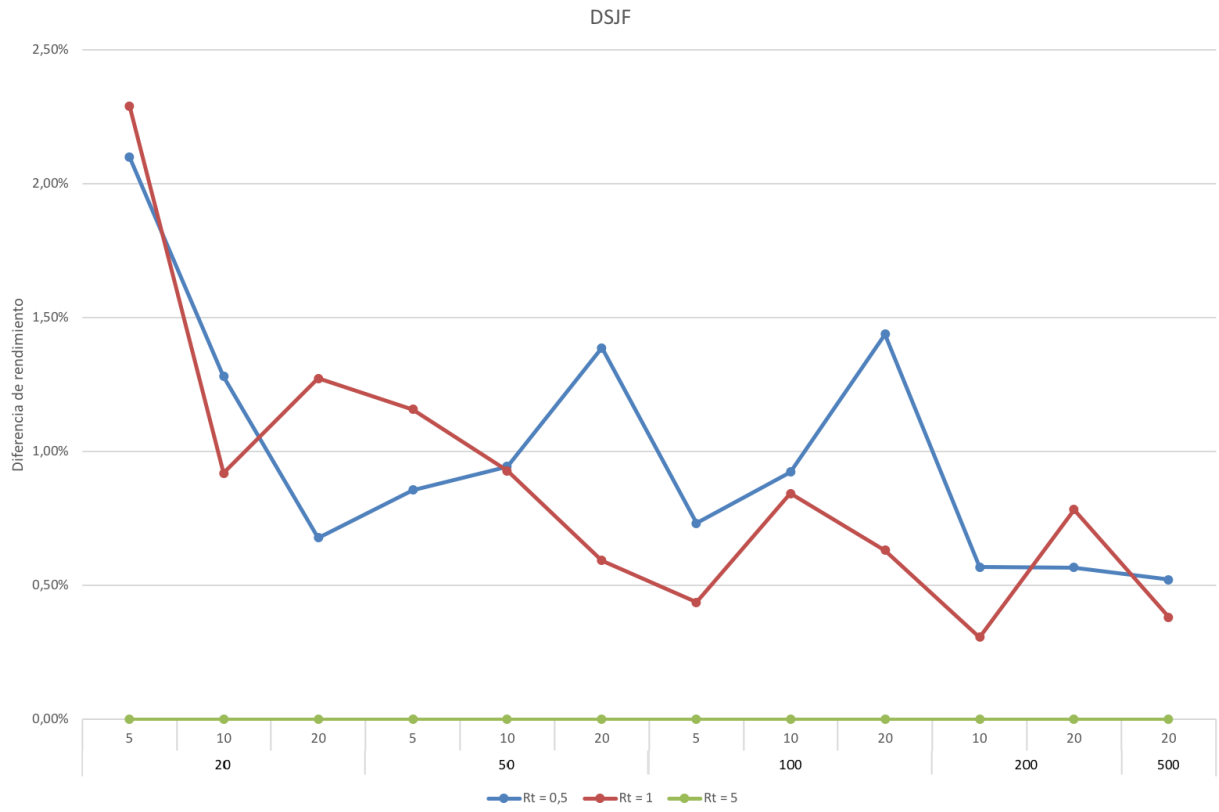


Figura 23. Efecto de R_t en el método *DSJF*.

Como se puede comprobar en la Figura 23, y se comentó anteriormente, los mejores resultados para el método *DSJF* se obtienen, independiente del número de trabajos y máquinas de los problemas, para $R_t = 5$. Si bien es cierto que este efecto se atenúa conforme mayor es el número de trabajos de los problemas.

Con respecto al método *iMGS* se puede comprobar, observando los resultados expuestos en la Figura 25, que pese a que en promedio los peores resultados de este método se obtienen para $R_t = 0,5$, este efecto se revierte para problemas con un gran número de trabajos, obteniendo los peores resultados en estos problemas con $R_t = 5$. Obteniendo por tanto los mejores resultados medios, como se explicó anteriormente, con $R_t = 1$.

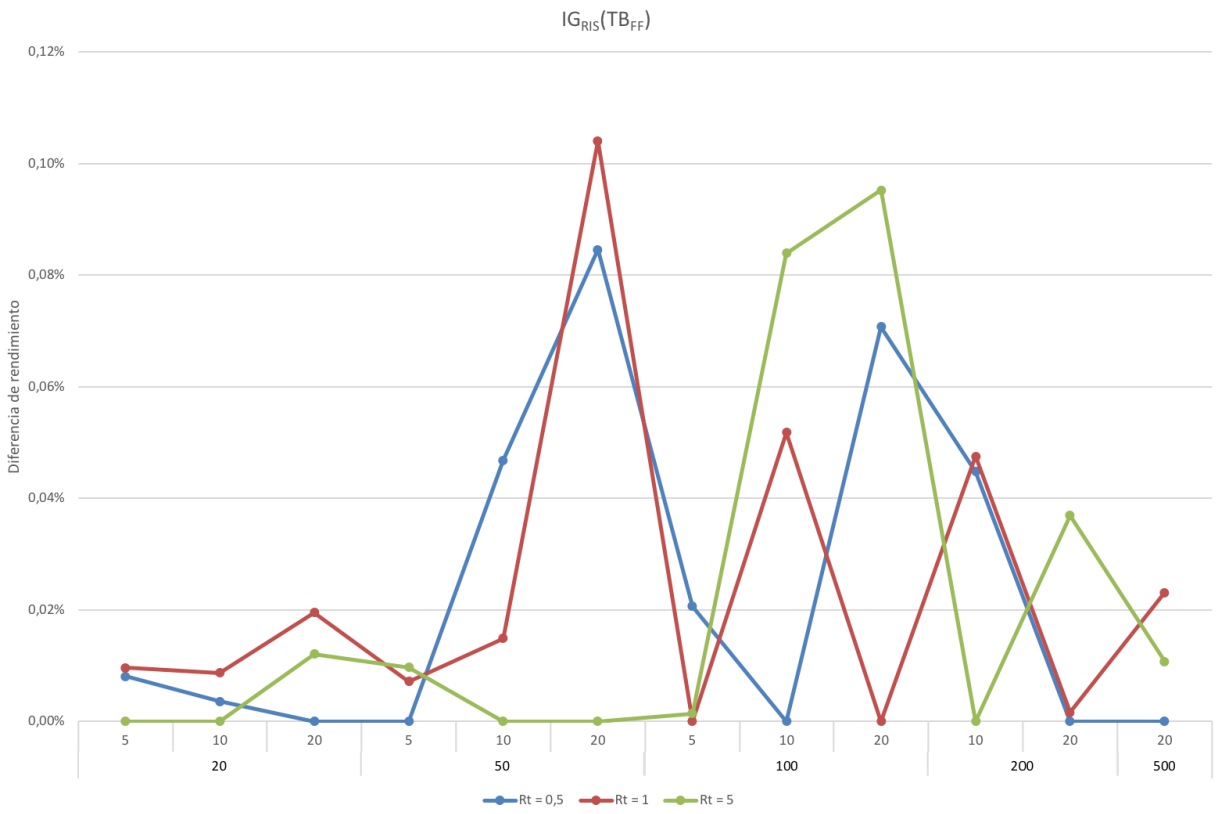


Figura 24. Efecto de R_t en el método $IG_{RIS}(TB_{FF})$.

