Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Metaheurísticas constructivas para la secuenciación de máquinas en paralelo con ajustes entre trabajos y necesidad de recursos adicionales

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: López Esteve, Axel Javier

Tutor: Perea Rojas Marcos, Federico

Curso: 2019-2020

Resumen

En este trabajo se pretende resolver el problema de secuenciación de máquinas paralelas no relacionadas con ajustes entre trabajos y recursos limitados adicionales. El objetivo del problema es minimizar el tiempo de finalización del último trabajo en ser procesado (*makespan*). Este *makespan* puede ser determinante para cualquier empresa con una cadena de producción a la hora de cumplir plazos de entrega o mejorar la productividad. Para abordar este problema se diseñarán e implementarán heurísticas que estarán formadas por dos fases: una fase constructiva donde se obtendrá una primera solución del problema, y una fase de reparación donde se arreglará la solución obtenida en la fase anterior si se supera el número de recursos permitidos. Además, para conseguir una mayor diversidad de soluciones, se aleatorizará parte de la fase constructiva, convirtiendo las heurísticas en metaheurísticas. Más adelante se realizará una extensa fase de experimentos y posterior análisis, en el que se probarán diferentes versiones de las heurísticas y metaheurísticas variando diferentes factores. A continuación, se compararán los resultados obtenidos por los algoritmos propuestos, con los logrados previamente para el mismo problema por un modelo de programación lineal entera (MILP, por sus siglas en inglés *Mixed Integer Linear Program*). La conclusión principal de estos experimentos es que, con un tiempo de proceso muy inferior, se pueden casi igualar y en algunos casos mejorar los resultados de este modelo utilizando los algoritmos heurísticos y metaheurísticos propuestos.

Palabras clave: heurísticas, problemas de secuenciación, máquinas paralelas, tiempos de ajuste dependientes de secuencia, recursos adicionales, *makespan*.

Resum

En aquest treball es tracta de resoldre el problema de seqüenciació de màquines paral·leles no relacionades amb ajustos entre treballs i recursos limitats adicionals. L'objectiu del problema es minimitzar el temps de finalització de l'últim treball a ser processat (*makespan)*. Este *makespan* pot ser determinant per a qualsevol empresa amb una cadena de producció a l'hora de complir terminis de lliurament o millorar la productivitat. Per a abordar aquest problema es dissenyaran e implementaran heurístiques que estaran formades per dues fases: una fase constructiva on s'obtindrà una primera solució del problema, i una fase de reparació on s'arreglarà la solució obtinguda en la fase anterior si es supera el nombre de recursos permitits. A més, per a aconseguir una major diversitat de solucions, s'aleatoritzarà part de la fase construcitva, convertint les heurístiques en metaheurístiques. Més endavant es realitzarà una extensa fase d'experiments i posterior anàlisi, en el qual es provaran diferents versions de les heurístiques i metaheurístiques variant diferents factors. A continuació, es compararan els resultats obtinguts pels algorismes proposats, amb els reeixits prèviament per al mateix problema obtinguts amb un model de programació lineal sencera (MILP, per les seues sigles en anglés *Mixed Integer Linear Program*). La conclusió principal d'aquests experiments és que, amb un temps de procés molt inferior, es poden quasi igualar i en alguns casos millorar els resultats d'aquest model utilitzant els algorismes heurístics i metaheurístics proposats.

Paraules clau: heurístiques, problemes de seqüenciació, màquines paral·leles, temps d'ajust dependents de la seqüencia, recursos addicionals, *makespan*.

Abstract

This project aims to solve the unrelated parallel machines scheduling problem with adjustments between jobs and additional limited resources. The goal of this problem is to minimize the completion time of the last job (*makespan*). This *makespan* can be decisive for any business with production chains when it comes to respect deadlines or improving productivity. To engage this problem, design and implementation of heuristics and metaheuristics will be done, and they will be split in two phases: a constructive phase where a first solution will be obtained, and a repair phase where this solution will be fixed in case of exceeding the resources limit. Furthermore, to achieve a major diversity of solutions, part of the construction phase will be randomized, transforming heuristics into metaheuristics. Later, an extensive experiments phase will be done with a proper analysis, where different versions of the heuristics and metaheuristics will be tested varying different factors. Next, the results obtained will be compared with the ones achieved for the same problem by an integer linear programming model (MILP). The main conclusion of these experiments is that, with much less processing time, the results obtained with the model are similar or in some instances even worse, compared to the ones achieved with the proposed heuristic and metaheuristic algorithms.

Keywords: heuristics, scheduling problem, parallel machines, sequence dependent setup times, additional resources, *makespan*.

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

En este trabajo de fin de máster se va a tratar un problema particular de secuenciación: la secuenciación de trabajos en máquinas paralelas no relacionadas con tiempos de ajuste y necesidad de recursos en todas las etapas. Este problema consiste en la asignación de n trabajos con un tiempo de proceso determinado, a un conjunto de m máquinas, con la finalidad de realizar el procesamiento de cada uno de ellos. Un ejemplo sencillo de este tipo de problema, ilustrado en parte en la Figura 1, podría ser una cadena de producción logística con diferentes máquinas, en las que en cada una de ellas se van embalando diferentes paquetes. Según el tamaño del paquete, la fragilidad, o la forma, entre otras características, se requiere de un tiempo mayor o menor de procesamiento en cada una de las máquinas. Además, si el paquete es frágil puede necesitar un supervisor para añadir espuma en el interior, otro supervisor puede ser necesario para revisar el embalaje de los paquetes frágiles, etc.

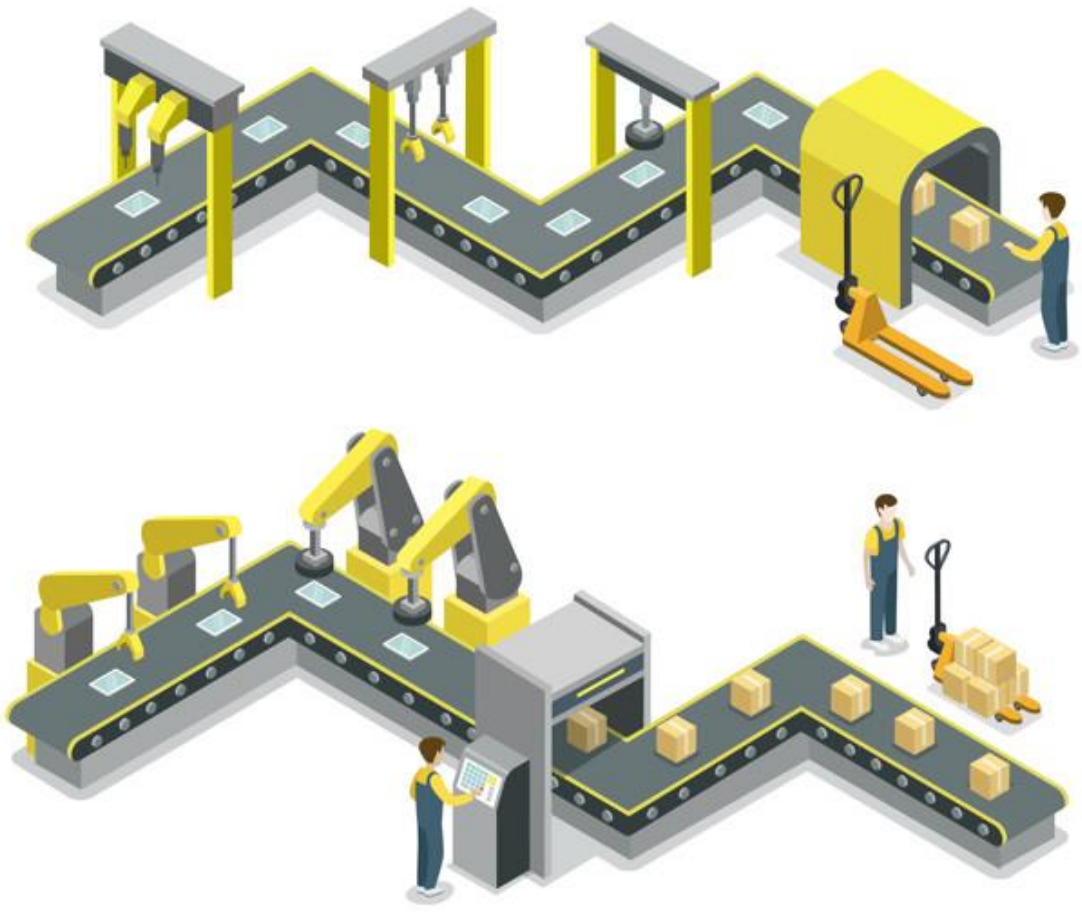

Figura 1: Cadena de producción. Studioworkstock (2019), Production line isometric illustration infographic Premium Vector.

Si el número de máquinas y trabajos es pequeño, se podrían probar todas las combinaciones posibles de los n trabajos y las m máquinas diferentes, y guardar la mejor. Se considerará que "mejor" significa que el instante en el que finaliza el procesado del último trabajo sea lo menor posible (concepto conocido como *makespan* en la literatura). Pero probar todas las combinaciones no es una opción viable cuando el tamaño del problema es grande, ya que la resolución del problema con optimalidad es de coste no polinomial. Puede ser una elección viable si el número de trabajos y máquinas es reducido, pero nada escalable conforme el número de estos dos aumentan.

En particular, este tipo de trabajos tienen una serie de restricciones básicas y lógicas que se deben cumplir a lo largo de toda la ejecución del algoritmo. Algunas de las más encontradas tanto en la literatura relacionada como en las plantas de producción de la industria son:

1) Ninguna máquina puede procesar más de un trabajo al mismo tiempo, aunque sí pueden trabajar en paralelo.
2) Todos los trabajos tienen que ser procesados hasta ser completados.
3) Una tarea no puede iniciarse hasta que la anterior tarea haya finalizado.
4) La ejecución finalizará cuando todos los trabajos hayan sido completados.
5) Se requiere un tiempo de ajuste de las máquinas entre dos trabajos.
6) Tanto el procesado de los trabajos como los ajustes de las máquinas requieren de la presencia de un cierto número de recursos adicionales (como operarios, por ejemplo).

Dada la complejidad del problema (catalogado como NP-Duro, como se adelantó anteriormente) no es posible resolverlo garantizando optimalidad en un tiempo de cómputo polinomial. Sí que existen métodos exactos que aseguran encontrar una solución óptima, pero estos requieren un tiempo de computación muy elevado. Por ello, en este trabajo se presentarán diferentes algoritmos heurísticos y metaheurísticos, cuya aplicación no garantiza encontrar la solución óptima, pero en cambio sí asegura encontrar una solución factible en un tiempo de cómputo razonable. Los algoritmos propuestos estarán divididos en diferentes fases y se realizarán diferentes pruebas, teniendo en cuenta diversos factores que se explicarán en el apartado "Resultados del algoritmo aleatorizado".

1.1 Motivación

Cualquier empresa hoy en día si quiere ser competitiva, entre otros muchos aspectos debe buscar constantemente estrategias para reducir costes e incrementar la productividad en los procesos que forman parte de su funcionamiento. Una de esas fases si la empresa tiene una cadena de producción, embalaje, o construcción de sus productos entre otros, es la programación de la producción en general, y entrando más en detalle, la secuenciación de tareas. Este aspecto puede ser una fase crucial a la hora de cumplir fechas de entrega, reducir costes y aumentar la productividad.

Es por ello que tener una metodología para organizar eficientemente entre las máquinas y los trabajadores las tareas a realizar, que es lo que se pretende con este proyecto, puede suponer una diferencia frente a la competencia muy importante para los beneficios de la empresa sin tener ningún factor negativo, o un coste alto adicional.

A pesar de que el problema de "organizar eficientemente las tareas a realizar entre los trabajadores y las máquinas" puede parecer un único problema, hay muchas variaciones que dependen específicamente de como funcione una cadena de producción en una empresa. Por ejemplo, como se ha explicado en el anterior ejemplo de la Figura 1, puede que para los trabajos de embalaje se precisen de recursos adicionales como personas, puede que en otro ejemplo no se necesiten de estos recursos adicionales, o que incluso se precisen de tipos de recursos concretos según la tarea a realizar. Entre todas estas variaciones, las soluciones pueden ser diferentes, y en este trabajo se propone abordar un problema concreto de entre los muchos existentes de secuenciación.

1.2 Objetivos

El objetivo del presente trabajo es realizar un algoritmo que resuelva un problema de secuenciación. Más concretamente, el problema de máquinas paralelas no relacionadas con ajustes entre trabajos y necesidad de recursos adicionales (la abreviatura utilizada en la literatura sería UPMSR, por sus siglas en inglés: *Unrelated Parallel Machines Scheduling Problem with Setup and Resources*).

Para ello, en el Capítulo 2 se dará una explicación de los problemas de máquinas paralelas relacionadas, el más sencillo, y se irá ampliando poco a poco hasta llegar a un problema más complejo que es el problema objetivo de este proyecto.

En el Capítulo 3 se realizará un repaso de cuatro métodos de resolución que sirven para resolver problemas de secuenciación como el tratado en este proyecto.

El siguiente paso será realizar una revisión bibliográfica enumerando algunos trabajos previos relacionados al problema a resolver en esta memoria, en el Capítulo 4.

Más adelante, en el Capítulo 5 se llevará a cabo una definición formal con notación matemática del problema de máquinas paralelas no relacionadas con tiempos de ajuste dependientes y recursos limitados. Se formalizará el problema a partir de un modelo de programación matemática de la literatura.

Una vez explicado y definido en detalle el problema, en el Capítulo 6 se diseñarán e implementarán algoritmos heurísticos y metaheurísticos que resuelvan este problema, no con una solución óptima (ya que esto no es posible para instancias de tamaño realista, por la complejidad del problema) pero sí con una solución cercana a la óptima y que se podría considerar como buena teniendo en cuenta el tiempo computacional utilizado para obtenerla.

A continuación, se explicará la heurística constructiva diseñada apoyándose en pseudocódigo para mayor entendimiento y se realizarán unas primeras pruebas. Para

mejorar el rendimiento de la heurística, en el Capítulo 7 se propondrán aleatorizaciones en algunos de sus pasos, lo cual se verá en los experimentos que aumenta significativamente la calidad de las soluciones encontradas.

Después llegará la fase de experimentos en el Capítulo 8, donde se utilizarán varios factores para variar algunas fases de la heurística y tratar de obtener mejores resultados. Estos resultados serán analizados para así poder decidir qué niveles de los factores de los algoritmos son los más idóneos.

Por último, en el Capítulo 9 se presentarán unas conclusiones en base a los resultados obtenidos y se expondrán los siguientes pasos en una posible continuación de este trabajo en el futuro.

2. Problemas de Secuenciación

En este capítulo se explicará con detalle el problema de secuenciación de máquinas paralelas que abordamos en este trabajo. Para ello se irán comentando varias versiones de este tipo de problemas, desde la más simple a la versión más compleja. Cada nueva versión añadirá un punto más de complejidad y realismo con respecto a la anterior. Para ayudar a la comprensión de cada problema mostrado se utilizará un ejemplo sencillo que exponga el distintivo común de dicha versión.