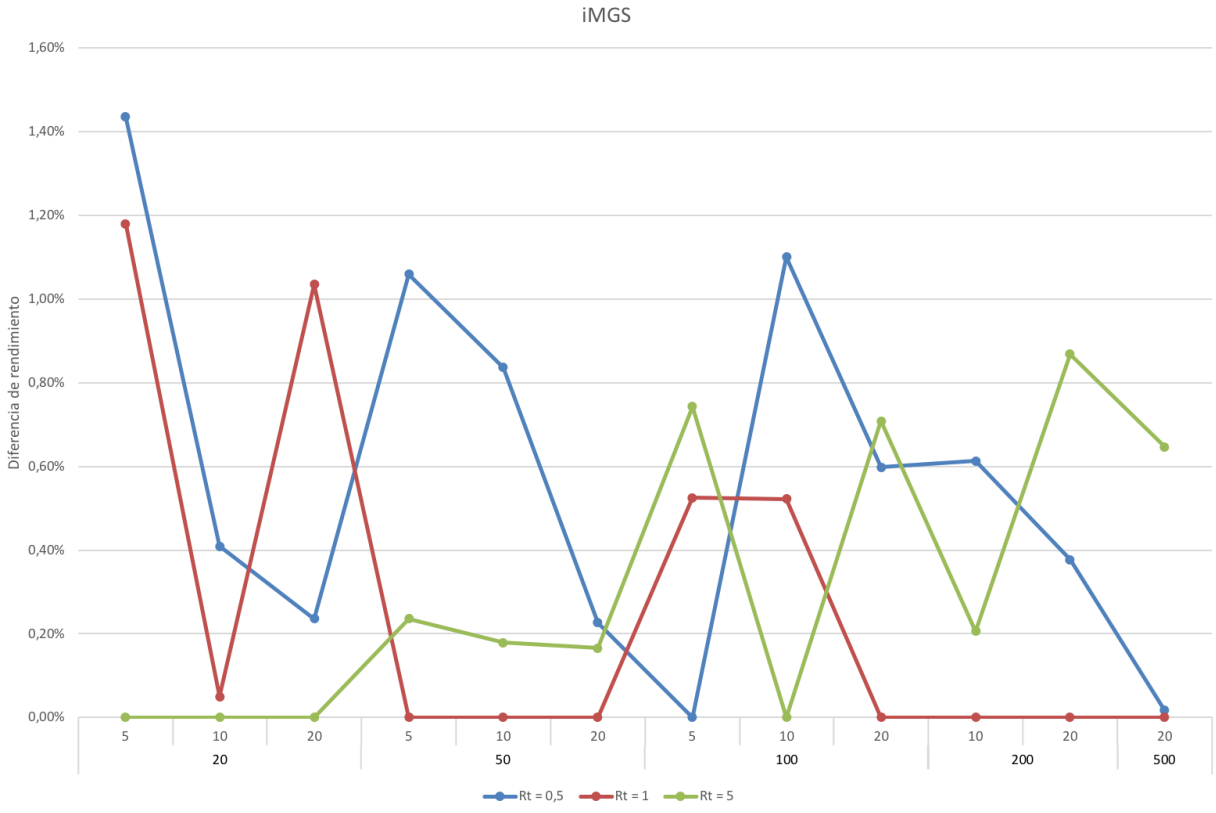


Figura 25. Efecto de R_t en el método *iMGS*.

Los resultados expuestos en la Figura 24 confirman el exiguo efecto del parámetro R_t en la calidad de las soluciones obtenidas mediante el método $IG_{RIS}(TB_{FF})$. Asimismo, muestra que la leve influencia de este parámetro es independiente de los valores del resto de los parámetros de estos problemas.

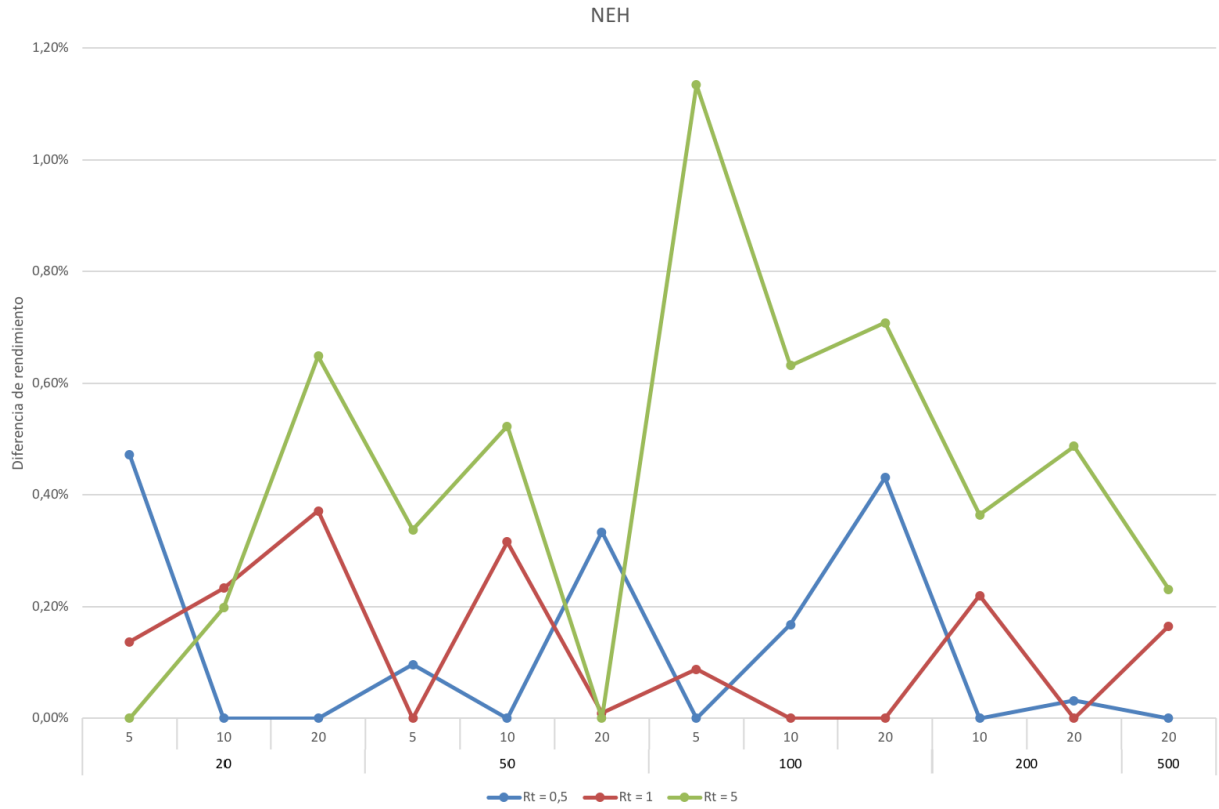


Figura 26. Efecto de R_t en el método NEH.

En la Figura 26 se puede apreciar como para la mayoría de los tipos de problemas, efectivamente, se obtienen los peores resultados con $R_t = 5$. Este efecto también ocurre en el método $NEH(TB_{FF})$, como se puede comprobar en la Figura 27. Como también muestra ese gráfico, este método obtiene, generalmente, los mejores resultados con $R_t = 0,5$, independiente del número de trabajos y máquinas de los problemas.

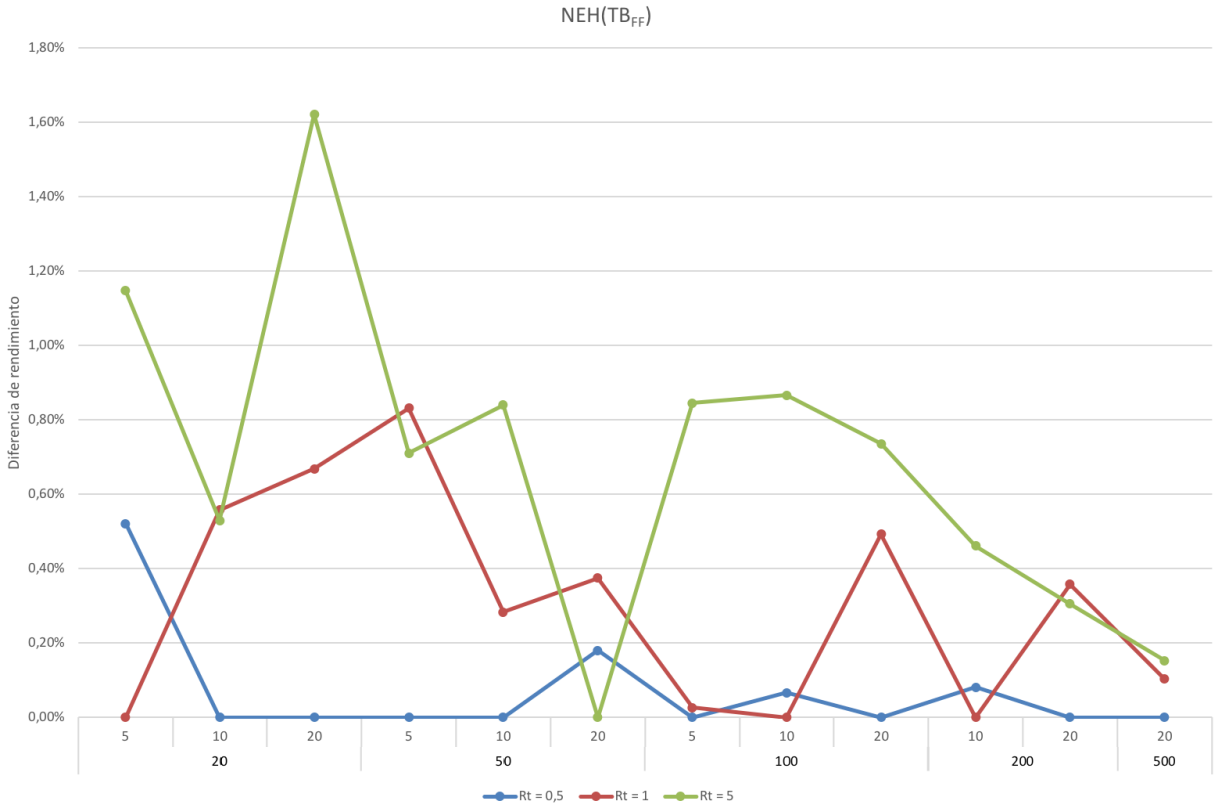


Figura 27. Efecto de R_t en el método *NEH(TB$_{FF}$)*.

Como se ha podido comprobar el efecto de los valores del parámetro R_t no es el mismo para los métodos NEH y *NEH(TB$_{FF}$)*. Por tanto, a continuación, Figuras 28-32, se analizará el efecto del parámetro R_t en la diferencia entre estos métodos. En la Figura 28 se presentan las diferencias medias entre la calidad de las soluciones de ambos métodos.

Figura 28. Rendimiento de los métodos NEH y *NEH(TB_{FF})* en función de los valores de R_t.

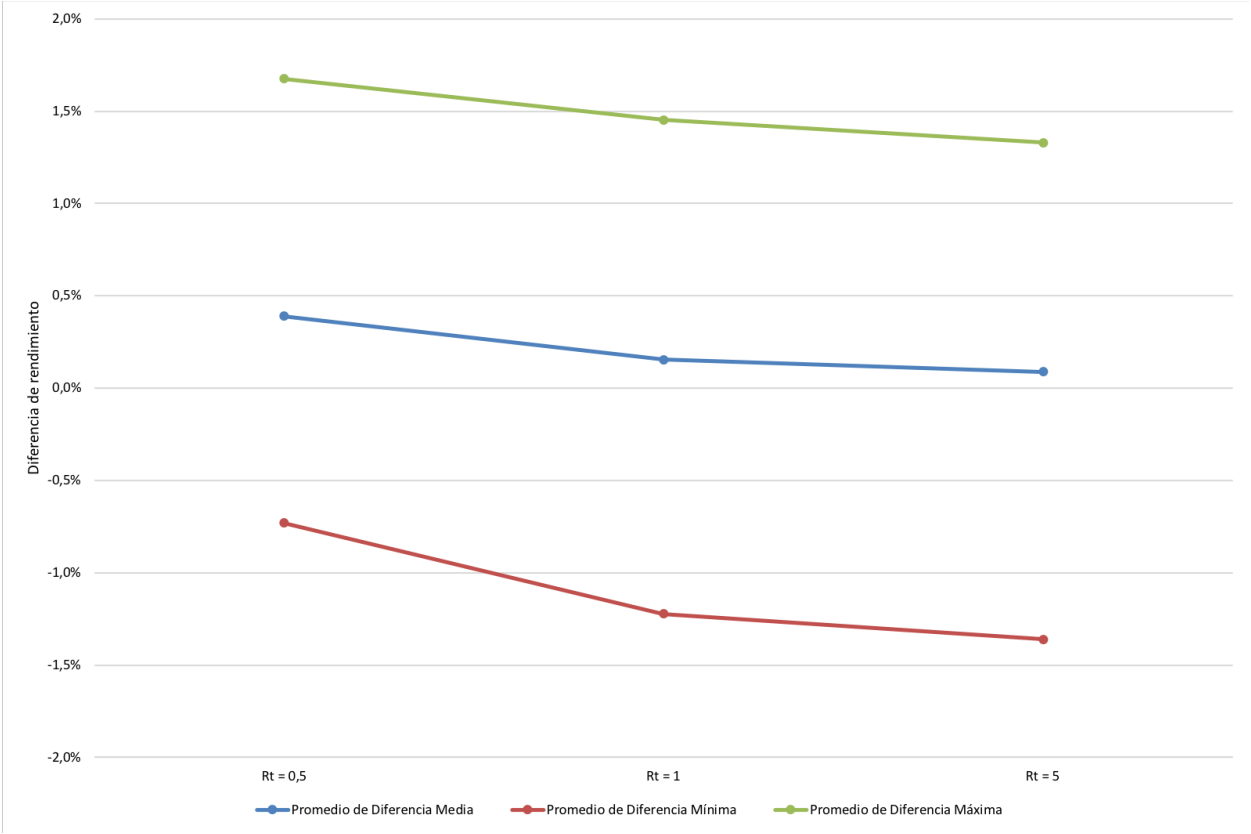


Figura 29. Diferencias de rendimiento máximas, mínimas y medias de los métodos NEH y *NEH(TB_{FF})* en función de los valores de R_t.

En la Figura 29 se puede apreciar como la calidad media de las soluciones obtenidas mediante el método *NEH(TB_{FF})* es superior a la de las obtenidas mediante el método NEH independientemente del valor de R_t, siendo esta diferencia media del 0,39% para $R_t = 0,5$, del 0,16% para $R_t = 1$ y del 0,09% para $R_t = 5$. Concretamente, para los casos en los que $R_t = 0,5$ el método *NEH(TB_{FF})* obtiene mejores resultados medios que el método NEH en todas las categorías de problemas, es decir problemas con el mismo número de trabajos y máquinas, como se puede comprobar en la Figura 28. Asimismo, en esta figura se puede confirmar que la superioridad media del método *NEH(TB_{FF})* frente al método NEH para problemas con un gran número de trabajos ($n \geq 200$) es independiente del parámetro R_t.

Seguidamente, Figuras 30-32, se mostrarán, junto con las diferencias medias entre la calidad de las soluciones, las diferencias entre el porcentaje de problemas en los que el método *NEH(TB_{FF})* es superior al método NEH y los problemas en los que ocurre lo contrario en función de los distintos valores de R_t.

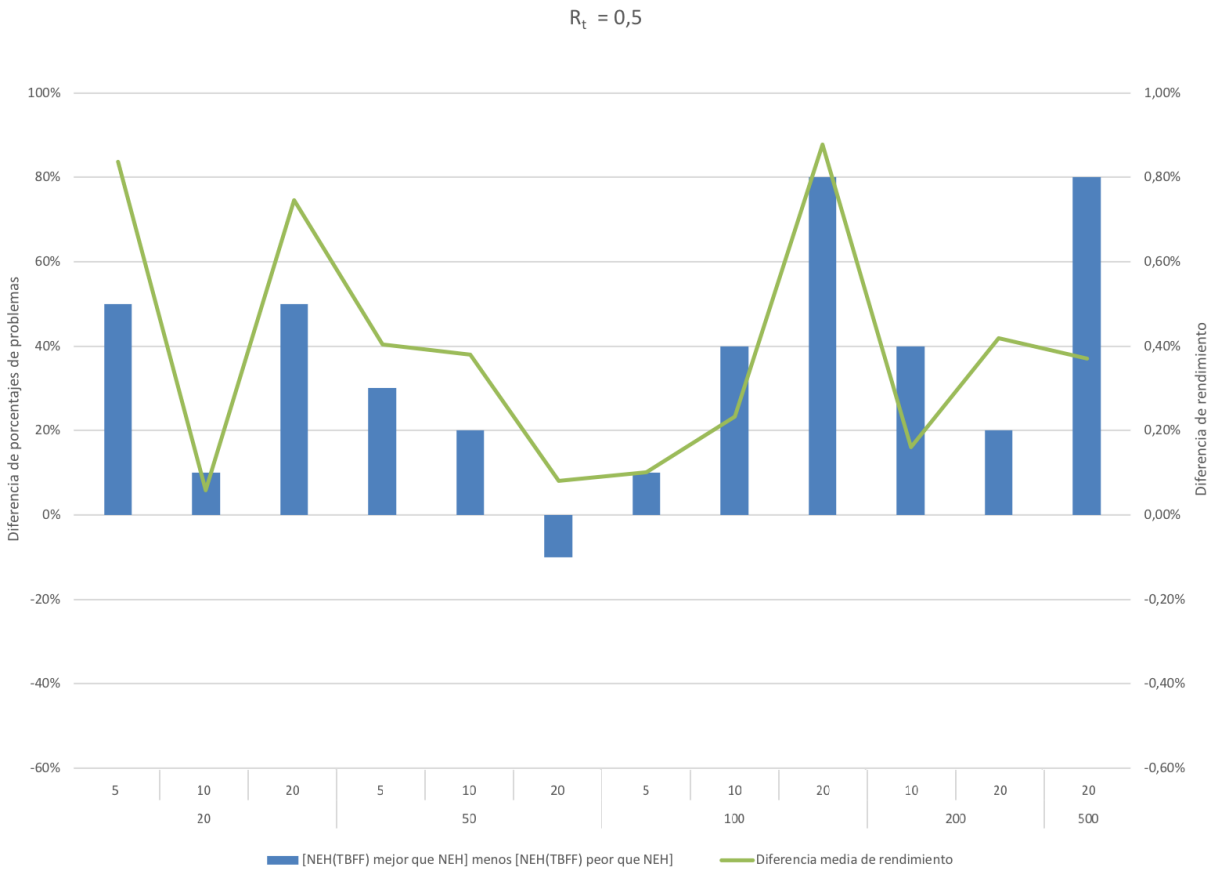


Figura 30. Diferencia de rendimiento media y diferencia entre el porcentaje de problemas en los que cada método es superior para $R_t = 0,5$.