2.1 Máquinas paralelas idénticas

El primer tipo, el más simple, será un problema en el que existirán un número de trabajos n, y un número de máquinas m. Todos estos trabajos deberán ser procesados en una de las máquinas, pero en cualquiera de las m máquinas el trabajo n va a tener el mismo tiempo de procesamiento porque las máquinas son idénticas. Para el tiempo de procesamiento de un trabajo es indiferente asignarlo a una máquina u otra. Los tiempos de proceso de cada trabajo (J1, J2, J3, J4) en cada máquina (M1, M2) vienen expresados en la Tabla 2:

	M1	**M2**
J1	7	7
J2	4	4
J3	2	2
J4	6	6

Tabla 1: Tiempos de proceso para dos máquinas paralelas idénticas

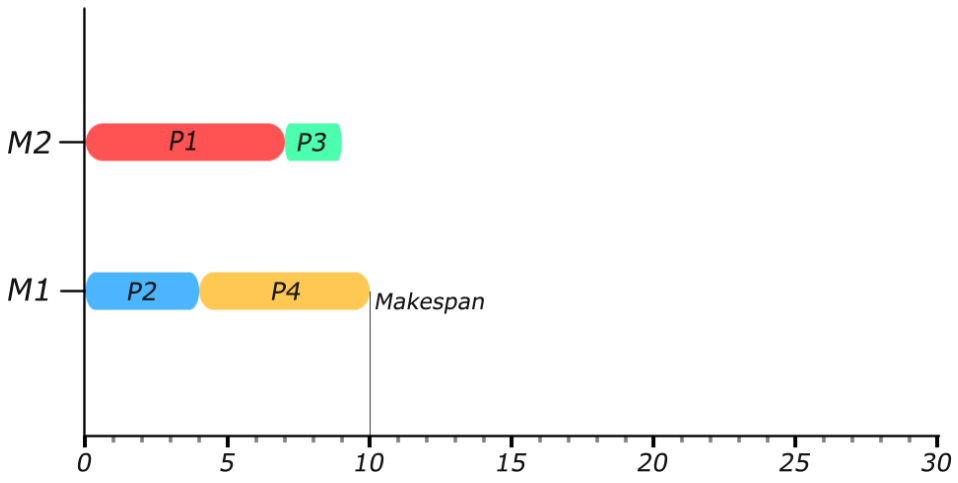

Figura 2: Ejemplo de ejecución con dos máquinas idénticas

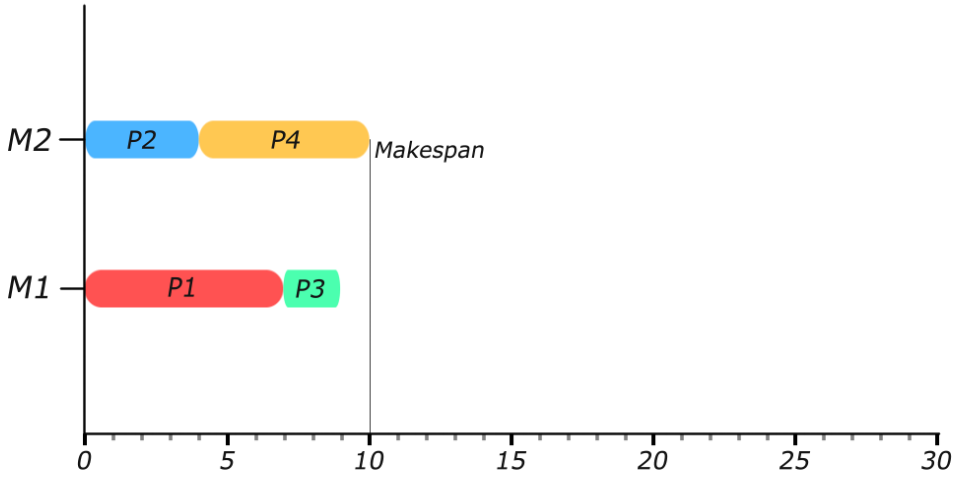

Figura 3: Ejemplo de ejecución con dos máquinas idénticas

En las Figuras 2 y 3 se muestran dos posibles asignaciones de los cuatro trabajos entre las dos máquinas disponibles. En estas dos asignaciones se puede observar la relación total entre las dos máquinas. En la primera figura de ambas se ha asignado el trabajo 2 a la máquina 1 con un tiempo de procesamiento de 4 unidades, en la Figura 3 se ha asignado el mismo trabajo a la otra máquina M2 generando el mismo tiempo de procesamiento. De igual forma sucede con el resto de trabajos. En cualquiera de las dos opciones, el *makespan* es igual a 10 (instante de finalización de la última tarea). Es por esto por lo que en este problema solo importa la asignación de trabajos en cada máquina, siendo estas máquinas indistinguibles.

2.2 Máquinas paralelas uniformemente relacionadas

Este problema añade una variación con respecto al anterior. En este aspecto las máquinas siguen siendo relacionadas, pero no totalmente. Es decir, no son máquinas idénticas. Sin embargo, mantienen una cierta relación. El tiempo de procesamiento de los trabajos depende del trabajo en sí y de un factor de velocidad asociado a cada máquina. Por ejemplo, una máquina es el triple de rápida que otra para todos los trabajos. Todos los trabajos deben de guardar la misma relación sean procesados en una máquina u otra. En la Tabla 3 se especifican los tiempos de proceso de cada trabajo (J1, J2, J3, J4) en cada máquina (M1, M2):

	M1	M2
J1	9	3
J2	3	1
J3	6	2
J4	3	1

Tabla 2: Tiempos de proceso para dos máquinas paralelas relacionadas

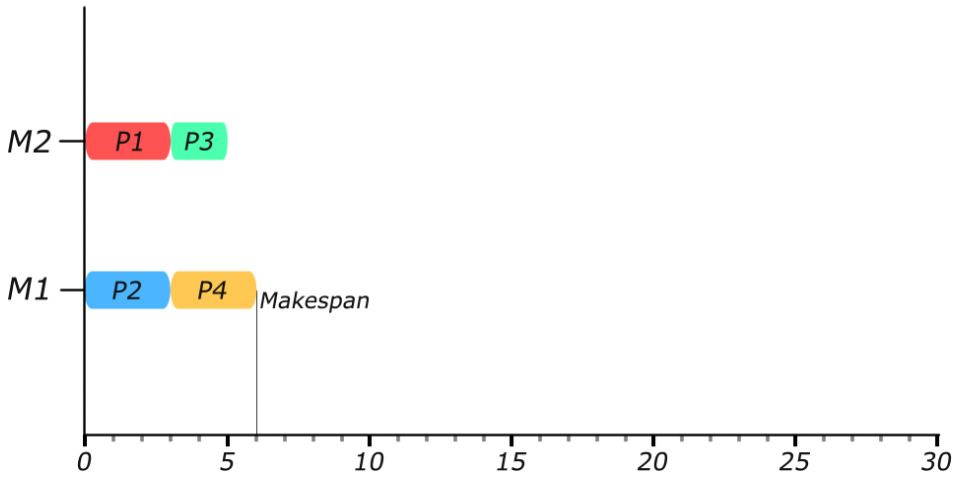

Figura 4: Ejemplo de ejecución con dos máquinas paralelas relacionadas

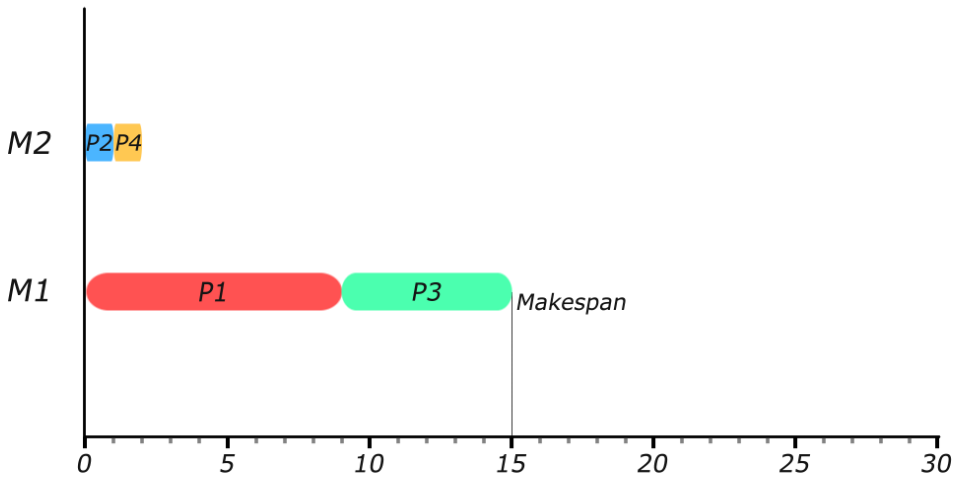

Figura 5: Ejemplo de ejecución con dos máquinas paralelas relacionadas

En las Figuras 4 y 5 se muestran dos ejecuciones con asignaciones diferentes para el mismo problema. En la primera Figura de las dos mostradas, la 4, se ha asignado el trabajo 1 a la máquina 2 causando un tiempo de procesamiento de 3 unidades, mientras que en la segunda ejecución se ha asignado el mismo trabajo a la otra máquina disponible originando un tiempo el triple mayor, 9. La relación entre las dos máquinas es que la segunda procesará cualquier trabajo el triple de rápido que la primera. En la Figura 4 se han realizado mejores elecciones para minimizar el tiempo de ejecución que en la Figura 5, generando un *makespan* de 6 frente a uno de 15, aun habiendo asignado el mismo número de trabajos a cada máquina en ambas asignaciones.

2.3 Máquinas paralelas no relacionadas

Es exactamente igual que el primer problema solo que en este caso las máquinas paralelas no están relacionadas de ninguna manera, por lo que los tiempos de proceso de un mismo trabajo en dos máquinas diferentes no guardarán ningún tipo de relación. Es decir, el tiempo de proceso de cada trabajo depende de la máquina asignada, sin un factor de velocidad por máquinas tal y como ocurría en el caso de las máquinas relacionadas. (En este caso también será importante tener en cuenta donde se va a asignar cada trabajo, porque podría afectar al tiempo final de ejecución.). En la Tabla 3 se muestran los valores de procesamiento de cada trabajo J1, J2, J3 y J4 para las máquinas M1 y M2:

	M1	**M2**
J1	4	7
J2	5	3
J3	7	6
J4	2	9

Tabla 3: Tiempos de proceso para dos máquinas paralelas no relacionadas

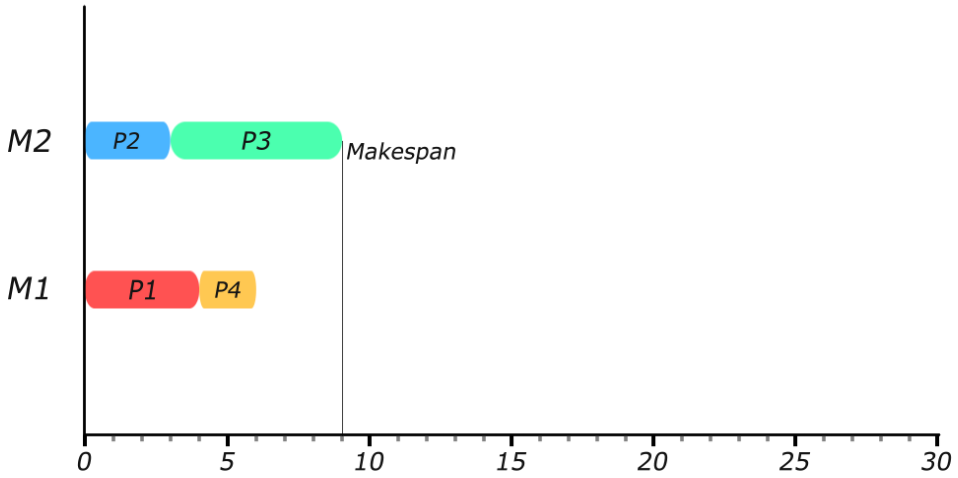

Figura 6: Ejemplo de ejecución con dos máquinas paralelas no relacionadas

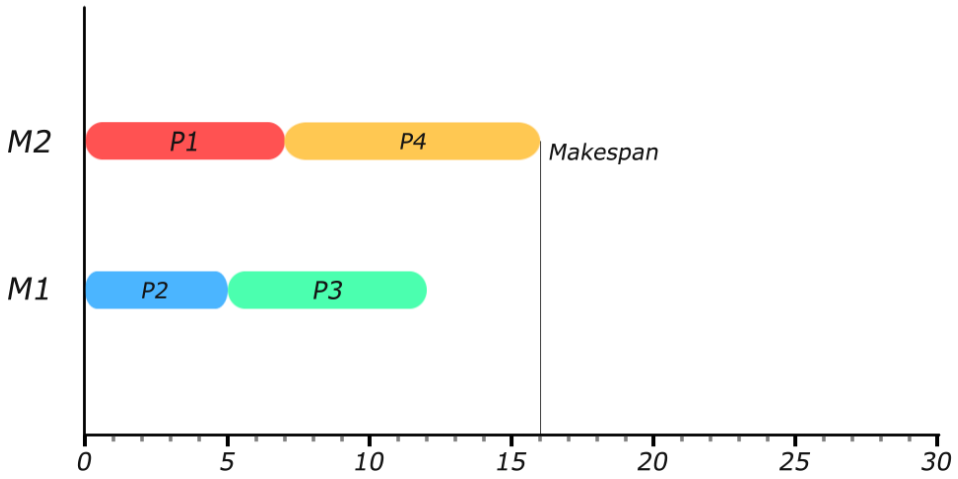

Figura 7: Ejemplo con dos máquinas paralelas no relacionadas

En las Figuras 6 y 7 se muestran dos asignaciones en las que se aprecia como en la primera de ellas, el trabajo J1 se ha asignado a la máquina M1 generando un tiempo de procesamiento de 4, mientras que en la segunda asignación se ha asignado a la otra máquina M2, obteniendo un tiempo de ejecución de 7, siendo este último peor desde el punto de vista del *makespan* y por tanto más largo que el de la máquina M1. Lo contrario sucede con el trabajo J2, que en la máquina M1 genera 5 unidades en la segunda asignación respecto a 3 en la primera donde se asigna a la máquina M2, corroborando la no relación de las máquinas. El *makespan* en cada asignación es de 9 para la Figura 6 y 16 para la siguiente.

2.4 Máquinas paralelas con tiempo de ajuste dependiente de secuencia

En los dos tipos de problemas restantes por explicar, los tiempos de procesamiento dependerán de la máquina siempre, así que se presentará un único ejemplo para ambos puesto que será relevante la elección de la máquina y ésta no tendrá menos influencia como en los ejemplos anteriores.

A este tipo de problema (UPMS: *Unrelated Parallel Machines scheduling problem with Setups*) se suma una complejidad importante con respecto al anterior. Para este caso, antes de procesar cada trabajo en la máquina correspondiente se necesitará de una fase de ajuste previa para preparar dicha máquina antes de su procesamiento. El tiempo de configuración de esta fase dependerá de dos factores: del trabajo previo que se haya realizado en la máquina donde se va a procesar, y de la máquina donde se vaya a llevar a cabo el procesamiento. Los tiempos de ajuste para el mismo trabajo no están relacionados entre diferentes máquinas ya que también son dependientes del trabajo anterior. Es decir, los tiempos de ajustes son dependientes tanto de la máquina como de la secuencia.

En los problemas anteriormente definidos (sin tiempos de ajuste), sólo hay que decidir la asignación trabajo-máquina, ya que el orden en que se procesen los trabajos dentro de cada máquina es irrelevante para el *makespan*. Sin embargo, al haber tiempos de ajuste para las máquinas entre el procesado de dos trabajos, ahora también habrá que decidir la secuencia en que se procesen los trabajos en cada máquina. Pasando al ejemplo, en la Tabla 4 se muestran los tiempos de proceso para las dos máquinas, M1 y M2. En la Tabla 5 (para la máquina M1) y siguiente (M2) se especifican los tiempos de ajuste requeridos para cada trabajo. Cada fila representa el trabajo anterior, y cada columna el actual, por ejemplo, la combinación J4-J3 en la máquina M1 poseería un tiempo de ajuste de 4, es decir, si la máquina 1 procesa el trabajo 4 y el siguiente que procesa es el 3 necesita un tiempo de ajuste de 4 unidades.

	M1	**M2**
J1	4	2
J2	5	6
J3	6	4
J4	8	4

Tabla 4: Tiempos de proceso para dos máquinas paralelas no relacionadas

M1	J1	J2	J3	J4
J1	2	7	4	5
J2	5	2	6	9
J3	8	3	2	7
J4	8	8	4	2

Tabla 5: Tiempos de ajuste para la máquina 1

M2	J1	J2	J3	J4
J1	8	8	6	5
J2	10	4	4	8
J3	3	3	1	7
J4	4	2	8	4

Tabla 6: Tiempos de ajuste para la máquina 2

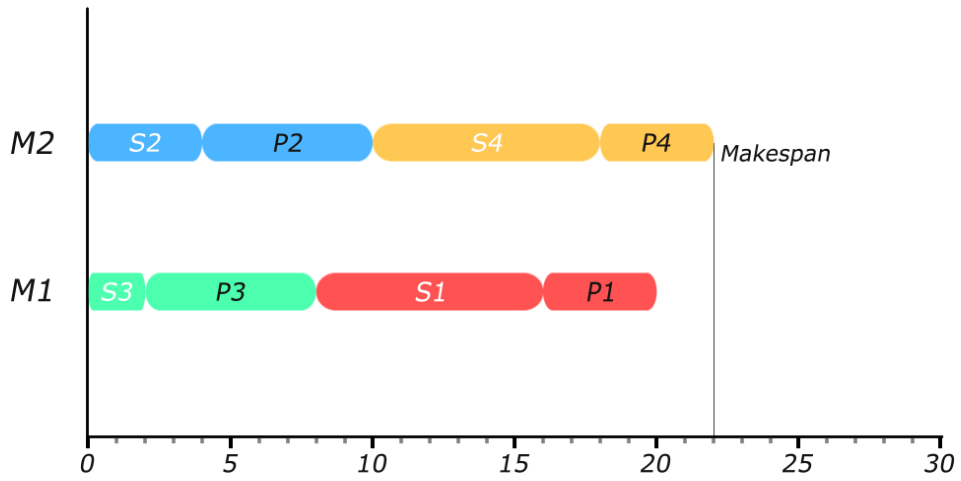

Figura 8: Ejemplo de ejecución con dos máquinas paralelas no relacionadas con tiempos de ajuste

En este ejemplo se puede apreciar ya visualmente que en las máquinas hay dos fases: el tiempo de configuración (SJ) y el de procesamiento (PJ), siendo el de configuración siempre predecesor del de procesamiento.