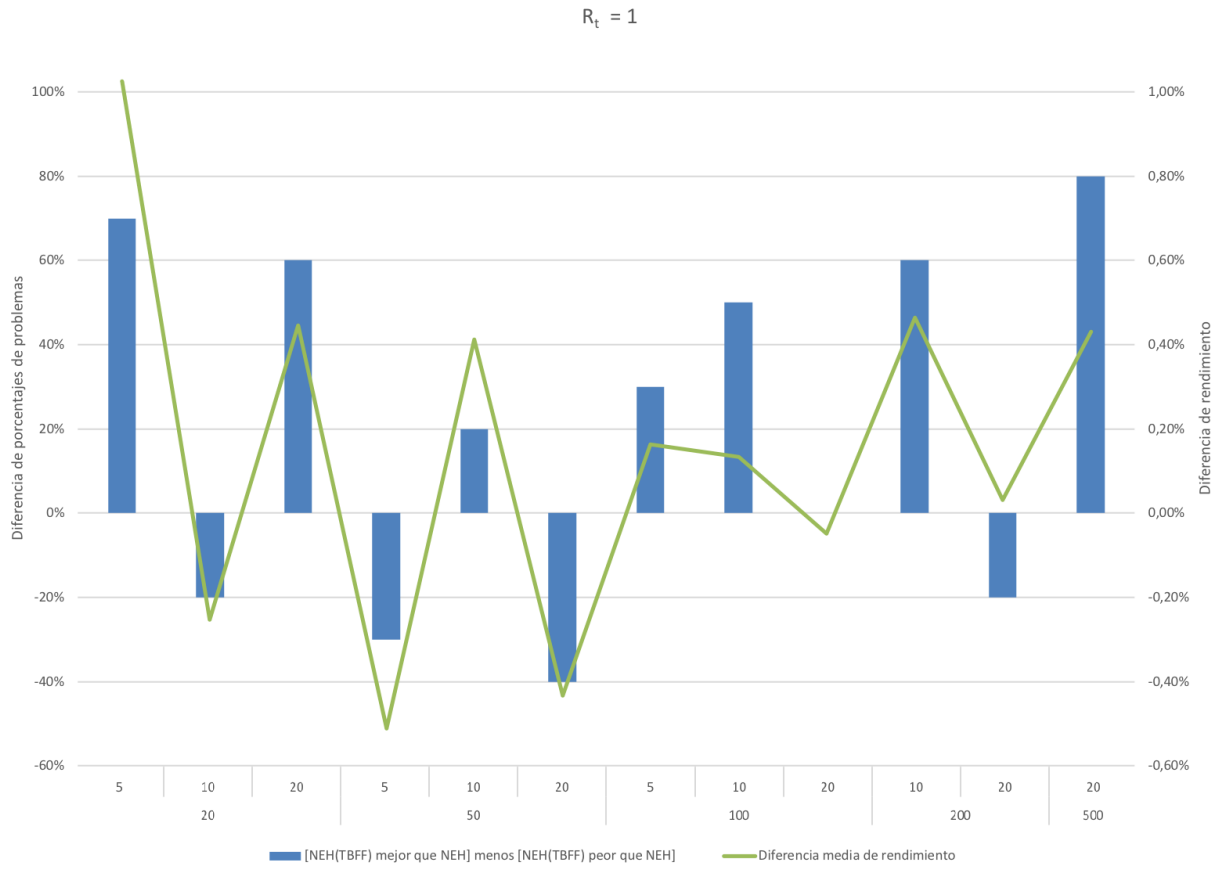


Figura 31. Diferencia de rendimiento media y diferencia entre el porcentaje de problemas en los que cada método es superior para $R_t = 1$.

Los resultados reseñables obtenidos del análisis del porcentaje de problemas en los que la calidad de las soluciones obtenidas por el método *NEH(TB_{FF})* es superior a la de las obtenidas por el método NEH en función de R_t son los siguientes:

- $R_t = 0,5$ (Figura 30): El método *NEH(TB_{FF})* consigue mejores soluciones en el 59% de los problemas e idénticas en el 17%. En todas las categorías de problemas, salvo una, 50 trabajos y 20 máquinas, el número de problemas en los que el método *NEH(TB_{FF})* consigue mejores soluciones supera al número de problemas en los que ocurre lo contrario. Por tanto, y como indicaba la diferencia media de rendimiento, el método *NEH(TB_{FF})* es consistentemente superior al método NEH para $R_t = 0,5$.
- $R_t = 1$ (Figura 31): El método *NEH(TB_{FF})* consigue mejores soluciones en el 55% de los problemas e idénticas en el 12%. Aunque para los problemas con 20, 100, 200 y 500 trabajos el método *NEH(TB_{FF})* obtiene un rendimiento medio superior en un 0,27% al del método NEH, obteniendo mejores resultados en un 61% de los problemas e idénticos en un 12% de los problemas, estos resultados se ven lastrados

por los obtenidos para los problemas con 50 trabajos, en los que el método *NEH(TB_{FF})* obtiene un rendimiento medio inferior en un 0,18% al del método NEH, obteniendo mejores resultados en solo un 37% de los problemas e idénticos en un 10% de los problemas.

- R_t = *5* (Figura 32): El método *NEH(TB_{FF})* consigue mejores soluciones en el 55% de los problemas e idénticas en el 13%. Si bien es cierto que, para los problemas con un gran número de trabajos ($n \geq 100$) el método *NEH(TB_{FF})* obtiene un rendimiento medio superior en un 0,31% al del método NEH, obteniendo mejores resultados en un 68% de los problemas e idénticos en un 10% de los problemas, para los problemas con un pequeño número de trabajos ($n \leq 50$) el método NEH obtiene un rendimiento medio ligeramente superior (0,13%) al del método *NEH(TB_{FF})*, obteniendo mejores resultados en un 42% de los problemas e idénticos en un 18% de los problemas.

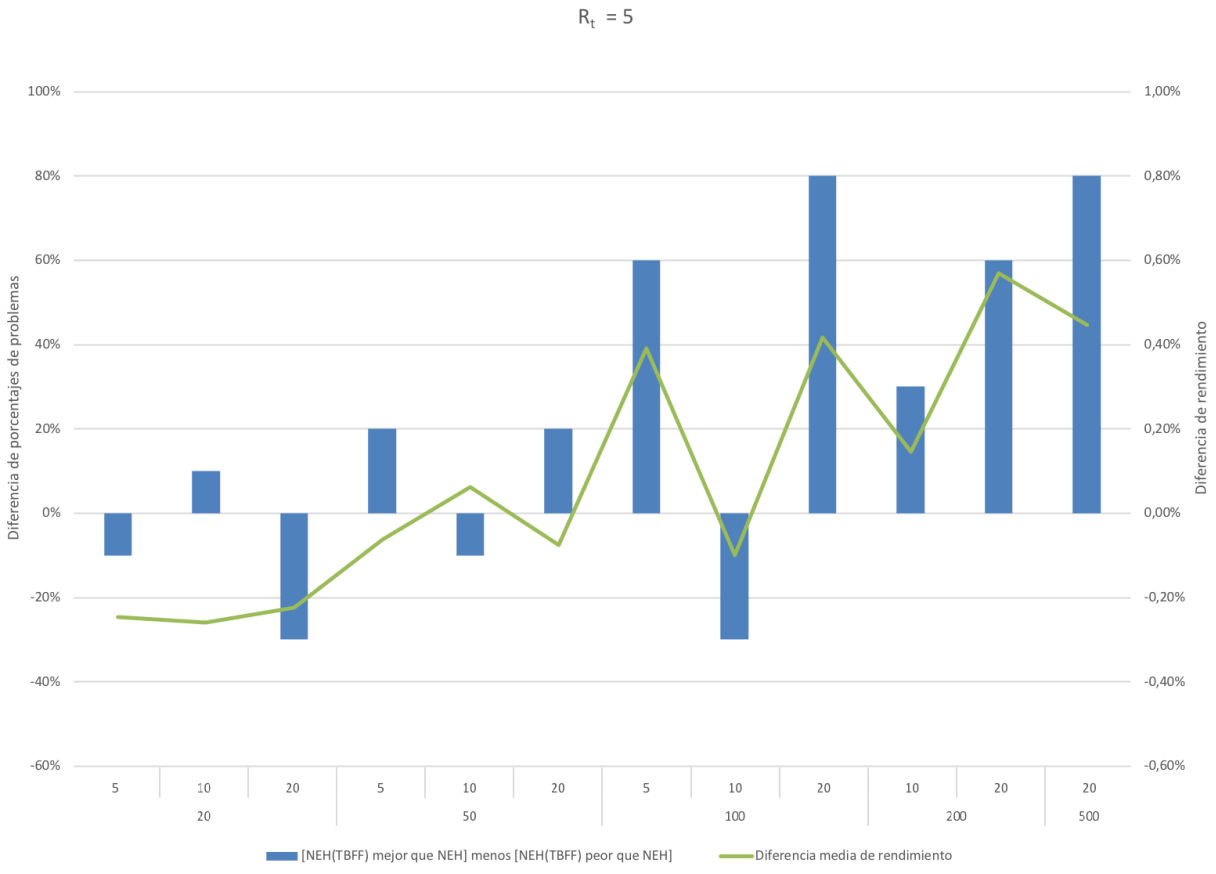


Figura 32. Diferencia de rendimiento media y diferencia entre el porcentaje de problemas en los que cada método es superior para R_t = *5*.

Una vez se a analizado el efecto del parámetro R_t en la Figura 33 se muestran los *ARPD* de cada método en función de los parámetros restantes, número de trabajos y máquinas, de los problemas.

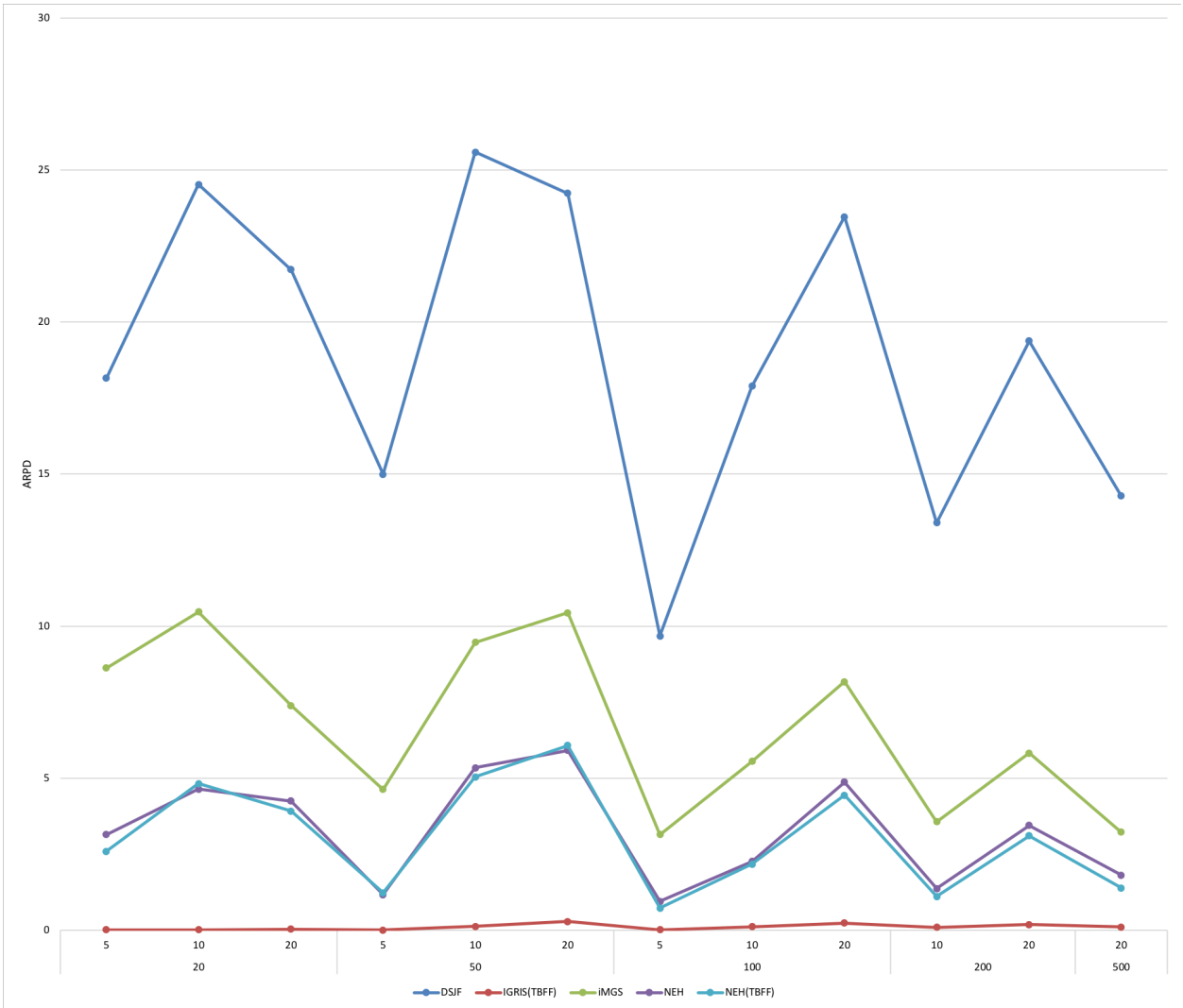


Figura 33. *ARPD* de cada método en función del número de trabajos y máquinas.

Por último, se analizará el efecto del número de trabajos y máquinas de los problemas en la calidad de las soluciones obtenidas por cada uno de los métodos. Para ello, en las Figuras 34-38 se mostrará, individualmente para cada método, la calidad de las soluciones obtenidas por estos en función del número de trabajos y máquinas de los problemas.