En la Figura 8 se muestra una posible solución teniendo en cuenta los datos de entrada. En esta figura se puede apreciar la asignación de los trabajos J3 y J1 a la máquina M1 y los J2 y J4 a la máquina M2. Por tanto, para J3 generará 2 unidades de tiempo de ajuste (la diagonal de la matriz está formada por los tiempos de ajuste iniciales, dependiendo del problema estos pueden existir o no) y 6 de procesamiento. A continuación, habrá que preparar la máquina para el siguiente trabajo, J1. Se busca en la tabla de secuencias de la máquina 1 la combinación fila (trabajo anterior) y la columna (trabajo actual) para obtener el valor de 8, que representará el tiempo de ajuste para la secuencia J3-J1 en la máquina M1. Este irá seguido del tiempo de procesamiento que tendrá J1 que será 4 y sumará un total de 20 de *makespan* en esta máquina.

2.5 Máquinas paralelas con tiempo de ajuste dependiente de la secuencia y recursos limitados para ajuste y procesamiento

Por último, este último tipo de problema será el que se tratará en este trabajo y el más complejo. Aparte de todo lo anterior este tendrá una limitación añadida: se precisará de unos recursos adicionales tanto para procesar trabajos como para ajustar máquinas. Los recursos serán por separado para los ajustes y procesamientos, y ambos serán dependientes de la máquina donde se procesen los trabajos y de cada trabajo en sí. En el caso de los requeridos para el ajuste también serán dependientes de la secuencia (teniendo en cuenta el trabajo anterior realizado en esa máquina).

Que los recursos estén limitados podrá provocar paradas en la ejecución, ya que, si en algún punto de todo el proceso no se disponen de los recursos suficientes para realizar una tarea, se deberá de esperar y parar la línea de ejecución en esa máquina hasta que se liberen recursos por la finalización de otro trabajo en otra máquina, y así poder proseguir cuando haya suficientes. Para esto existirán dos variables $R^{s}_{máx}$ y $R^{p}_{máx}$ que determinarán el número máximo de recursos que se pueden utilizar para trabajos de ajuste y de procesamiento, respectivamente. Los valores que posean estas dos variables no podrán ser superados en ningún instante de tiempo de la ejecución.

En el siguiente ejemplo se utilizan los datos del ejemplo anterior, pero añadiendo las tablas correspondientes a los recursos necesarios para cada fase. Primero se han especificado como 5 unidades el número máximo de recursos que se pueden utilizar para ajuste o procesamiento en cualquier instante de tiempo. A continuación, en la Tabla 7 se mostrarán los recursos necesarios para procesamiento que no dependerán de la secuencia. Por último, en las Tablas 8 y 9 se muestran los valores para los recursos necesarios que precisarán los trabajos de ajuste de las máquinas M1 y M2, respectivamente. Para acceder al valor de cada trabajo se seguirá el mismo patrón que con los tiempos de ajuste. Por ejemplo, si se produce la secuencia de trabajos J1-J4 en la máquina M2, la fila será la correspondiente al trabajo J1, y la columna la correspondiente al actual J4, requiriendo unos recursos de ajuste de 1 unidad.

$$R^{S}_{máx} = 5$$

$$R^{p}_{máx} = 5$$

	M1	M2
J1	3	1
J2	4	4
J3	1	2
J4	1	3

Tabla 7: Recursos necesarios de procesamiento para problema UPMSR

M1	J1	J2	J3	J4
J1	1	2	1	4
J2	4	2	2	3
J3	4	4	2	2
J4	5	1	2	2

Tabla 8: Recursos de ajuste para la máquina 1 del problema UPMSR

M2	J1	J2	J3	J4
J1	2	3	1	1
J2	3	2	5	1
J3	4	1	2	2
J4	1	3	5	1

Tabla 9: Recursos de ajuste para la máquina 2 del problema UPMSR

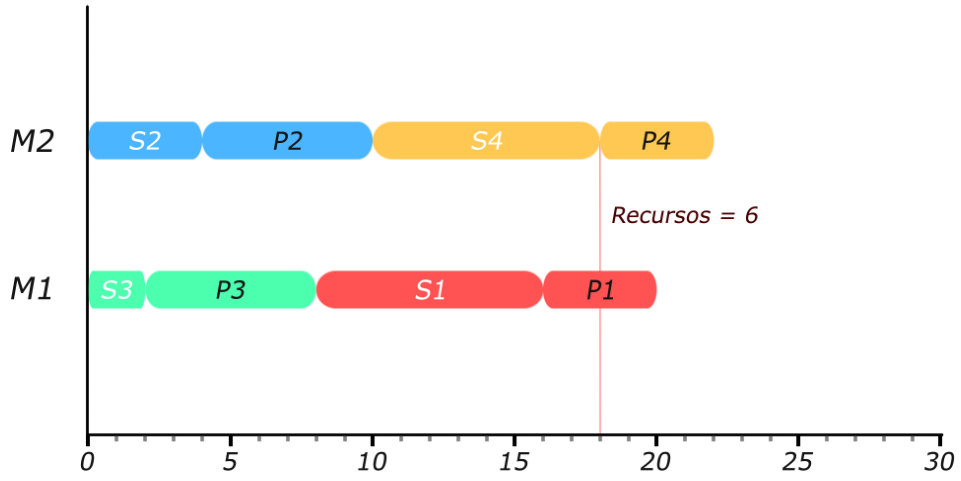

Figura 9: Solución inválida para el problema UPMSR

Siguiendo la ejecución anterior de la Figura 9 se puede observar como no se superan los recursos (realizando las pertinentes comprobaciones en la tabla) hasta que se llega al punto 18 de la ejecución donde se están ejecutando al mismo tiempo el trabajo J4 en la máquina M2 y el trabajo J1 en la máquina M1, donde se está sobrepasando el límite de recursos de procesamiento, ya que el trabajo J4 requiere de 3 unidades de recursos y el J1 de 3, sumando 6 en total y superando el límite establecido en 5 para los trabajos de procesamiento.

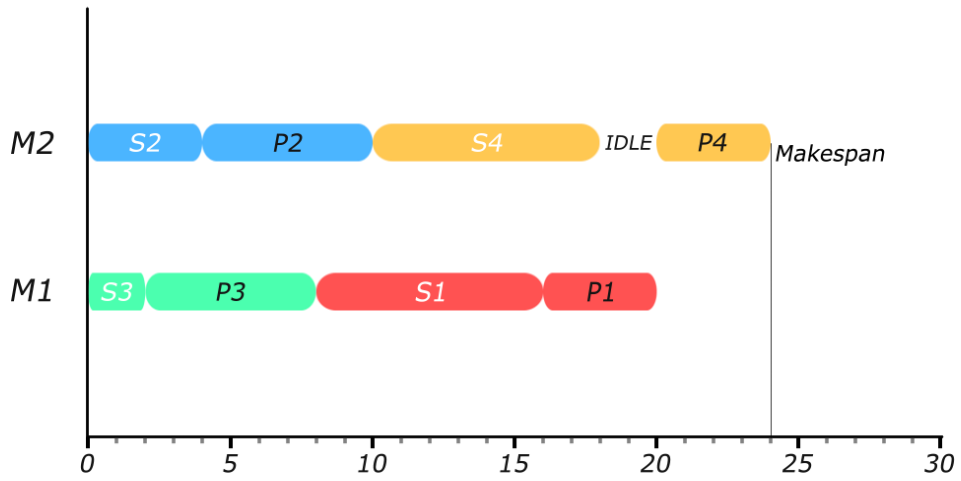

Figura 10: Solución 1, válida para el problema de UPMSR

Por tanto, existirán dos movimientos posibles para respetar el límite de los recursos de procesamiento y que así ambos trabajos puedan ser llevados a cabo. Una opción es retrasar el trabajo 4 en su fase de procesamiento de la máquina 2, hasta que termine el trabajo P1, tal y como se puede observar en la Figura 10. Entonces una vez termine la máquina 1 de procesar dicho trabajo ya se dispondrán de suficientes recursos para llevar a cabo P4, que cuando termine generará un *makespan* de 24.

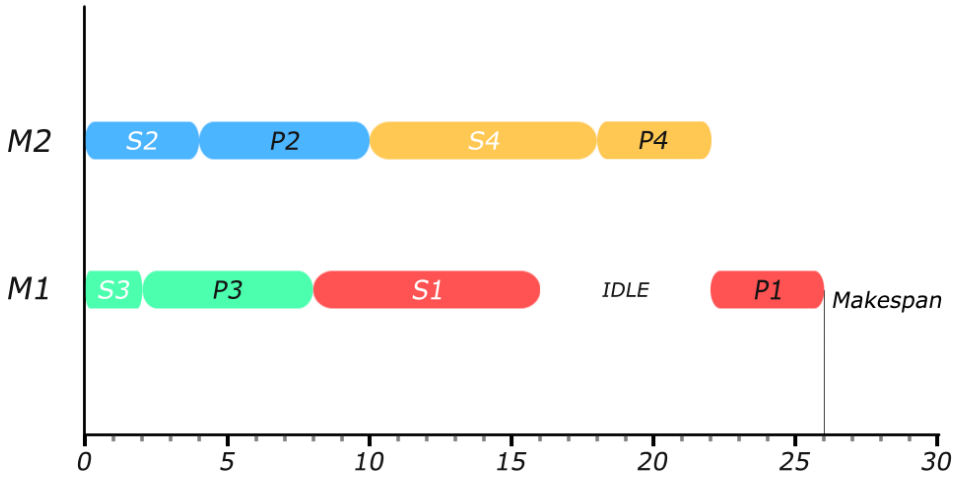

Figura 11: Solución 2, válida para el problema UPMSR

La otra opción, visible en la Figura 11, es atrasar el trabajo J1 de la otra máquina M1, este trabajo se deberá retrasar hasta el punto 22, que será cuando el trabajo J4 de la otra máquina habrá terminado y ya se poseerán de recursos de procesamiento suficientes para realizar el procesamiento de J1. Una vez haya terminado se habrá generado un *makespan* de 26, frente a uno de 24 de la asignación anterior. Por ello la primera asignación correspondiente a la Figura 10 será la mejor desde el punto de vista del *makespan*.

3. Métodos de resolución

En este trabajo se introducen y proponen varios métodos de resolución que se utilizan para obtener soluciones de tipos de problemas como los explicados en el apartado anterior. Algunos de estos métodos pueden ser: MILP (Mixed integer linear program, o Programación Lineal Entera en castellano), greedy (heurística constructiva) o GRASP (metaheurística constructiva aleatorizada). En esta sección se tratará de dar una explicación clara de estos métodos de resolución.

3.1 Programación Lineal Entera

La programación matemática lineal entera (MILP por sus siglas en inglés), es una técnica de modelado matemático dedicada a resolver los problemas de optimización, en donde se encontrará la mejor solución de todas las existentes, o problemas con restricciones, en los que se definirá un conjunto de reglas/restricciones que representen el problema a resolver para obtener una solución.

Un MILP se caracteriza por tener variables tanto enteras como continuas, y funciones implicadas lineales. La programación no lineal es mucho más costosa computacionalmente, por lo que se intenta siempre modelar los problemas utilizando funciones lineales.

Aparte de las restricciones, estos problemas tendrán una función objetivo, que será minimizar o maximizar un valor numérico. Por ejemplo, en el caso del problema a resolver en este trabajo este valor numérico será minimizar el tiempo de finalización del último trabajo que se procese, el *makespan*.

La programación entera está catalogada del tipo NP-completa (NP: non deterministic polynomial time, son las siglas que se asocian a los tipos de problemas que no se pueden resolver en tiempo polinómico). Esto significa que se puede verificar si la solución encontrada es la óptima, pero no existe un algoritmo eficiente para encontrar esta solución óptima. Es aquí donde toman importancia el diseño e implementación de heurísticas y metaheurísticas.

3.2 Heurísticas

El segundo método de resolución son las heurísticas. Una heurística es una técnica específica para resolver los problemas de optimización de forma eficiente, aunque no pueden asegurar optimalidad. Por lo tanto, pueden servir para cuando los métodos exactos no puedan encontrar la solución óptima y la heurística pueda obtener una solución cercana a la mejor y utilizando mucho menor tiempo computacional, que es su principal ventaja y el principal inconveniente de los modelos o métodos exactos.

El diseño de una heurística se basa en una especificación de reglas a seguir para encontrar la solución. Varios ejemplos de estas reglas pueden ser: escoger el mejor trabajo en cada paso, seleccionar el peor, ordenar los trabajos del mejor a peor y elegir uno de los primeros aleatoriamente en cada iteración, etc.

Una heurística suele ser dependiente del problema, ya que se diseñan e implementan para resolver un problema concreto. A diferencia de estas, las metaheurísticas suelen ser técnicas independientes al tipo de problema con unas ideas o pasos a seguir más generales que pueden ser aplicadas a un espectro de problemas mucho mayor que una heurística, que está más enfocada a uno en concreto.

3.3 Algoritmo greedy

Un algoritmo greedy es aquel en el que en cada estado de búsqueda de la solución, siempre se seleccione la mejor opción local, con el objetivo de que la elección de la mejor opción en cada iteración pueda asegurar una solución global buena. En castellano se les conoce como algoritmos voraces o algoritmos miopes.

Esto se puede ver con el ejemplo del viajante. Este viajante deberá visitar tres ciudades, por ejemplo, pero deberá hacerlo recorriendo la cantidad menor posible de kilómetros empezando en Valencia. Las distancias se pueden observar en la Tabla 10.

	Barcelona	Madrid	Valencia	Zaragoza
Barcelona		624	350	290
Madrid	624		355	320
Valencia	350	355		300
Zaragoza	290	320	300	

Tabla 10: Distancias en kilómetros entre las cuatro ciudades

Si se sigue un algoritmo greedy, se debería viajar entre ciudades escogiendo cada vez que se llegue a una ciudad la que más cerca esté de ésta como siguiente destino. Por esto, el recorrido del viajero sería: Valencia - Zaragoza (300), Zaragoza - Barcelona (290) y Barcelona - Madrid (624) sumando un total de 1214 km. Sin embargo, si se traza el recorrido en un mapa no parece ser un recorrido muy interesante a pesar de que se ha elegido siempre la mejor opción “de forma miope”. Si se escoge por ejemplo la ruta: Valencia - Barcelona (350), Barcelona - Zaragoza (290) y Zaragoza - Madrid (320) se obtendrán 960 km, mejorando bastante la solución obtenida por el algoritmo greedy.

Hay posibilidades de que la solución encontrada pueda ser la óptima pero no lo garantiza de ningún modo. Este tipo de algoritmo suele realizar una búsqueda miope, ya que solo tiene en cuenta una combinación en cada selección del mejor candidato, no mira más allá, si las siguientes opciones después de escoger la mejor en el paso actual serán mejores o peores.

Su gran ventaja es que con un tiempo muy inferior respecto al necesario para que un MILP o un método exacto obtengan la solución óptima, este algoritmo pueda encontrar una solución muy aproximada. Es decir, son algoritmos rápidos pero en general no dan una buena calidad de solución. De hecho, en algunos problemas el algoritmo greedy encuentra la peor solución posible.

3.4 Algoritmo GRASP

Un algoritmo GRASP (*Greedy Randomized Adaptative Search Procedure)* está basado en el algoritmo greedy visto en el apartado anterior. La diferencia es que la elección del mejor trabajo en cada paso ya no será siempre el mejor, esta vez la elección que se realice será aleatoria de entre un conjunto específico que contenga los mejores candidatos. Los algoritmos tipo GRASP se han usado para encontrar soluciones de forma eficiente a multitud de problemas de optimización combinatoria.

El tamaño del conjunto que contiene los mejores candidatos puede ser variable, y a dicho conjunto se le suele denominar RCL (*Restricted candidates list*). La variabilidad de las soluciones obtenidas con este algoritmo dependerá del tamaño del conjunto de los mejores candidatos. Cuanto más grande sea, más dispares serán las soluciones obtenidas.

La ejecución del algoritmo completo GRASP se podrá repetir un número concreto de veces determinado por el tiempo disponible, para que cada vez el algoritmo pueda obtener una solución diferente, y así una vez se haya terminado el tiempo quedarse con la mejor opción.