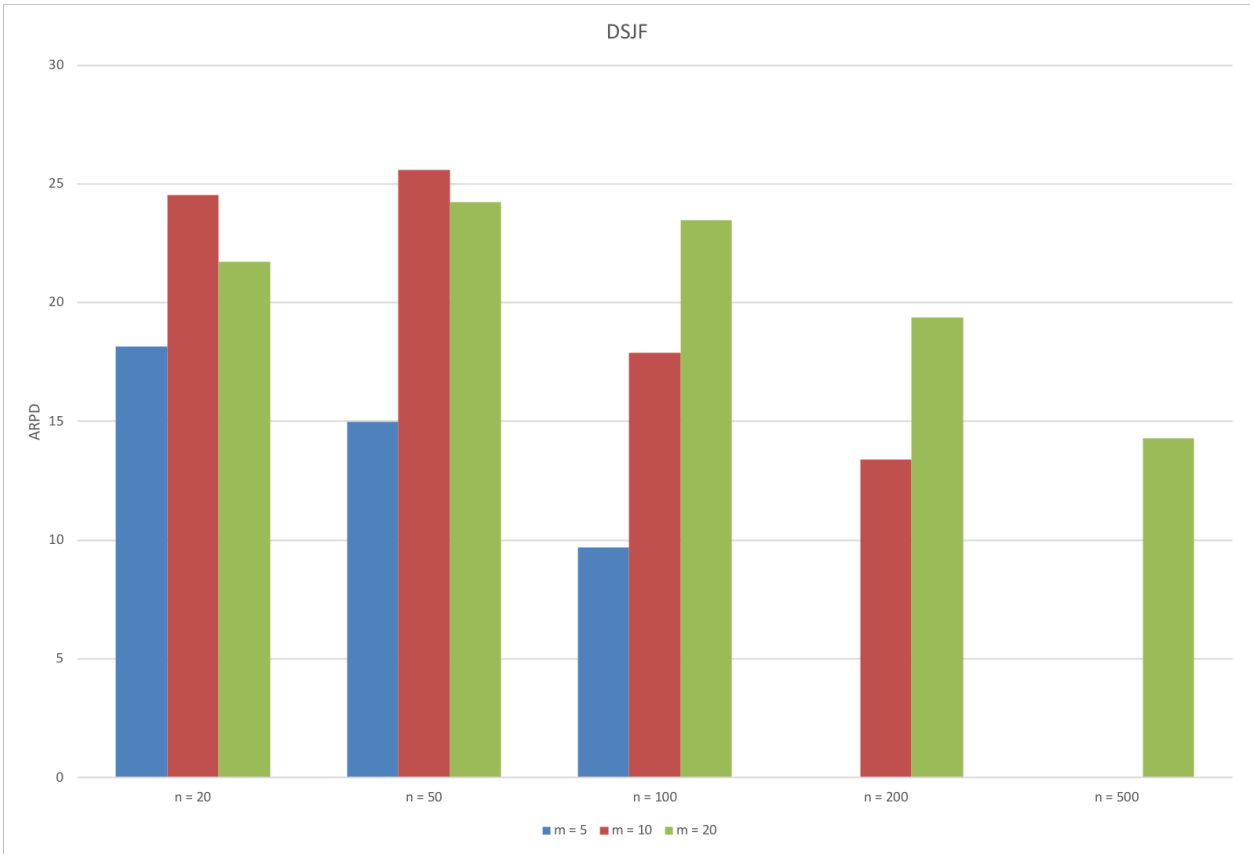


Figura 34. Efecto del número de trabajos y máquinas en el método *DSJF*.

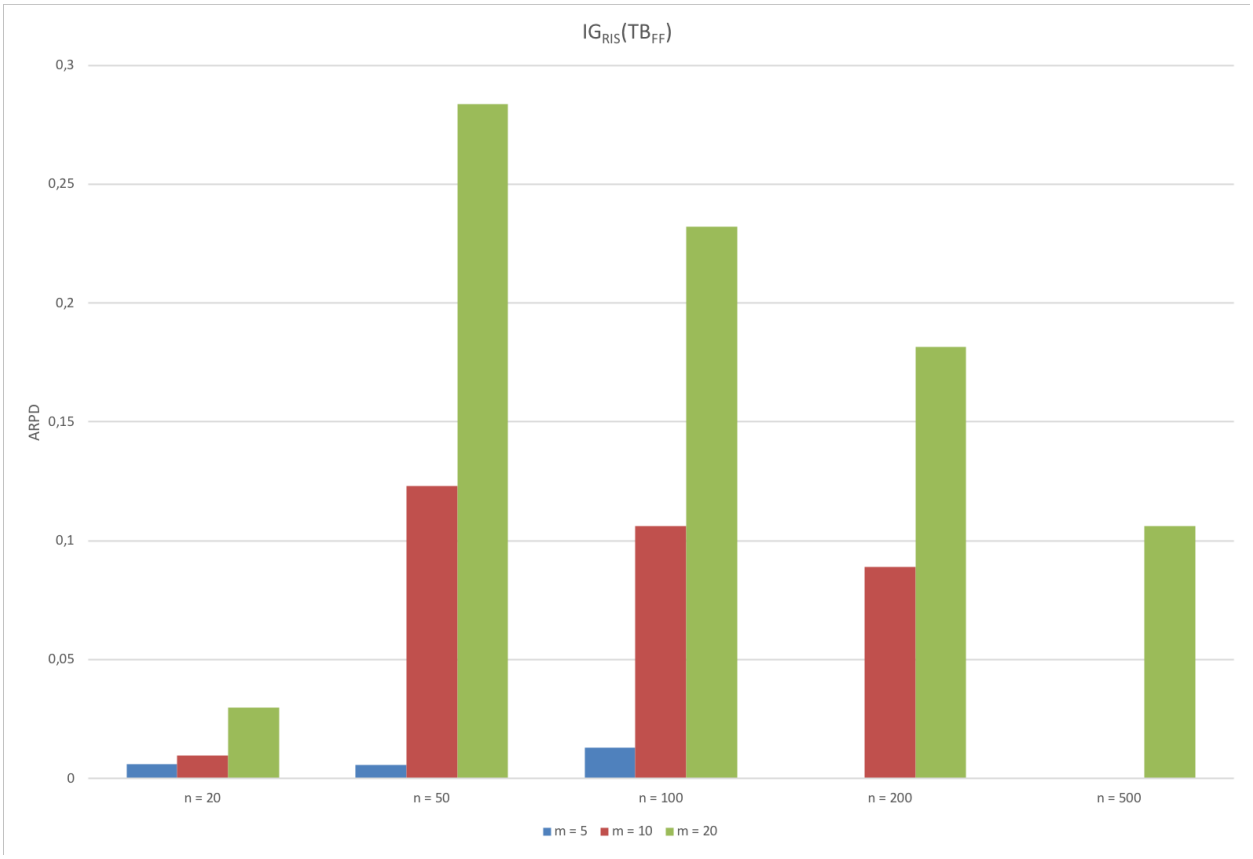


Figura 35. Efecto del número de trabajos y máquinas en el método $IG_{RIS}(TB_{FF})$.

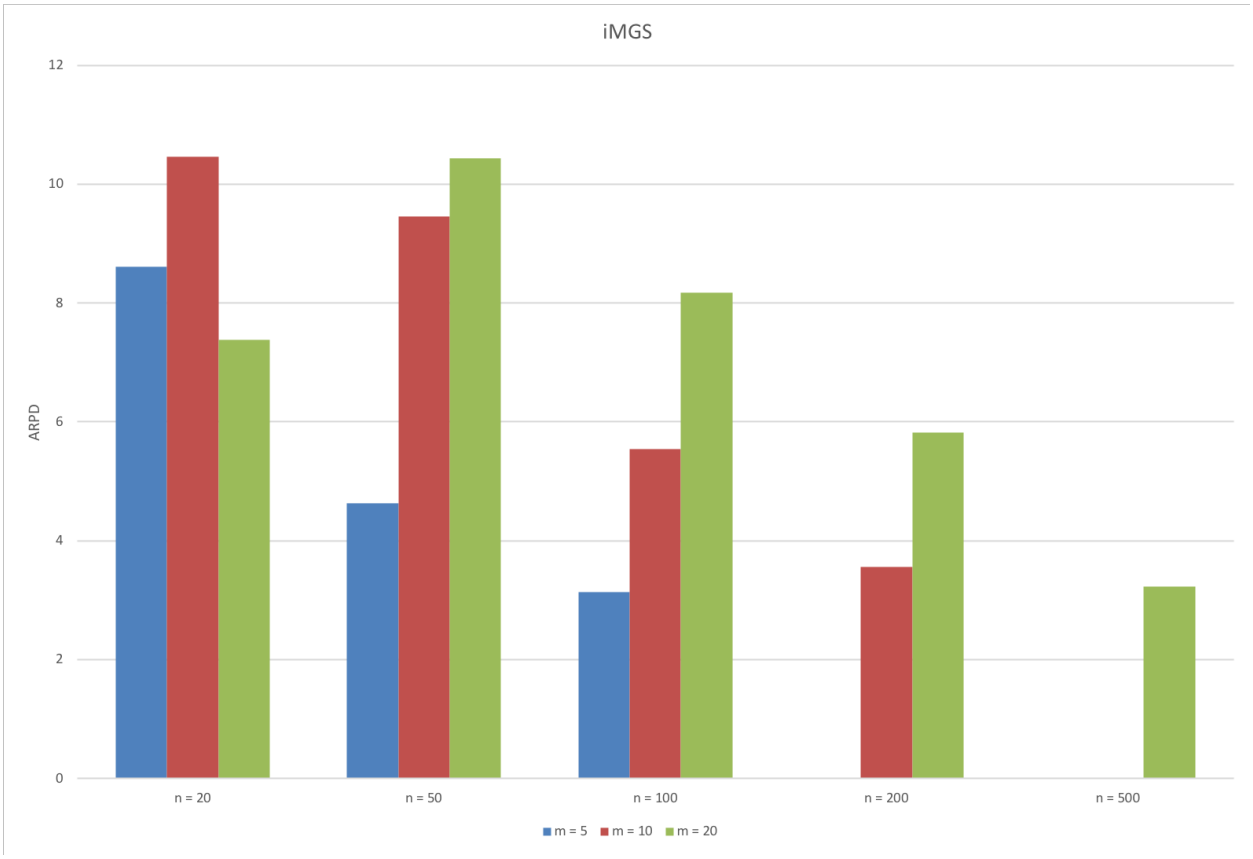


Figura 36. Efecto del número de trabajos y máquinas en el método *iMGS*.

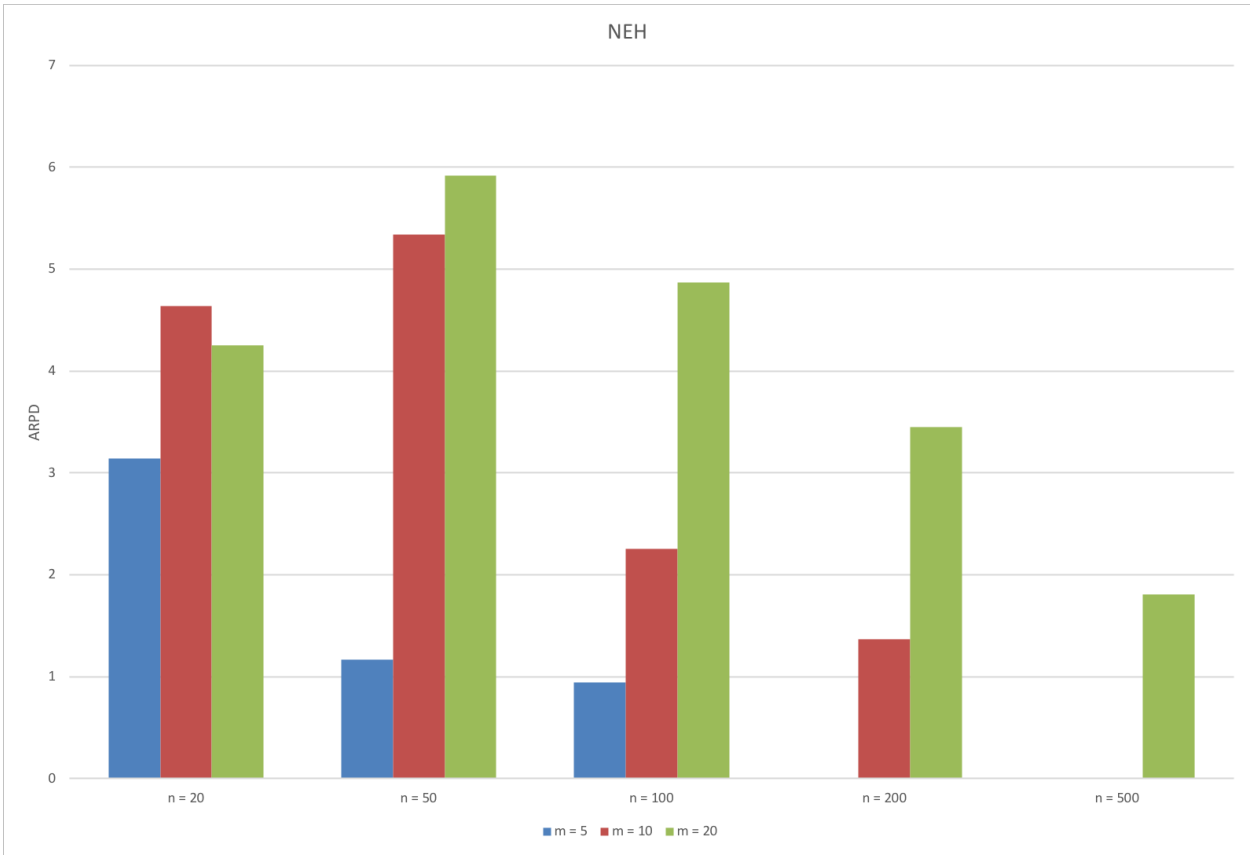


Figura 37. Efecto del número de trabajos y máquinas en el método NEH.

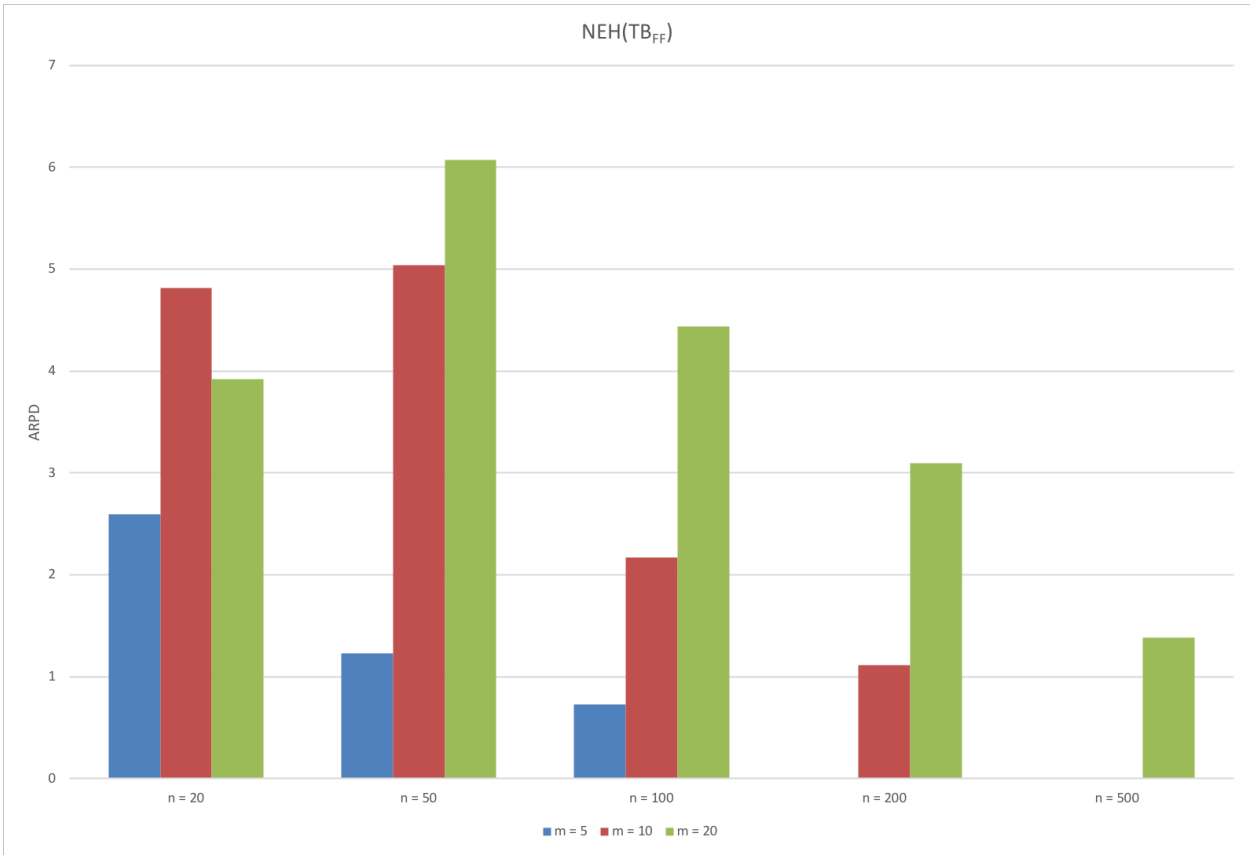


Figura 38. Efecto del número de trabajos y máquinas en el método *NEH(TB$_{FF}$)*.

Los resultados más significativos obtenidos tras analizar el efecto, en cada uno de los métodos objeto de estudio de este Trabajo de Fin de Grado, del número de trabajos y máquinas de los problemas en la calidad de las soluciones obtenidas son los siguientes:

- *DSJF*: Como cabe esperar, ya que este método es asintóticamente óptimo para problemas con un número de trabajos suficientemente grande, la calidad de las soluciones obtenidas aumenta conforme aumenta el número de trabajos de los problemas, para un determinado número de máquinas. Este fenómeno no se cumple en los casos en los que el orden de magnitud del número de trabajos es igual al del número de máquinas, como se puede comprobar en la Figura 34.
- *IG$_{RIS}$(TB$_{FF}$)*: Debido a la sobresaliente calidad de las soluciones obtenidas mediante este método las mejores soluciones encontradas para cada uno de los problemas se han alcanzado mediante este. Sirviendo, por tanto, el *ARPD* de este método en función de los parámetros de los problemas más como un indicador de cómo afectan estos a la variabilidad de sus soluciones que a la calidad de estas.
- *iMGS*: Aunque con una mayor calidad de las soluciones el efecto del número de trabajos y máquinas en este método es similar al del método *DSJF*, como se muestra

en las Figura 34 y 36. Es decir, la calidad de las soluciones obtenidas aumenta conforme aumenta el número de trabajos de los problemas, para un determinado número de máquinas y esto no se cumple en los casos en los que el número de trabajos es similar al número de máquinas ($n \cong m$).