Si por ejemplo se aplica este algoritmo al ejemplo anterior con un RCL=2, sí que existen posibilidades de obtener la solución óptima (existen dos en este caso) a diferencia de el algoritmo greedy teniendo en cuenta el planteamiento actual del problema. En la Figura 12 se podrían ver los diferentes caminos que podría tomar la ejecución si el conjunto de los mejores candidatos estuviera formado por los dos mejores trabajos en cada paso:

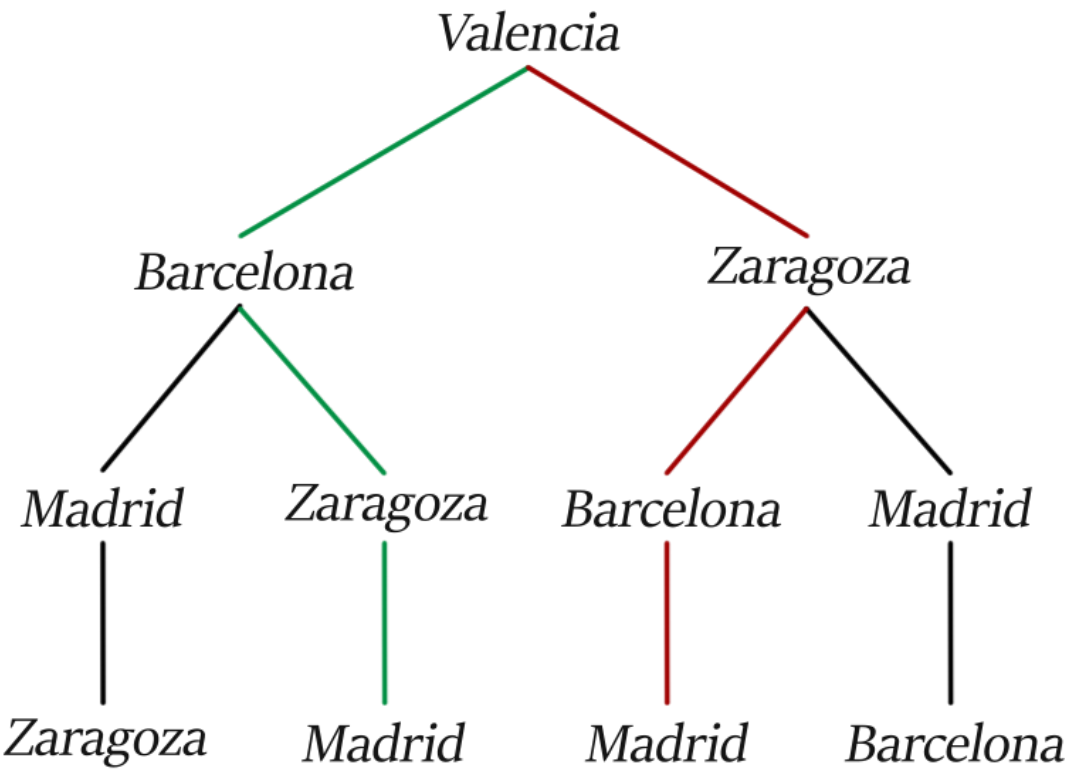

Figura 12: Posibles caminos de la ejecución del problema ejemplo

Se puede apreciar en la anterior figura como está marcado de rojo el camino que se seguiría si se ejecutara el algoritmo greedy, y de verde una posible solución del algoritmo GRASP mostrando la utilidad del factor de aleatorización para acercarse o encontrar soluciones óptimas.

Es común complementar la fase constructiva de un algoritmo GRASP (lo que se ha explicado anteriormente) con una fase de búsqueda local. Sin embargo, en este trabajo solo se propondrá la fase constructiva.

4. Estado del arte

En este apartado se realizará un repaso de algunos artículos que tratan con la secuenciación de máquinas paralelas y necesidad de recursos, publicados en la literatura. La mayoría de los trabajos estarán relacionados con versiones explicadas en el apartado anterior que no son la que se lleva a cabo en este trabajo, ya que la última versión corresponde a un problema novedoso y del que no hay heurísticas ni mucha bibliografía actualmente. Aunque también hay artículos con otras variaciones posibles que pueden presentar los problemas de secuenciación. El orden de enumeración de los trabajos estará basado en el tiempo.

Ruiz-torres, López y Ho (2007) investigan el problema de la programación de trabajos en un conjunto de máquinas paralelas donde la velocidad de estas depende de unos recursos secundarios. Estos recursos son fijos en cantidad y necesitan ser asignados a las máquinas al inicio de la ejecución. La función objetivo de este problema será minimizar el número de trabajos tardíos. Existirán dos versiones del mismo problema: una en la que los trabajos estén ya preasignados a las máquinas y la segunda en la que no lo estarán y deberán ser asignados en la ejecución. Para resolver ambos problemas se propondrá un modelo de programación matemática entera (*Integer programming formulation*) y una heurística, respectivamente para el primer y segundo problema.

Ruiz y Romano (2011) proponen un modelo de programación lineal entera para resolver el problema de las máquinas paralelas no relacionadas con tiempos de ajuste para trabajos dependientes de la secuencia de trabajos, la máquina donde se procesen y con la complejidad añadida de que estos tiempos también dependen de la cantidad de recursos asignados. En consecuencia, un tiempo de ajuste para un trabajo podrá ser mayor o menor según los recursos que tenga asignados. La función objetivo será una combinación lineal entre el tiempo de finalización (*makespan*) y la cantidad total de recursos asignados. Este problema está enfocado a las plantas industriales donde son frecuentes los trabajos de ajustes en las cadenas de producción.

Edis y Oguz (2012) tratan el problema de las máquinas paralelas con recursos flexibles limitados, como por ejemplo operarios. Flexibles significa que pueden ser asignados libremente a cualquier máquina y realizar cualquier trabajo para así agilizar lo máximo posible la ejecución. También trabajarán la variación del problema en el que los trabajos no están preasignados a cada máquina. Para ambos de estos problemas se propone un modelo de programación entera con el objetivo de minimizar el *makespan*. Para resolver problemas que posean instancias grandes se presenta también un modelo de programación entera con restricciones.

Edis, Oguz y Ozkaraham (2013) trabajan el problema de las máquinas paralelas no relacionadas con recursos limitados adicionales. Se basan en que la mayoría de las soluciones obtenidas hasta este momento se fijan en las máquinas como el único recurso necesario para procesar trabajos obviando que en el mundo real se puedan necesitar otros muchos tipos de recursos como herramientas, operadores de máquinas, robots, etc. Primero muestran algunos artículos de soluciones en la

literatura que traten este tipo de problemas exponiendo sus debilidades y fortalezas, así como líneas futuras de continuación para estos trabajos. Y por último también presentan dos extensiones de modelos de programación entera para los dos problemas relacionados con este y se dan unas conclusiones basadas en los experimentos realizados.

Fanjul, Perea y Ruiz (2017) tratan el problema de la secuenciación de máquinas paralelas en el cual el procesamiento de los trabajos requiere de un número de recursos escaso. Este número depende tanto del trabajo como de la máquina donde se procese. Además, la disponibilidad de estos recursos está limitada y fijada durante toda la ejecución. El objetivo minimizar el *makespan,* el tiempo de finalización del procesamiento del último trabajo. Para abordar el problema se implementan dos modelos de programación lineal entera. Uno de ellos está basado en una implementación anterior realizada en la literatura científica, y el otro está basado en los problemas de embalaje de paquetes, siendo este último un modelo original. Además, como los dos modelos presentados no son capaces de resolver instancias de tamaño medio debido al tiempo de cómputo necesario, se proponen tres matheurísticas para cada uno de los dos modelos. Una matheurística es un algoritmo metaheurístico que utiliza en alguno de sus pasos un modelo de programación matemática. Todos los algoritmos son probados exhaustivamente en una fase de pruebas. La conclusión es que las matheurísticas mejoran significativamente los resultados de los modelos matemáticos.

Villa, Vallada y Fanjul-Peyro (2017) proponen una heurística para resolver el problema de máquinas paralelas no relacionadas con un recurso adicional escaso. Se presentan varias heurísticas que siguen dos estrategias para minimizar el *makespan* final. La primera está basada en la restricción del uso de los recursos durante toda la ejecución y la segunda tiene en cuenta varias reglas de asignación sin considerar la restricción de los recursos, pero tiene una fase de reparación para poder obtener una solución factible. También se realizarán una serie de experimentos comparando modelos matemáticos y heurísticas propuestos anteriormente para concluir que estos modelos propuestos mejoran los resultados de los comparados y con la segunda estrategia se consigue mejorar el rendimiento para las instancias grandes.

Arbaoui y Yalaoui (2018) presentan un artículo que trata el problema de las máquinas paralelas no relacionadas con recursos adicionales (UPMR: Unrelated parallel machine schedulling with additional resources). El tiempo de proceso y número de recursos de cada trabajo depende de la máquina donde se procesa. Los recursos tienen una cantidad fija y se pueden utilizar varias veces durante la ejecución mientras estén libres. La función objetivo es minimizar el *makespan* final. Para resolverlo se utiliza un modelo de restricciones utilizando un *solver* (programa para resolver un problema definido y modelado). Una vez obtenidas las soluciones se comparan los resultados con aproximaciones a este problema de anteriores trabajos dividiendo el análisis en instancias de tamaño pequeño y tamaño mediano. En las instancias pequeñas, el modelo propuesto mejora los modelos que se han realizado anteriormente ajenos a este artículo. Por otro lado, con las instancias de tamaño

medio se muestra que obtiene más veces la solución óptima respecto a cualquiera de los otros artículos parecidos.

Vallada, Villa y Fanjul-Peyro (2019) proponen un algoritmo de búsqueda dispersa junto a un algoritmo greedy para resolver el problema de las máquinas paralelas con un recurso adicional. El objetivo vuelve a ser minimizar el *makespan,* el tiempo final donde habrá terminado el procesamiento del último trabajo. Estos dos algoritmos están basados en las mejores heurísticas realizadas para resolver este problema. Para resolverlo se permite que la solución no sea factible porque existe una fase posterior de reparación para que así sea respetando las restricciones de los recursos. Anteriormente se realizan diferentes variaciones de una búsqueda local para obtener las mejores asignaciones. Al final de las ejecuciones también existirá una fase de experimentación donde se concluirá que el algoritmo greedy mejora notablemente los resultados para otras heurísticas con el mismo problema como objetivo.

Yepes-Borrero, Villa, Perea y Caballero-Villalobos (2019) tratan el problema de secuenciación de máquinas paralelas no relacionadas con tiempo de ajuste dependientes de secuencia y recursos limitados para los tiempos de ajuste, con el mismo objetivo de minimizar el *makespan.* Se presenta como un problema realista ya que considera que los tiempos de ajuste precisan de recursos como trabajadores para poder ser llevados a cabo. Se propondrán tres metaheurísticas siguiendo dos estrategias, la primera ignora información acerca de los recursos en la fase constructiva del algoritmo y la segunda sí que tiene en cuenta esta información en la primera fase. Se han llevado a cabo experimentos para instancias pequeñas y grandes que han determinado que la segunda estrategia mejora y muestra un rendimiento excelente respecto a la primera propuesta.

Fanjul-Peyro (2020) introduce por primera vez el problema de las máquinas paralelas no relacionadas con tiempos de ajuste dependientes de la secuencia de trabajos y de la máquina, y recursos limitados de ajuste, procesamiento y compartidos (los recursos compartidos son lo que se pueden utilizar para ambos procesos). Se presenta un modelo matemático lineal para modelar un problema y un algoritmo de tres fases basado en un método matemático exacto. Tanto el modelo como el algoritmo han sido probados exhaustivamente y el análisis de los resultados ha mostrado soluciones muy cercanas a las óptimas.

En el problema presentado en este último artículo es el cual en el que está basado este TFM.

5. Definición formal del problema

Una vez explicados los diferentes tipos de problemas y repasada la bibliografía relacionada, se pasará a realizar una definición más detallada del problema preciso a tratar en este trabajo: secuenciación de máquinas paralelas no relacionadas con tiempos de ajuste y necesidad de recursos adicionales. Para ello se definirán todas las partes del problema con notación matemática, primero se enumerarán los conjuntos, parámetros y variables necesarias, y por último las restricciones. Este modelo fue presentado en la referencia *Fanjul-Peyro 2020*.

Conjuntos:

- Conjunto de n trabajos llamado $N = \{1, \dots, n\}$, utilizando j y k para indexar los trabajos. Se ampliará dicho conjunto a $N_0 = \{0, 1, \dots, n\}$ siendo el 0 un trabajo ficticio necesario para modelar el problema.
- Conjunto de m máquinas llamado $M = \{1, \dots, m\}$ utilizando i para indexar las máquinas.

Parámetros:

- p_{ij}: tiempo de procesamiento del trabajo j en la máquina i.
- s_{ijk}: tiempo de configuración de la máquina i tras procesar el trabajo j si el trabajo a procesar después es el k.
- $R^S_{máx}$: número máximo de recursos que se pueden utilizar exclusivamente para tareas de ajuste.
- $R^P_{máx}$: número máximo de recursos que se pueden utilizar únicamente para tareas de procesamiento.
- r^p_{ij}: recursos necesarios de procesamiento para procesar el trabajo j en la máquina i.
- r^s_{ijk}: recursos necesarios para realizar el ajuste en la máquina i del trabajo k sucesor del trabajo j.

Una vez definidos los datos de entrada del problema, se realizará el modelado utilizando programación matemática. Más concretamente, se definirá un modelo de programación lineal entera mixta (una referencia a este modelo de programación se puede encontrar en el libro "*Operations Research: A Model-Based Approach. (H. A. Eiselt, Carl-Louis Sandbloom), 2012)*".

Para este tipo de modelos, se han de definir las variables (decisiones que se pueden controlar), la función objetivo (lo que se quiere hacer), y las restricciones (lo que hay que hacer, impuesto por el problema).

Variables:

- $C_{máx}$: tiempo en el que termina el último trabajo de procesarse, es el makespan.
- Y_{ij}: variable binaria que tendrá valor 1 si el trabajo j se procesa en la máquina i, y 0 en caso contrario.
- X_{ijk}: variable binaria que tomará valor 1 cuando el trabajo j sea predecesor del trabajo k en la máquina i, y 0 en caso contrario.

Un modelo para este problema consiste en:

$$min\ C_{máx} \quad (1)$$

$$\sum_{j\in N_0, k\in N, k\neq j} s_{ijk}X_{ijk} + \sum_{k\in N} p_{ik}Y_{ik} \leq C_{máx}, i \in M \quad (2)$$

$$\sum_{k\in N} X_{i0k} <= 1, i \in M \quad (3)$$

$$\sum_{i\in M} Y_{ik} = 1, k \in N \quad (4)$$

$$Y_{ik} = \sum_{j\in N_0, j\neq k} X_{ikj}, i \in M, k \in N \quad (5)$$

$$Y_{ik} = \sum_{j\in N_0, j\neq k} X_{ijk}, i \in M, k \in N \quad (6)$$

$$C_k \leq C_{max}, k \in N \quad (7)$$

$$R^p_{máx} \geq R^p_k \geq \sum_{i\in M} Y_{ik}r^p_{ik}, k \in N \quad (8)$$

$$R^s_{máx} \geq R^s_k \geq \sum_{i\in M, l\in N} X_{ilk}r^s_{ilk}, k \in N \quad (9)$$

$$X_{ijk} \in \{0,1\}, Y_{ij} \geq 0, C_k \geq 0 \quad (10)$$

La expresión (1) define la función objetivo del problema, que consistirá en minimizar el *makespan* final. La restricción (2) asegura que la suma del tiempo de ajuste y de procesamiento de todos los trabajos siempre será menor o igual que el *makespan*. La restricción (3) especifica que solo puede haber como máximo un trabajo inicial en una máquina. (4) garantiza que un trabajo solo puede asignarse a una única máquina. La restricción (5) expone que solo puede haber un trabajo sucesor a otro en una máquina. (6) asegura un único trabajo como predecesor de otro en una máquina. La ecuación (7) limita el tiempo de finalización de un trabajo siendo este siempre menor al *makespan*.

En cuanto a los recursos, la restricción (8) especifica que el número de recursos de procesamiento de un trabajo siempre va a ser menor o igual al máximo. La definición (9) asegura que el número de recursos de configuración de un trabajo siempre será menor o igual al máximo. (10) especifica el rango de valores válidos para las variables.

Como se verá en los experimentos, el modelo propuesto solo puede resolver instancias del UPMSR de tamaño reducido. Es por ello que en la siguiente sección se proponen algoritmos más eficientes para este problema.

6. Algoritmo propuesto

En este apartado se va a explicar la heurística con detalle y por fases. El anterior modelo consigue resolver el problema obteniendo soluciones óptimas para instancias pequeñas con pocas máquinas y trabajos, pero al aumentar el tamaño deja de ser una opción viable debido al alto coste computacional sin garantizar la mejor solución, dándole mayor valor a la heurística.

La heurística propuesta se puede dividir en dos fases:

1. En la primera fase se construirá una solución teniendo en cuenta que el problema a resolver es un UPMS (máquinas paralelas no relacionadas con tiempos de ajuste dependiente a la secuencia), es decir, sin tener en cuenta los recursos donde se construirá una solución que puede ser definitiva.
2. En la segunda fase se comprobará si la solución es factible, si no se incumplen las restricciones del uso de recursos o de lo contrario habrá que realizar ajustes para que así lo sea.

A la primera fase se le llamará fase constructiva, y a la segunda se le llamará fase de reparación.

6.1 Fase constructiva

La primera fase de esta heurística es constructiva, y en cada iteración que se realice en esta fase se asignará un trabajo a una máquina. La asignación que se realizará será la que suponga un menor *makespan* general, es decir, se buscará la combinación de máquina y trabajo que genere un menor *makespan* teniendo en cuenta el tiempo de ajuste más el tiempo de procesamiento de este trabajo en dicha máquina. Con esto lo que se busca es que siempre se asigne el mejor trabajo, que será el que menos alargará el tiempo de ejecución total. Obviamente esto no garantiza que la asignación sea óptima, ya que por ejemplo si se asigna un trabajo j a una máquina i, puede que todos los trabajos restantes que sucedan al trabajo j en i tengan un coste de ajuste o procesamiento muy alto. Y esto puede pasar con más de un trabajo cuando ya quedan pocos por asignar.

En cada iteración se asignará un trabajo a una máquina hasta que ya no queden trabajos por asignar, será entonces cuando la fase constructiva habrá terminado. Si fuera el caso de que el problema a resolver no tuviera en cuenta los recursos la heurística terminaría aquí. En el Pseudocódigo 1 se muestra el pseudocódigo de esta fase. La variable $C_{mín}$ representará el valor del *makespan* mínimo de la iteración actual (se evita mostrar la parte donde comprueba si es el mínimo en el Pseudocódigo 1 para evitar que sea demasiado verboso). $nTAsig$ será una lista que contendrá los trabajos que queden por asignar, y por último la variable $mAsig$ poseerá una matriz en la que cada fila representará una máquina y cada columna poseerá los trabajos asignados (si existe la asignación, $null$ en caso contrario) en orden de procesamiento. Un ejemplo de esta matriz podría ser el siguiente:

M1	J5	J3	null	null
M2	J2	J1	J4	null

Tabla 11: Ejemplo de matriz de asignaciones de trabajos

$$
\begin{aligned}
&while\ (nTAsig\ !=\ 0)\ do\\
&C_{mín} = \infty\\
&\quad for\ j\ \in\ nTAsig\ do\\
&\quad\quad for\ i\ \in M\ do\\
&\quad\quad\quad C_{mín}\ =\ Sum(C_i)\ +\ s_{ijk}\ +\ p_{ij}\\
&\quad\quad end\\
&\quad end\\
&\quad nTAsig.remove(j)\\
&\quad mAsig.add(i,j)\\
&end
\end{aligned}
$$

Pseudocódigo 1: Fase constructiva del algoritmo

Para un mejor entendimiento de la fase constructiva se va a presentar un ejemplo con 7 trabajos y 3 máquinas. La Tabla 12 muestra los tiempos de procesamiento para la máquina uno, dos y tres. Después de esta, se mostrarán las tres Tablas 13, 14 y 15 con los tiempos de ajustes entre trabajos para las máquinas M1, M2 y M3, respectivamente.

	M1	**M2**	**M3**
J1	5	9	7
J2	7	4	9
J3	5	4	4
J4	8	2	5
J5	6	8	9
J6	4	1	3
J7	3	5	2

Tabla 12: Tiempos de proceso para la explicación del algoritmo con tres máquinas

M1	J1	J2	J3	J4	J5	J6	J7
J1	8	5	2	9	6	2	4
J2	1	3	5	8	7	5	4
J3	4	1	4	5	4	1	3
J4	8	2	8	6	9	1	4
J5	9	8	7	9	2	8	1
J6	2	6	1	6	6	2	9
J7	3	3	2	9	3	5	1

Tabla 13: Tiempos de ajuste para la explicación del algoritmo para la máquina 1

M2	J1	J2	J3	J4	J5	J6	J7
J1	4	7	6	4	3	6	1
J2	6	6	2	4	6	3	7
J3	5	9	3	6	6	6	7
J4	2	8	6	3	5	1	2
J5	9	6	6	9	7	3	7
J6	3	7	6	9	4	2	2
J7	5	1	5	3	8	7	3

Tabla 14: Tiempos de ajuste para la explicación del algoritmo para la máquina 2

M3	J1	J2	J3	J4	J5	J6	J7
J1	5	8	3	7	1	3	3
J2	7	8	6	2	9	2	5
J3	5	3	7	7	4	7	3
J4	2	3	1	3	4	6	2
J5	1	4	2	6	6	3	7
J6	3	7	9	7	4	4	3
J7	1	6	2	6	2	9	7

Tabla 15: Tiempos de ajuste para la explicación del algoritmo para la máquina 3

Como se ha explicado, el primer paso será buscar la combinación del tiempo de configuración más el tiempo de procesamiento para un trabajo en cualquier máquina que tenga el mínimo valor posible. Se realizan todas las pruebas y se obtiene que la asignación del trabajo 6 en la máquina 2 es la que menos aumenta el *makespan*, ya que necesita dos unidades de tiempo de ajuste y una de tiempo de procesamiento, 3 en total. En la Figura 13 se muestra la asignación.

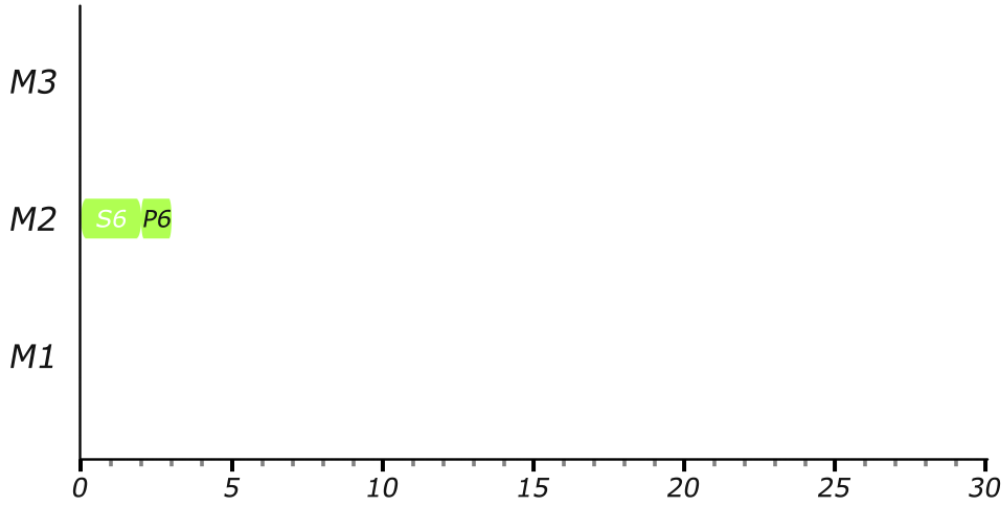

Figura 13: Primer trabajo asignado para el ejemplo explicativo

Ahora que ya ha sido asignado el primer trabajo se tendrá como referencia el valor 3 del *makespan* para realizar todas las comprobaciones siguientes, la asignación que iguale o cuyo aumento sea el menor será la seleccionada. En este caso, el algoritmo obtiene la combinación del trabajo 7 en la máquina 1 que aumenta en 1 unidad el valor anterior del *makespan*, siendo ahora de 4 como se puede apreciar en la siguiente Figura 14:

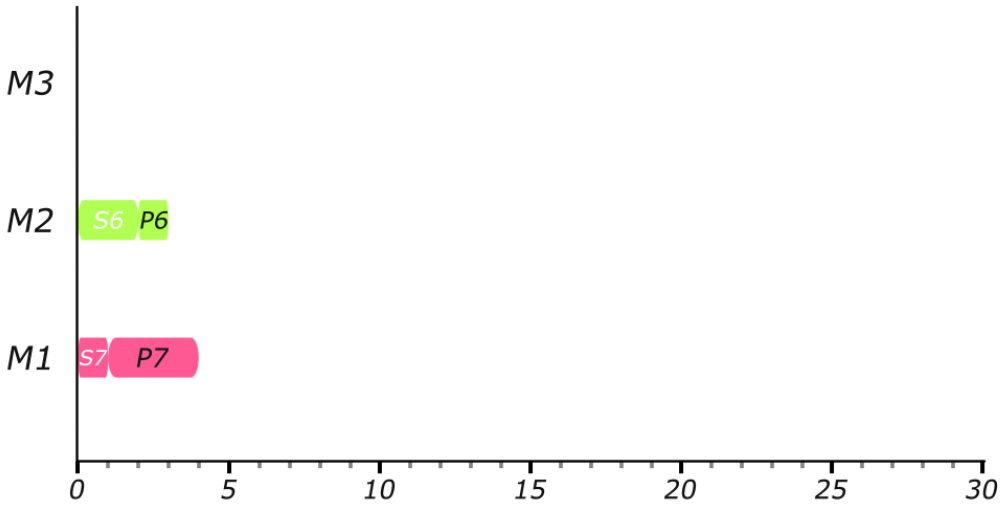

Figura 14: Segundo trabajo asignado para el ejemplo explicativo

Este proceso de la fase constructiva se continuará hasta que no quede ningún trabajo por asignar tal y como se ha comentado. Una vez todos los trabajos tengan una máquina asignada se habrá obtenido una solución para la fase constructiva con 21 de *makespan*. El resultado es la solución que se muestra en la siguiente Figura 15 cuyo orden en el tiempo de las asignaciones de los trabajos ha sido: J6 (M2) - J7 (M1) - J4 (M3) - J3 (M1) - J2 (M2) - J1 (M3) - J5 (M1).

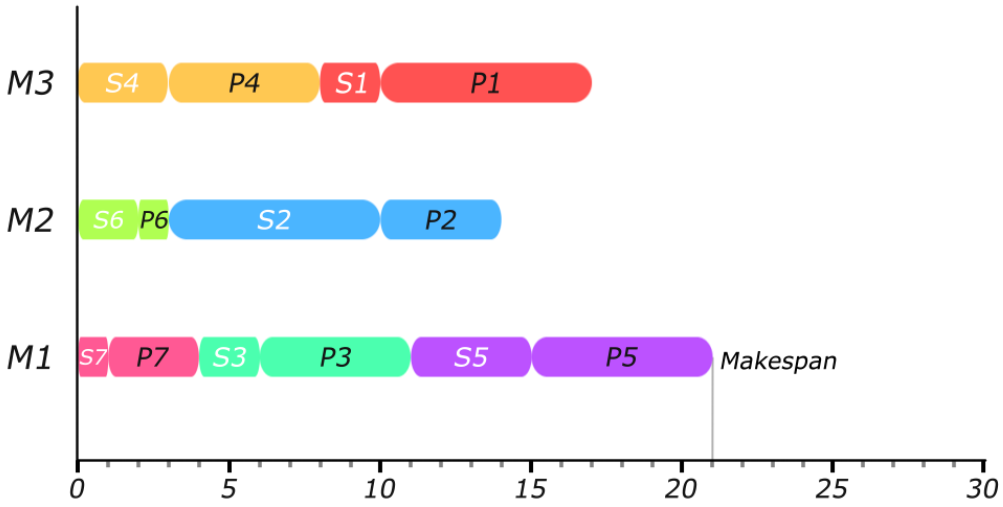

Figura 15: Resultado fase constructiva

6.2 Fase de reparación

La fase de reparación tratará de analizar la solución obtenida y de revisar si hay algún instante a lo largo de la ejecución en el que se superen los recursos. Si no existe ningún instante de tiempo en el que se supere el uso de recursos disponibles, significa que la solución es factible y la heurística terminaría. En el caso de que se superen los recursos en algún instante de tiempo, se procederá a modificar la solución. Para ello, si por ejemplo en el instante 20 de la ejecución se superan los recursos para ajustes, se alargarán las ejecuciones de las diferentes máquinas en las que se estén utilizando recursos de ajuste en el tiempo 20 hasta que dejen de superarse. Si en un ejemplo existen 4 máquinas y hay 3 en las que se utilizan recursos de configuración se alargará cada una de las 3 por separado hasta que no se superen y se calculará el *makespan* una vez realizado el retraso. Se escogerá la opción que incremente menos el *makespan* final. Este proceso se repetirá hasta que se llegue al final de la ejecución donde terminará la heurística con una solución válida que respete el límite de los recursos.

El Pseudocódigo 2 muestra la metodología utilizada para obtener la posición ($null$ en caso contrario) en la que se ha superado el límite de los recursos. Las variables r_l^S y r_l^P servirán para guardar los recursos de ajuste y de procesamiento, respectivamente, que se estén utilizando en cada instante de tiempo de la ejecución. La variable l representará el índice para recorrer toda la matriz que represente una ejecución y contenga las asignaciones de los trabajos a cada máquina, un ejemplo de esta matriz podría ser el mostrado en la Tabla 16:

l	0	1	2	3	4	5	6	7	$l_{máx} = 8$
M1	S3	S3	P3	S2	P2	P2	P2	null	null
M2	S4	P4	P4	P4	P4	S1	S1	P1	P1

Tabla 16: Representación visual de una ejecución con las asignaciones de los trabajos

$l = 0$
$while\ l\ <=\ l_{máx}\ do$
    $r_l^S = 0;\ r_l^P = 0;$
    $for\ i \in M\ do$
        $r_l^S\ +=\ r_{ijk}^S$
        $r_l^P\ +=\ r_{ij}^P$
    end
    $if(r_l^S\ > r_{máx}^s\ or\ r_l^P >\ r_{máx}^p)\ do$
        $return\ l$
    end
    $l = l + 1$
end

Pseudocódigo 2: Fase de reparación: búsqueda de un punto donde se superen recursos

En el Pseudocódigo 3 se realiza el retraso del trabajo si l es diferente de $null$. El valor de l es el obtenido por el anterior Pseudocódigo 2. Si es diferente de $null$ se realizarán los retrasos en cada máquina donde se utilice este recurso que está siendo superado, y se quedará con el retraso que suponga un menor *makespan* final. La variable C_{min} representará el tiempo de finalización (*makespan*) mínimo con el retraso realizado en todas las máquinas donde se supera el recurso. La variable i_{min} tendrá almacenada la máquina donde el retraso supone incrementar menos el makespan, y será donde se realice el retraso.

$while(l\ !=\ null)\ do$
    $i_{min} = 0$
    $C_{min} = \infty$
    $for\ i \in M\ do$
        $C_i =\ calcularMakespanConSolape(i)$
        $if(C_i < C_{min})\ do$
            $C_{min} = C_i$
            $i_{min} = i$
        end
    end
    $atrasar(l, i)$
end

Pseudocódigo 3: Fase de reparación: retraso de trabajos que superen los recursos

Siguiendo con el ejemplo de la primera fase ahora se expondrán los datos necesarios relacionados con los recursos. Primero se establecerán los recursos máximos para los trabajos de ajuste y de procesamiento. A continuación, se mostrará la Tabla 17 que tendrá los recursos necesarios para realizar las tareas de procesamiento de todos los trabajos en cada máquina. Por último, las Tablas 18, 19 y 20 corresponden a los

recursos necesarios para los ajustes de todos los trabajos en las diferentes máquinas M1, M2 y M3.

$$R^{s}_{máx} = 9$$
$$R^{p}_{máx} = 9$$

	M1	M2	M3
J1	3	2	5
J2	3	5	4
J3	4	3	3
J4	1	5	1
J5	3	4	1
J6	1	5	5
J7	1	5	3

Tabla 17: Recursos de proceso necesarios para cada trabajo en la máquina 1 para la explicación del algoritmo

M1	J1	J2	J3	J4	J5	J6	J7
J1	5	1	5	1	3	2	3
J2	1	3	1	3	4	3	5
J3	3	4	3	3	3	2	5
J4	1	5	5	2	2	4	3
J5	1	2	2	4	4	5	5
J6	3	5	2	3	4	4	4
J7	3	4	4	2	3	5	2

Tabla 18: Recursos de ajuste necesarios para cada trabajo en la máquina 1 para la explicación del algoritmo

M2	J1	J2	J3	J4	J5	J6	J7
J1	4	4	2	3	2	5	5
J2	1	5	3	2	2	4	4
J3	5	5	4	2	1	3	3
J4	5	3	5	2	5	3	3
J5	1	4	2	1	3	2	5
J6	5	3	4	2	3	4	3
J7	3	4	4	2	3	5	2

Tabla 19: Recursos de ajuste necesarios para cada trabajo en la máquina 2 para la explicación del algoritmo

M3	J1	J2	J3	J4	J5	J6	J7
J1	3	5	4	3	2	2	5
J2	2	1	2	4	1	5	3
J3	4	2	4	3	4	1	4
J4	4	2	5	5	1	5	4
J5	5	1	1	2	1	2	5
J6	2	4	3	4	4	4	4
J7	4	1	2	5	5	2	2

Tabla 20: Recursos de ajuste necesarios para cada trabajo en la máquina 3 para la explicación del algoritmo

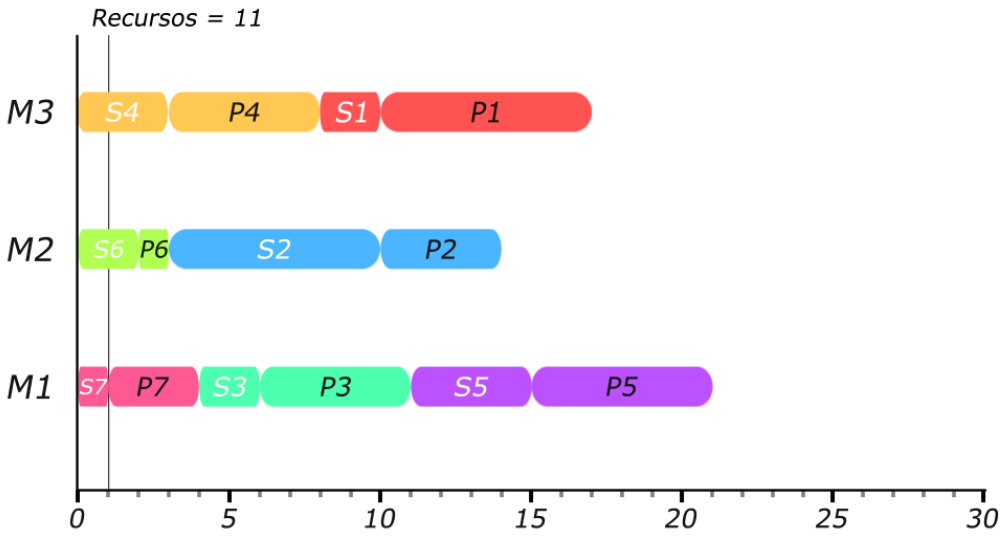

Figura 15: Superación de recursos de ajuste en el punto 0

Si se observa la Figura 15, en el punto 0 de la ejecución se están superando los recursos permitidos para los tiempos de ajuste. La máquina M1 necesita 2 recursos para el trabajo 7, la máquina M2 necesita 4 recursos para el trabajo J6 y la máquina M3 5 para el trabajo J4, sumando un total de 11 que supera el máximo establecido en 9. Por lo que se precisará de retrasar alguno de los tres trabajos.