- NEH: Aunque con una calidad de las soluciones ampliamente superior, el efecto del número de trabajos en este método es tambien muy similar al del método *DSJF*. Con respecto al número de máquinas, las mejoras producidas al reducir su número, para un determinado número de trabajos, son significativamente superiores a las producidas en los métodos *DSJF* e *iMGS*.
- *NEH(TB_{FF})*: El efecto del número de trabajos y máquinas en este método es igual al del método NEH, lógicamente, debido a su idéntico funcionamiento salvo en la forma de romper los empates.

Finalmente, se analizarán brevemente los tiempos de ejecución de cada uno de los métodos, mostrados en la Tabla 2:

- *DSJF*: Los tiempos de ejecución medios de este método fueron menores de 0,1 segundos para todas las categorías de los problemas realizados.
- *IG_{RIS}(TB_{FF})*: Los tiempos de ejecución de este método están determinados, como se explico en las secciones 3.3 y 4.3, por un tiempo de finalización prescrito en función del número de trabajos y máquinas del problema a resolver.
- *iMGS*: Aunque para los problemas con un reducido número de trabajos y máquinas los tiempos medios de ejecución de este método fueron menores de 0,1 segundos, estos aumentan vertiginosamente conforme aumenta el tamaño de los problemas, como se puede comprobar en la Tabla 2. A continuación se muestran los tiempos medios de ejecución de este método para esas categorías de problemas:
  I. 100 trabajos y 20 máquinas: 0,3 segundos.
  II. 200 trabajos y 10 máquinas: 0,4 segundos.
  III. 200 trabajos y 20 máquinas: 1,2 segundos.
  IV. 500 trabajos y 20 máquinas: 10,6 segundos.
- NEH: Los tiempos de ejecución medios de este método fueron menores de 0,1 segundos para todas las categorías de los problemas realizados.
- *NEH(TB_{FF})*: Los tiempos de ejecución medios de este método fueron menores de 0,1 segundos para todas las categorías de los problemas realizados salvo en aquellos con 500 trabajos y 20 máquinas, en los que los tiempos medios de ejecución fueron 0,1 segundos.

6 CONCLUSIONES

Mediante este capítulo se destacan las conclusiones más importantes obtenidas tras la realización de este Trabajo de Fin de Grado sobre la programación de operaciones en flujo uniforme con fechas de comienzo.

A continuación, se presentan de manera resumida los pasos que se han seguido en la elaboración de este trabajo:

- En primer lugar, se explican los objetivos de este trabajo, Sección 1.1, y se presenta el problema a resolver, Capítulo 2, comentando sus particularidades y las razones por las que es interesante su estudio.
- A continuación, se presentan y explican los distintos métodos a estudiar, Capítulo 3, señalando las distintas ventajas e inconvenientes de cada uno de ellos, y se justifica el *software* utilizado, se explican los experimentos realizados y los indicadores utilizados, Capítulo 4.
- Por último, en el Capítulo 5, finalmente se procede a mostrar y analizar los resultados obtenidos.

Una vez se han presentado y analizado los resultados obtenidos, estas son las conclusiones más destacables para cada uno de los métodos estudiados:

- $IG_{RIS}(TB_{FF})$: La calidad de sus soluciones es holgadamente superior a la de las obtenidas por el resto de los métodos. Su único inconveniente es su mayor tiempo de ejecución, el cual podría reducirse asignando al parámetro t_s valores más pequeños.
- *iMGS*: Aunque la calidad de sus soluciones es claramente superior a la de las obtenidas por el método *DSJF,* esta es inferior, para la inmensa mayoría de problemas, a la del resto de los métodos. Además, sus tiempos de ejecución aumentan rápidamente conforme aumenta el número de trabajos de los problemas.
- *DSJF*: La utilidad de este método queda reducida a problemas en los que el número de trabajos es varios ordenes de magnitud superior al mayor número de trabajos que se ha considerado en este Trabajo de Fin de Grado. Para estos los tiempos de

ejecución de este método se mantienen en valores suficientemente bajos, pero los de métodos como $IG_{RIS}(TB_{FF})$ e *iMGS* alcanzan valores que hacen inviable su uso.

- $NEH(TB_{FF})$: La calidad de sus soluciones es superior a la de las obtenidas por los métodos *DSJF* e *iMGS,* pero inferior a la del método $IG_{RIS}(TB_{FF})$, sobre el que su principal ventaja son sus menores tiempos de ejecución. Con respecto a su comparación con el método NEH, la calidad de sus soluciones es globalmente superior a la de las obtenidas con el método NEH, destacando especialmente para los problemas con $n \geq 100$ y también en aquellos en los que $R_t = 0,5$.
- NEH: La calidad de las soluciones obtenidas mediante este método es generalmente inferior a la de las obtenidas con el método $NEH(TB_{FF})$, aunque ligeramente superior para la categoría de problemas en los que $R_t = 0,5$ y $n = 20$. Con respecto a su comparación con los métodos restantes las conclusiones son las mismas que para el método $NEH(TB_{FF})$.

Finalmente, y tras haber analizado todos los resultados, es posible afirmar que el mejor de los métodos analizados es $IG_{RIS}(TB_{FF})$ y que el uso del mecanismo de *tie-breaking* TB_{FF} mejora los resultados obtenidos mediante el método NEH.

BIBLIOGRAFÍA Y REFERENCIAS

Bai, D. & Tang, L., 2009. New heuristics for flow shop problem to minimize makespan. Journal of the Operational Research Society, 61(6), pp.1032–1040.

Cook, Stephen A., 1971. The complexity of theorem-proving procedures. STOC '71 Proceedings of the third annual ACM symposium on Theory of computing, pages 151-158.

Dong X, Huang H, Chen P. An improved NEH-based heuristic for the permutation flowshop problem. Comput Oper Res 2008;35(12):3962–8.

Fernandez-Viagas, V. & Framinan, J.M., 2014. On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. Computers & Operations Research, 45, pp.60–67.

Framinan, J., Leisten, R. and Ruíz, R., 2014. Manufacturing Scheduling Systems.

Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math 1979;5:287–326.

Gonzalez, T. and Sahni, S., "Flowshop and jobshop schedules: complexity and approximation," Operations Research, vol. 26, no. 1, pp. 36–52, 1978.

Johnson SM. Optimal two- and three-stage production schedules with setup times included. Naval Res Logist Q 1954;1:61–8.

Kalczynski PJ, Kamburowski J. On the NEH heuristic for minimizing the makespan in permutation flow shops. OMEGA, Int J Manag Sci 2007;35 (1):53–60.

Kalczynski PJ, Kamburowski J. An improved NEH heuristic to minimize makespan in permutation flow shops. Comput Oper Res 2008;35(9):3001–8.

Kalczynski PJ, Kamburowski J. On recent modifications and extensions of the NEH heuristic for flow shop sequencing. Found Comput Decis Sci 2011;36(1):17–34.

Karp, Richard M., 1972. Reducibility among Combinatorial Problems. Complexity of Computer Computations pp 85-103.

Levin, L. A., 1973. Universal Sequential Search Problems. Probl. Peredachi Inf., 1973, Volume 9, Issue 3, Pages 115–116

Nawaz, M., Enscore, E.E. & Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega, 11(1), pp.91–95.

Pan Q-K, Tasgetiren M, Liang Y-C. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Comput Ind Eng 2008;55(4):795–816.

Ren, T. et al., 2015. A Local Search Algorithm for the Flow Shop Scheduling Problem with Release Dates. Discrete Dynamics in Nature and Society, 2015, pp.1–8.

Ribas I, Companys R, Tort-Martorell X. Comparing three-step heuristics for the permutation flow shop problem. Comput Oper Res 2010;37(12):2062–70.

Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res 2007;177(3):2033–49.

Ruiz, Rubén. Scheduling Heuristics, 2015, pp 1-24. 10.1007/978-3-319-07153-4_44-1.

Taillard E. Some efficient heuristic methods for the flow shop sequencing problem. Eur J Oper Res 1990;47(1):65–74.

Taillard E. Benchmarks for basic scheduling problems. Eur J Oper Res 1993;64(2):278–85.

http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html

http://www.rae.es