El inicio de la ejecución siempre puede suponer un inconveniente para este problema, porque será el único punto de la ejecución donde se tendrá la certeza de que todos los trabajos estarán en su fase de ajuste siendo más probable que haya que realizar retrasos como ha sido en el caso de este ejemplo.

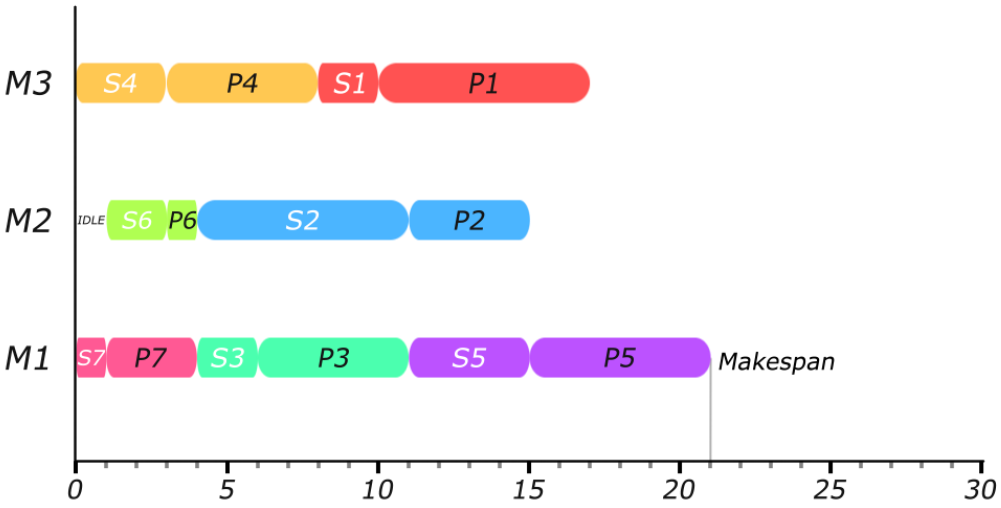

Figura 16: Retraso de máquina 2 en el punto 0

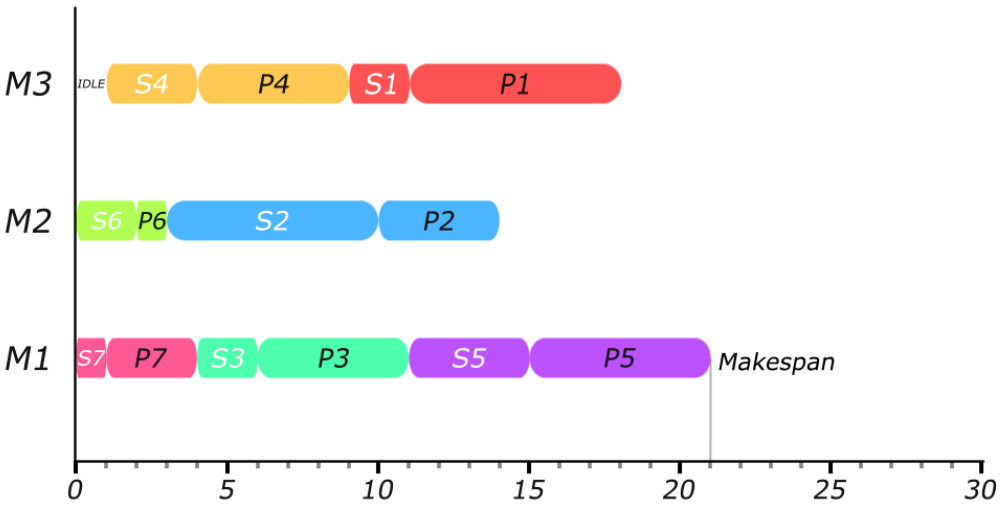

Figura 17: Retraso de máquina 3 en el punto 0

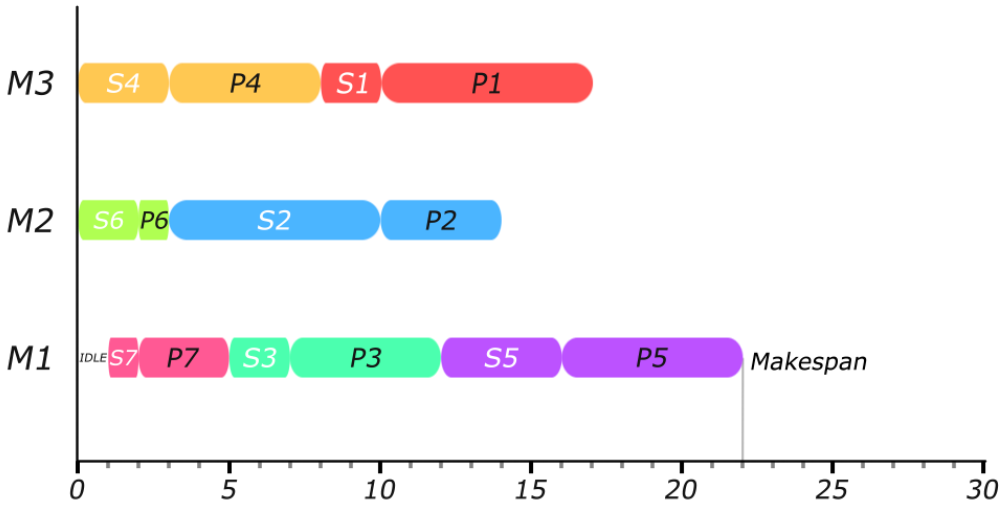

Figura 18: Retraso de máquina 1 en el punto 0

En las Figuras 16 y 17 se observa que con ambos retrasos el *makespan* no se ve afectado, en ambas sigue siendo 21. Cualquiera de estas dos opciones sería válida y se elegiría una al azar, porque con ambas el *makespan* aumentaría lo mínimo posible, en este caso 0. Con la opción restante, la asignación mostrada en la Figura 18, se observa que en este caso el *makespan* aumenta a 22 y deja de ser la mejor opción.

El algoritmo decide retrasar la ejecución de la máquina 2 porque ha sido la primera comprobación que ha rebajado el *makespan*, la máquina 3 ha sido la siguiente y como no lo ha disminuido la ha descartado. Esto sucede porque el *makespan* resultante es el mismo, y el algoritmo para decidir entre una solución u otra solo se basa en el *makespan* que se produce al realizar un retraso.

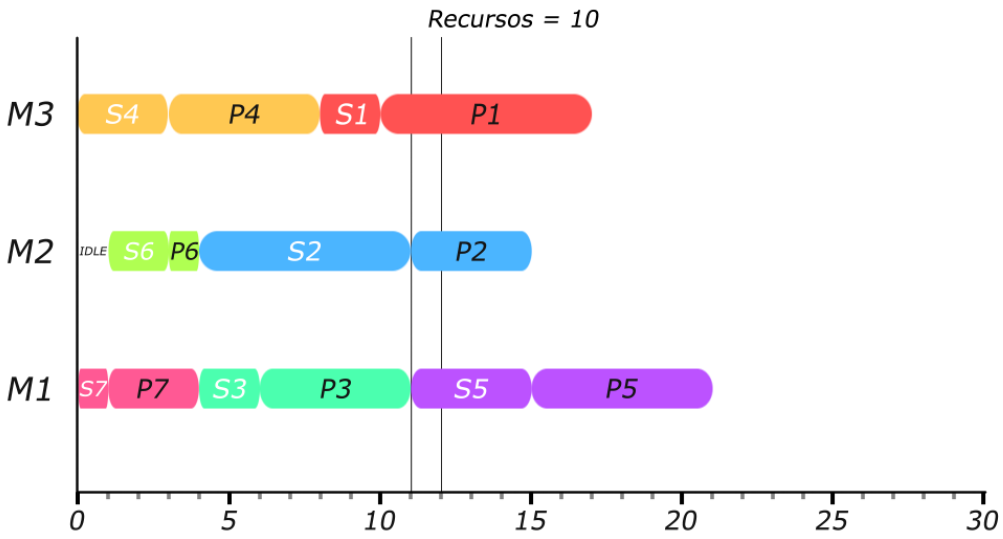

Figura 19: Superación de recursos de procesamiento en el punto 11

Una vez se ha realizado el primer solape no se vuelven a superar los recursos hasta que se llega al punto 10 como muestra la Figura 19. A pesar de que en este momento solo existe un trabajo en la fase de ajuste y dos en la de procesamiento (no todos forman parte de la misma fase), también se puede superar el número de recursos de procesamiento máximo como ocurre aquí. En este punto la máquina M3 necesita 5 recursos para el procesamiento del trabajo J1 igual que el trabajo J2 en la máquina M2 que necesita de otros 5, sumando 10 y superando por 1 el límite establecido en 9.

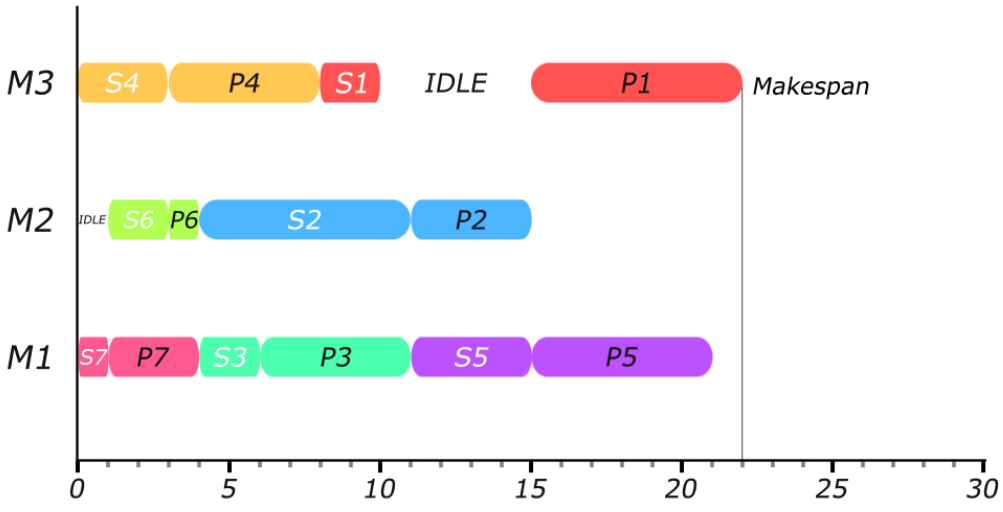

Figura 20: Retraso de la máquina 3 en el punto 10

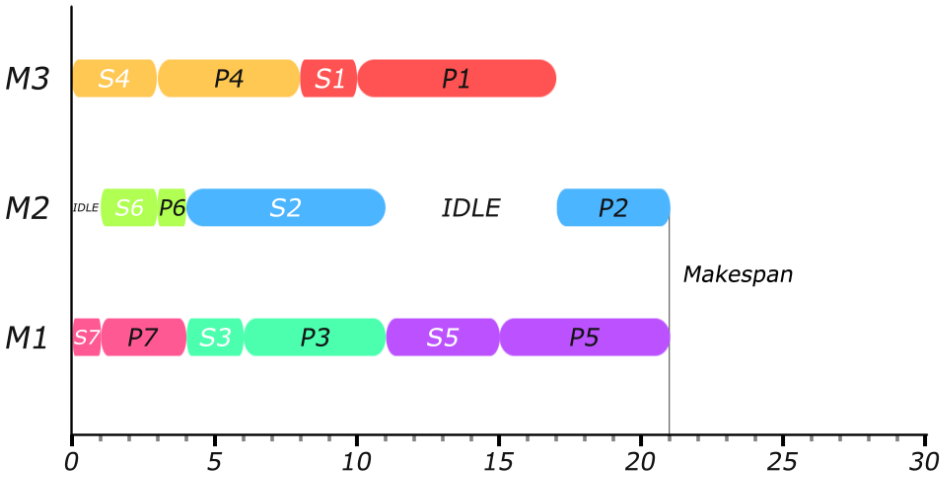

Figura 21: Retraso de la máquina 2 en el punto 11

Al ser dos trabajos los que provocan la superación de recursos, habrá dos retrasos posibles. En la Figura 20 se puede observar el primer atraso que se producirá en la máquina M3. En esta se deberá atrasar el trabajo J1 hasta que termine el trabajo J2 en la máquina 2, esto es, en el punto 15. Una vez ya ha sido recolocado se observa que el *makespan* ha aumentado en una unidad respecto al original, 21 frente a 22 con la nueva asignación.

En la Figura 21, se realiza el otro retraso posible, el de J2 en la máquina 2. Este trabajo deberá ser atrasado también hasta que finalice J1 en la máquina 3, esto sucederá en el punto 17. Habiendo sido colocado el trabajo en su nueva posición, el *makespan* no aumenta ya que termina justo en el mismo instante en el que lo hacía el trabajo J5 que era el que marcaba el *makespan* actual. Por tanto, esta última asignación será la idónea y la definitiva.

Con este último movimiento el proceso de reparación habrá terminado, así como el algoritmo, porque no se vuelven a superar los recursos. Se ha obtenido una solución válida, la mostrada en la Figura 21, con un *makespan* de 21.

7. Aleatorización

Cómo se verá en la sección de experimentos, el *makespan* obtenido con la heurística anterior dista en promedio un 7.52% con respecto al obtenido con el modelo de programación lineal entera, en un conjunto de instancias aleatorias utilizadas para validar los algoritmos desarrollados en este trabajo. Para intentar obtener soluciones aún más cercanas al óptimo, se procederá a una aleatorización del algoritmo anterior. Más específicamente, en la fase constructiva se elegirá aleatoriamente una asignación de entre una lista con los mejores candidatos. El tamaño de esta lista es un parámetro del algoritmo y ha de ser calibrado. Típicamente tal tamaño no debe ser muy grande porque de ser así la solución sería prácticamente aleatoria, y se busca una "aleatoriedad" entre las mejores posibilidades, no entre todas.

Para ello en la fase constructiva después de que se hayan probado todos los trabajos en todas las máquinas se obtendrán las mejores asignaciones (conjunto llamado *RCL* por sus siglas en inglés, *Restricted Candidate List*), se almacenarán en una lista, y se elegirá uno de ellos aleatoriamente. Se eliminará dicho trabajo de la lista de trabajos por asignar y se pasará a realizar la siguiente iteración en la que se repetirá el mismo proceso aleatorio, o por el contrario se pasará a la fase siguiente del algoritmo si ya no quedan más trabajos por asignar, como en la primera fase original.

$$tiempo_{máx} = \frac{núm\ trabajos \times núm\ máquinas \times t}{2}, t \in U\{1, \ldots, 5\}$$

$r = 2;\ RCL = 4$
$for\ r \in RCL\ do$
    $t = 0$
    $while\ t < tiempo_{máx}\ do$
        $\underline{Fase\ de\ elección\ de\ mejores\ candidatos}$
        $index = random(1, r)$
        $j = RCL_{list}.get(index)$
        $nTAsig.delete(j)$
        $mAsig.add(i, j)$

        $t \mathrel{+}= tiempo\ utilizado\ en\ esta\ iteración$
    end
end

Pseudocódigo 4: Aleatorización del algoritmo

Con el Pseudocódigo 4 el algoritmo se ejecutaría un número determinado de veces hasta que el tiempo acotado para dicha instancia se termine. Este tiempo, $t_{máx}$, se calculará según la fórmula anterior al Pseudocódigo 4. En cada iteración del bucle interno se realizará el algoritmo completo con la elección aleatoria de uno de los mejores candidatos en cada iteración de la fase constructiva, y se guardará el resultado del *makespan* obtenido cuando termine la ejecución del algoritmo para un valor de RCL concreto. Este resultado se comparará en cada iteración general con el mejor resultado obtenido hasta este momento, para guardarlo o descartarlo según lo mejore o no. Esta será una versión simple. En el siguiente apartado se realizarán diferentes experimentos modificando diversas partes de la heurística. Más concretamente, se evaluarán diferentes tamaños de RCL, se probarán distintos tiempos máximos de cómputo (variación del valor t en la fórmula que calcula el tiempo máximo $t_{máx}$), se analizará si conviene dar la misma probabilidad a todos los elementos del RCL o no, y se verá la conveniencia o no de tener en cuenta la cantidad de recursos utilizados por cada trabajo a la hora de realizar una asignación durante la fase constructiva.

8. Experimentos computacionales

En esta sección se efectuarán pruebas en las que se modificarán distintos factores que forman parte de los algoritmos propuestos. El objetivo es intentar minimizar el *makespan* lo máximo posible utilizando el algoritmo base ya diseñado.

8.1 Generación de instancias

A lo largo del trabajo, tanto la comparación de las heurísticas para escoger una como para realizar los experimentos se ha realizado siempre sobre la misma base de instancias, las cuales variaban en número de trabajos (5 valores discretos: 10, 20, 30, 40 o 50 trabajos) y número de máquinas (3 valores: 4, 6 u 8 máquinas). Cada instancia estaba formada por todas las combinaciones posibles entre los conjuntos de valores de las máquinas y los trabajos formando un total de 15 instancias. Además, cada combinación estaba repetida 5 veces (con diferentes valores para los tiempos de ajuste y procesamiento así como para los recursos en ambas etapas), obteniendo un total de 75 instancias. Estas instancias son las mismas que se utilizaron en el artículo de Fanjul-Peyro 2020, el último artículo repasado en el capítulo dedicado al estado del arte, donde se resuelven con el modelo de programación matemática anteriormente descrito y con el que se comparan los resultados. Se explica a continuación cómo se generaron dichas instancias.

Los tiempos de procesos y de ajuste de cada trabajo en cada máquina se generan aleatoriamente, a partir de una distribución uniforme discreta en el rango {50, …, 100}. En cada máquina, antes de procesar su primer trabajo, es necesario un tiempo de ajuste y por tanto de necesidad de recursos. El número de recursos necesarios también se genera aleatoriamente, ya sean para el procesamiento o para los ajustes. Sus valores se obtienen a partir de una distribución uniforme discreta en el rango {1, …, 9}. Para establecer el límite superior de recursos $R^s_{máx}$ y $R^p_{máx}$ éste era calculado con la fórmula:

$$R^s_{máx} = media\ del\ conjunto\ de\ valores\ posibles\ de\ recursos \times núm\ máquinas$$

$$R^p_{máx} = media\ del\ conjunto\ de\ valores\ posibles\ de\ recursos \times núm\ máquinas$$

El esquema de datos resumidos sería:

$$-n \in \{10, 20, 30, 40, 50\}$$
$$-m \in \{4, 6, 8\}$$
$$-p_{ij} \in U\{50, \dots, 100\}$$
$$-s_{ijk} \in U\{50, \dots, 100\}$$
$$-r^p_{ij} \in U\{1, \dots, 9\}$$

$$-r_{ijk}^{s} \in U\{1, \dots, 9\}$$

Antes de pasar a realizar los experimentos se establecerá el parámetro de calidad utilizado en este trabajo para medir si una heurística es buena o no. Esto se realiza según la media del GAP, definido como la diferencia porcentual entre el *makespan* obtenido con el algoritmo propuesto en este trabajo y el *makespan* obtenido con el modelo matemático, expresado como:

$$GAP = \frac{100 \times (makespan\ heurística - makespan\ modelo)}{makespan\ modelo},$$

donde *makespan heurística* es el *makespan* obtenido por el algoritmo, y *makespan modelo* es el *makespan* obtenido por el modelo de programación matemática para una instancia concreta. Un valor positivo de GAP implica que el *makespan* obtenido con la heurística es mayor (peor) que el *makespan* obtenido con el modelo. Análogamente, si el GAP es negativo esto indica que el *makespan* obtenido por la heurística es menor (mejor) que el *makespan* obtenido por el modelo. Si el *makespan modelo* es igual al *makespan heurística* el valor del GAP es 0.

8.2 Resultados del algoritmo sin aleatorizar

Para dar con la heurística final que es la elegida para los experimentos posteriores se han realizado diferentes versiones con pequeñas modificaciones entre ellas, (se explicarán a continuación) que se han ido probando y midiendo sus gaps, y que comparten en mayor o menor medida el esquema de dos fases que se ha definido anteriormente.

1. La primera versión es una adaptación de la propuesta en Yepes 2017.
2. En la segunda heurística se elegiría el mejor trabajo en cada iteración (el que genere menor *makespan*) en la fase constructiva, y en la siguiente fase se retrasarían los trabajos que sobrepasen el límite de recursos hasta que finalizan.
3. La tercera heurística, a diferencia de la segunda, realizaría el solape hasta que se dejaran de superar los recursos máximos y no hasta que terminaran los trabajos que están utilizando dichos recursos, como en la segunda.
4. La cuarta heurística descartaría en la primera fase los trabajos que superen un umbral de 80 de *makespan* y 7 de uso de recursos (estos valores eran elegidos en una fase previa en la que se probaban diferentes valores y se comprobaba el gap resultante). La elección de esta fase podría perjudicar el proceso de aleatorización posterior porque habría menos trabajos para escoger y era una versión más compleja que la primera heurística.

Previo al anterior análisis con las 4 versiones de las heurísticas, dos que no forman parte de esas cuatro fueron descartadas y no introducidas en la elección final por

razones de gap y rendimiento. La que podría haber sido la quinta versión se diferenciaba en la primera fase, en la que para decidir un trabajo u otro se tenía en cuenta el *makespan* generado igual que antes, pero si se superaban los recursos, este *makespan* era el generado retrasando los trabajos para respetar el límite de recursos. Empeoraba bastante el GAP.

La última versión, la sexta, realizaba dos asignaciones en vez de una en cada iteración en la primera fase, para asegurarse de que después de un trabajo siempre hubiera otro con el que "combinara" bien para minimizar el *makespan* lo máximo posible. El *makespan* fue un poco superior y el tiempo de cómputo mucho mayor porque tenía que realizar muchas más combinaciones en la etapa de asignación de la fase constructiva.

Los resultados promedio sobre las 75 instancias obtenidos de las cuatro versiones de la heurística, están resumidos en la Tabla 19, donde se especifica en la primera fila las diferentes versiones con sus valores respectivos de GAP en la fila inferior:

Versión	1	2	3	4
GAP	15.64	10.02	7.52	7.46

Tabla 19: Tabla resumen del GAP obtenido para cada versión

En el siguiente gráfico *boxplot* se muestra de forma general la distribución del GAP de cada instancia probada en las cuatro versiones diferentes de las heurísticas:

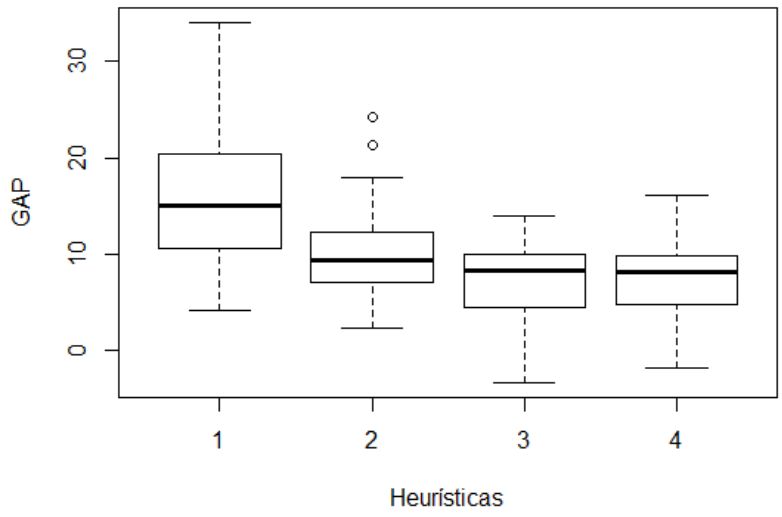

Figura 22: Comparación de heurísticas implementadas utilizando el GAP

En el eje horizontal se pueden apreciar las 4 versiones de la heurística y en el eje vertical el valor de los gaps con sus diferentes cuartiles que ha ido tomando cada instancia para esa heurística.

Viendo los resultados se ha terminado por seleccionar la tercera versión, ya que tiene un GAP (7.52) muy similar respecto al mejor obtenido en general (7.46) pero es una heurística más simple que tiene en cuenta todos los trabajos en su fase constructiva independiente de sus tiempos y recursos para el ajuste y procesamiento.

8.3 Resultados del algoritmo aleatorizado

A partir de esta versión "inicial" se han realizado diferentes experimentos, probando las versiones aleatorizadas de los algoritmos propuestos que variarán según los siguientes factores:

1. El primer factor sería el tamaño de RCL, la lista con los mejores candidatos que se ha explicado en la fase de la aleatorización, y que estará formada por los trabajos que generen un menor *makespan* en cada iteración de la fase constructiva. Los tres niveles experimentados para este factor serán 2, 3 y 4.

$$RCL \in \{2,3,4\}$$

2. El segundo factor es el tiempo de ejecución que se le permite a la metaheurística. El tiempo que estará en ejecución será calculado según la siguiente fórmula, la misma que la del Pseudocódigo 4 vista anteriormente:

$$tiempo = \frac{núm\ trabajos \times núm\ máquinas \times t}{2}, t \in U\{1, \dots, 5\}$$

   El valor obtenido serán los segundos que la heurística estará en funcionamiento para dicha instancia. El intervalo de tiempo en el que el algoritmo estará en ejecución estará comprendido entre 20 segundos (mínimo con 10 trabajos y 4 máquinas con $t = 1$) y 1000 segundos, casi 17 minutos (máximo con 50 trabajos, 8 máquinas y $t = 5$). Por tanto, este factor tiene cinco niveles diferentes.

   En comparación con los tiempos del modelo, para las instancias grandes, si no encuentra la solución óptima la estará buscando hasta llegadas a las 3 horas, que es lo que ocurre con las instancias con un número de trabajos elevado.

3. El tercer factor es la distribución de probabilidad para escoger los trabajos en la fase constructiva: una versión en la que cada uno de los trabajos de la lista de mejores candidatos tengan la misma probabilidad de ser elegidos (uniforme) y otra en la que los trabajos que tengan mejor *makespan* tengan mayor probabilidad (con pesos). Al ser dos las opciones posibles para este factor, tendrá dos niveles.

   La versión con pesos se realizará asignando pesos a cada uno de los trabajos candidatos según su *makespan* siguiendo la siguiente fórmula, donde la

variable x tendrá el valor del peso que deberá tener cada trabajo según su *makespan*:

$$RCLx + (RCL - 1)x + \ldots + x = 1$$

Por ejemplo, para RCL = 4:

$$4x + 3x + 2x + x = 1 \rightarrow x = \frac{1}{10}$$

Según el orden que ocupen los trabajos por el *makespan* que generarían al ser asignados tendrán un porcentaje de: 40%, 30%, 20% y 10% de ser escogidos, de mejor a peor. Cada trabajo tendrá asignado un rango comprendido entre 0 y 1, ya que se generará un número aleatorio dentro de este intervalo y según su valor se elegirá un trabajo u otro, teniendo más posibilidades el que mejor *makespan* tenga. Estos serían los rangos de probabilidades si el RCL fuera 4:

Trabajos	1er mejor trabajo	2do mejor trabajo	3er mejor trabajo	4to mejor trabajo
Rango	0 - 0.40	0.40 - 0.70	0.70 - 0.90	0.90 - 1

Tabla 20: Fraccionamiento de probabilidades para escoger un trabajo

4. El último factor cambiaría la elección de los trabajos de la heurística. En lugar de elegirlos solo en base al *makespan* que producirían, también se propone elegir los trabajos en la fase constructiva en base a una combinación convexa entre el *makespan* producido y el uso de recursos necesarios. Este valor (denominado $norm$ de normalización) es calculado siguiendo la siguiente fórmula, y variando el valor de una variable llamada alfa (α) en el intervalo $\{0, 0.1, \ldots, 1\}$, para darle mayor o menor peso al *makespan* conforme el valor es más grande. El motivo con el que se introduce este factor es para darle importancia al uso de recursos en la elección de un trabajo en la fase constructiva, ya que hasta ahora solo se realizaba la selección en base al *makespan*.

$$norm = (\alpha \times (makespan - \acute{u}ltimo\ makespan) + (1 - \alpha) \times recursos),$$

donde *makespan* es el *makespan* que generaría la asignación de dicho trabajo, último *makespan* es el *makespan* que existía previo a la supuesta asignación que se va a realizar, y recursos sería la suma de recursos de ajuste y procesamiento necesarios para esta asignación.

Para concluir cual puede ser el valor de α más interesante para reducir el gap se ha ejecutado la versión de la heurística sin aleatorizar sobre las 75 instancias y en cada ejecución se ha probado un valor de alfa distinto. Para cada valor de alfa en ese intervalo, se han obtenido los siguientes GAPs promedio:

Valor α	**GAP promedio**
0	50.81
0.1	17.84
0.2	12.29
0.3	9.31
0.4	7.49
0.5	7.68
0.6	7.24
0.7	6.97
0.8	7.26
0.9	7.42
1	7.35

Tabla 21: Valores de GAP obtenidos con alfa $\in \{0, 0.1, \dots, 1\}$.

Como se puede observar en la anterior tabla el valor de α con el que se obtiene el mejor gap es con 0.7, que ya mejora en cierto modo respecto a la versión en la que solo se tenía en cuenta el *makespan* para la elección (6.97 respecto a 7.52). Este valor de α será el utilizado para realizar todas las aleatorizaciones posteriores en las que se utilice la normalización para seleccionar un trabajo.

De una forma resumida, los cuatro factores probados y sus niveles serían los siguientes:

$$-RCL \in \{2,3,4\}$$

$$-Tiempo = \frac{(n \times m \times t)}{2}, t \in \{1, \dots, 5\}$$
$$-Probabilidad: \{Uniforme, Con\ pesos\}$$
$$-Recursos: \{NO, SÍ\}$$

El número de posibles combinaciones de los niveles de estos factores es $3x5x2x2 = 60$. Cada una de esas combinaciones se aplica a cada una de las 75 instancias que tenemos, teniendo que resolver por lo tanto $60x75 = 450$ ejecuciones.

8.4 Análisis de resultados

Antes de pasar a los resultados finales cabe destacar que el algoritmo ha sido desarrollado en la versión de Java 8, sobre una máquina con sistema operativo *Windows 10*, AMD Opteron Abu Dhabi 6344 a 2.5 GHz y 8 GB de memoria RAM. Es importante tenerlo en cuenta porque no es una máquina muy actualizada por lo que los resultados podrían variar favorablemente en caso de que así lo fuera.

Como se ha comentado anteriormente, se medirá la calidad de una solución en función de lo pequeño que sea su GAP y del tiempo de ejecución utilizado para obtenerla. Por ello, uno de los factores principales ha sido la variación del tiempo para observar la variabilidad del gap obtenido. A continuación, se enumerarán en un listado todos los resultados de los experimentos realizados. Cada experimento probará todos los tamaños de RCL y todos los posibles tiempos de cómputo, variando los otros dos factores obteniendo un total de 4 experimentos. En la Tabla 22 se resumen los experimentos a realizar.

	RCL	Tiempo	Probabilidad	Recursos
1er experimento	$RCL \in \{2,3,4\}$	$t \in \{1, \dots, 5\}$	Uniforme	No
2ndo experimento	$RCL \in \{2,3,4\}$	$t \in \{1, \dots, 5\}$	Con pesos	No
3er experimento	$RCL \in \{2,3,4\}$	$t \in \{1, \dots, 5\}$	Uniforme	Sí
4to experimento	$RCL \in \{2,3,4\}$	$t \in \{1, \dots, 5\}$	Con pesos	Sí

Tabla 22: Resumen con los experimentos a realizar

Por tanto, la primera prueba es con una probabilidad de elección del trabajo con RCL uniforme y sin considerar los recursos en la fase constructiva, solo se tiene en cuenta el *makespan* generado en cada iteración. El resumen de resultados está en la Tabla 23, donde cada fila estará formada por una combinación de los factores de tiempo y RCL y cada columna tendrá el GAP obtenido para las instancias con un tamaño de 10 a 50 trabajos y por último el GAP de media total.

		GAP					
		n					
TIEMPO	RCL	10	20	30	40	50	**TOTAL**
(n*m*1)/2	2	3,76	6,10	4,94	4,33	1,86	**4,20**
	3	3,99	7,24	6,68	5,71	3,20	**5,37**
	4	3,19	8,56	7,80	7,84	4,44	**6,36**
(n*m*2)/2	2	3,75	5,96	4,84	4,49	1,41	**4,09**
	3	3,06	6,67	5,99	4,85	2,47	**4,61**
	4	2,89	7,08	7,57	6,76	3,68	**5,60**
(n*m*3)/2	2	3,53	5,03	4,09	3,47	1,08	**3,44**
	3	2,41	5,95	5,29	5,59	2,13	**4,27**
	4	1,96	6,95	6,87	5,92	3,50	**5,04**
(n*m*4)/2	2	3,45	5,22	4,27	3,33	1,12	**3,50**
	3	2,61	5,57	4,96	4,98	1,67	**3,96**
	4	**1,81**	6,75	6,80	5,66	2,94	**4,79**
(n*m*5)/2	2	3,45	4,34	4,20	3,69	0,65	**3,27**
	3	1,98	5,61	5,39	4,26	1,85	**3,82**
	4	1,83	5,65	6,41	5,79	2,97	**4,53**

Tabla 23: GAPs obtenidos para la versión aleatorizada sin pesos

En esta primera ejecución, ya se puede ver como la aleatorización ha reducido el gap a más de la mitad de su valor inicial (7.52 frente a los 3.27, obtenidos con el mayor tiempo permitido y RCL = 2). También se aprecia cómo $RCL = 2$, el RCL en el que se elige un candidato de entre los dos mejores destaca frente al resto, y a pesar de que el tiempo aumente ningún otro lo mejora. Asimismo, cabe destacar que desde t=3 la evolución del gap para $RCL = 2$ ha sido pequeña, esto puede ser un indicativo de que aumentar el tiempo ya no suponga una mejora notable del GAP.

Por último, para las instancias con 10 trabajos, el mejor RCL no es el 2, en el cual sí que destaca para el resto, si no que el mejor gap se obtiene con un $RCL = 4$. Para

problemas con 10 trabajos debería interesar este RCL porque disminuye el GAP frente a $RCL = 2$. Esto es debido en gran medida a que en estas instancias tan pequeñas el algoritmo realiza muchas más iteraciones que en las grandes, y el hecho de que el RCL sea mayor y se realizan tantas iteraciones provoca que los GAPs sean mejores, dado que la diversidad de soluciones encontradas es mayor. Obviamente si el RCL fuera igual al número de trabajos y se dejara el algoritmo ejecutando durante un tiempo indefinido, en algún momento encontraría la solución óptima.

En la siguiente Figura 23, se observará la evolución del GAP. En esta gráfica cada línea representará la evolución del GAP conforme el tiempo de ejecución aumente (eje horizontal).

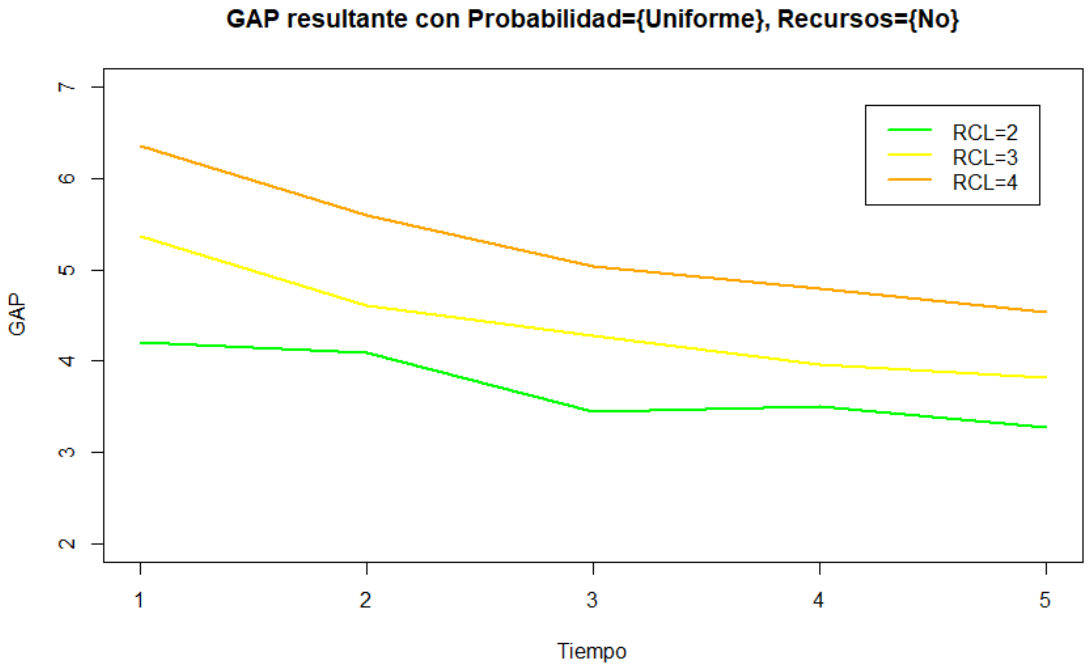

Figura 23: GAP resultante con probabilidad uniforme y sin tener en cuenta recursos

A pesar de que con $RCL = 2$ es con el que mejor valor de GAP se obtiene, se observa en la Figura anterior que con este es con el que menos evoluciona el GAP según se aumente el tiempo disponible. Esto confirma lo que se ha comentado anteriormente, al ser un RCL que contiene solo a los dos mejores candidatos la variabilidad de las soluciones es menor que con $RCL = 3$ y $RCL = 4$, que a pesar de notar una mayor evolución respecto a $RCL = 2$, se observa una tendencia en la forma de sus curvas a estabilizarse conforme el tiempo aumente.

En la siguiente tabla se variará la probabilidad de elegir el trabajo cuando se realiza la asignación en función de la normalización calculada, que tendrá en cuenta el *makespan* y los recursos como se ha explicado en el apartado anterior. Con este nuevo factor es con el que se realiza la segunda ejecución mostrada en la Tabla 24, acaparando también todas las combinaciones de t y RCL.

		GAP					
		TRABAJOS					
TIEMPO	RCL	10	20	30	40	50	**TOTAL**
(n*m*1)/2	2	4,00	5,80	4,99	3,87	0,99	**3,93**
	3	3,69	6,35	4,88	4,64	2,09	**4,33**
	4	2,19	6,55	7,17	6,32	3,56	**5,16**
(n*m*2)/2	2	3,91	4,91	4,33	3,55	0,90	**3,52**
	3	3,25	5,72	5,00	4,29	1,49	**3,95**
	4	2,32	6,15	6,35	5,28	2,57	**4,53**
(n*m*3)/2	2	3,60	4,81	3,93	3,03	0,40	**3,15**
	3	2,86	5,26	4,77	4,11	1,52	**3,70**
	4	1,67	5,77	5,90	4,46	1,98	**3,96**
(n*m*4)/2	2	3,64	4,68	3,32	2,92	0,55	**3,02**
	3	2,93	4,86	4,01	3,65	1,25	**3,34**
	4	1,62	5,43	5,56	4,23	2,41	**3,85**
(n*m*5)/2	2	3,64	4,32	3,41	3,03	0,37	**2,95**
	3	2,54	5,02	4,16	3,42	0,48	**3,13**
	4	1,85	5,06	4,64	4,28	1,66	**3,50**

Tabla 24: GAPs obtenidos para la versión aleatorizada con pesos

Con esta nueva ejecución, la primera conclusión que se puede obtener es que ha mejorado el gap respecto a la anterior ejecución en algo más de 3 décimas (3.27 -> 2.95 para $RCL = 2$ y tiempo máximo de cómputo). También mejora el gap en el resto de los tamaños de RCL y t bastante respecto a la anterior, lo que puede significar que la introducción de pesos podría ser interesante y mejoraría generalmente la metaheurística. También es visible como en esta segunda ejecución, la diferencia entre los gaps de un mismo t para distintos $RCLs$ es mucho menor que la anterior. Esto seguramente sea debido a la introducción de los pesos que provoca que, aunque sea el mismo RCL, este está condicionado porque se le da mayor prioridad a los mejores trabajos.

Por otra parte, es curioso observar como para las instancias de 10 trabajos, con RCL=4, que es el mejor para las instancias de este tamaño, de t=4 a t=5 el gap empeora (1.62 -> 1.85). Esto podría ser indicativo de que la heurística ya ha llegado al máximo dando a entender que, aunque se aumente el tiempo para encontrar una solución, es poco probable que mejore.

Lo último a comentar en esta ejecución sería la proximidad al valor 0 del GAP para las instancias con 50 trabajos, viendo que para RCL=2 (0.37), el rendimiento de la heurística sería prácticamente el mismo al del modelo, aun teniendo el modelo hasta 3 horas para buscar la mejor solución frente a los casi 17 minutos que habría estado la heurística para una única instancia. En este caso la heurística puede mejorar la solución para muchas instancias frente a las obtenidas por el modelo, en tiempos de cómputo mucho más reducidos.

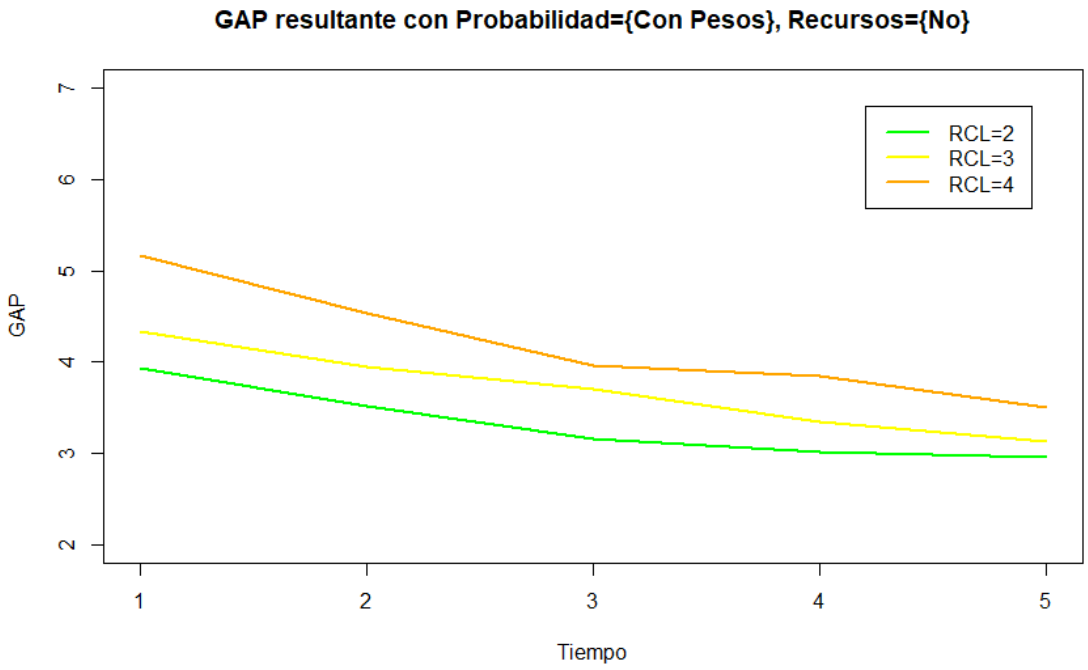

Figura 24: GAP resultante con Probabilidad con pesos y sin tener en cuenta recursos

En la Figura 24 se ha realizado la misma gráfica de comparación que en la anterior Figura 23. En este caso la primera diferencia observable es que la diferencia entre el GAP obtenido por los diferentes $RCLs$ es mucho menor que con una versión en la que la probabilidad sea equitativa entre los mejores candidatos. En este caso el punto donde las tres líneas puedan converger parece estar cerca, indicio de que si el algoritmo se quisiera dejar algo más de tiempo puede que RCL=3 o RCL=4 fueran las mejores opciones. Aparte de que RCL=2 a partir de t=3 presenta una evolución muy pequeña.

Para la tercera y cuarta ejecución se tendrán en cuenta los recursos en la fase constructiva. Existirá una versión con distribución de probabilidad uniforme (la tercera), y una con pesos (la cuarta) al igual que con las anteriores dos ejecuciones.

En esta versión se utilizará la fórmula de normalización explicada anteriormente, con el valor de $\alpha = 0.7$ que es con el que se ha obtenido el mejor GAP de media para la heurística sin aleatorizar. Se realizará la aleatorización siguiendo el mismo proceso, solo que en vez de tener en cuenta el *makespan* para la elección de los mejores trabajos, esta elección se realizará según el resultado de la fórmula de normalización. La Tabla 25 muestra los resultados obtenidos:

		GAP					
		TRABAJOS					**TOTAL**
TIEMPO	RCL	10	20	30	40	50	
(n*m*1)/2	2	3,30	5,97	5,31	4,82	2,53	**4,39**
	3	3,51	7,26	6,89	6,18	3,48	**5,46**
	4	4,01	7,81	7,35	7,32	4,44	**6,19**
(n*m*2)/2	2	3,00	5,50	4,82	3,44	1,27	**3,60**
	3	2,92	6,48	5,65	5,31	2,58	**4,59**
	4	2,45	7,47	7,10	5,73	4,34	**5,42**
(n*m*3)/2	2	3,13	4,67	4,38	3,49	1,15	**3,36**
	3	2,73	6,51	5,50	4,17	1,95	**4,17**
	4	2,73	6,82	6,87	5,92	3,44	**5,16**
(n*m*4)/2	2	3,17	4,63	3,61	3,61	0,76	**3,16**
	3	2,56	5,62	5,13	4,67	2,11	**4,02**
	4	2,17	7,31	6,39	5,03	3,07	**4,80**
(n*m*5)/2	2	3,00	4,56	3,96	3,41	0,73	**3,13**
	3	2,21	5,24	4,81	4,05	1,73	**3,61**
	4	1,81	6,07	6,28	5,92	2,80	**4,58**

Tabla 25: GAPs obtenidos para la versión aleatorizada sin pesos y teniendo en cuenta los recursos en la fase constructiva

En esta penúltima tabla, se vuelven a obtener unos resultados parecidos a la primera, con mucha diferencia entre GAPs y con peores resultados que con la segunda tabla, que es la mejor hasta el momento. Aunque los resultados son bastante buenos pese a ser una versión sin pesos (que hasta el momento parece ser la peor opción no considerar los pesos para la asignación de trabajos).

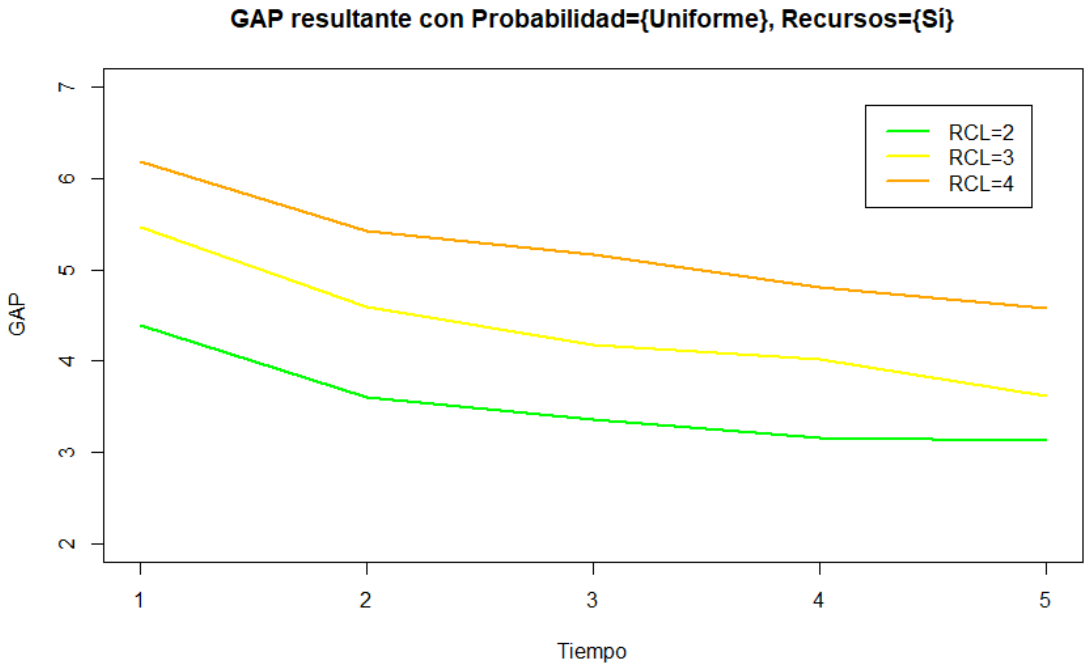

Figura 25: GAP resultante con probabilidad uniforme y teniendo en cuenta recursos

Así como si se cambia la probabilidad entre uniforme y con pesos, se puede observar como aparte de que el GAP disminuye un poco, los distintos $RCLs$ tienen menos diferencia entre ellos, no ocurre lo mismo para los recursos como se puede observar en la Figura 25. La diferencia de GAP es notoria entre los diferentes $RCLs$ y no parece disminuir conforme el tiempo aumenta. A pesar de que el mejor GAP ha disminuido respecto a la misma versión en la que no se tienen en cuenta los recursos de la fase constructiva. Esto puede ser indicativo de que la versión en la que se considera una probabilidad con pesos y se tienen en cuenta los recursos en la fase constructiva pueda ser la que mejores resultados de GAP se obtenga.

Por último, en la Tabla 26 se ejecutará la versión con pesos y con elección del mejor trabajo según el valor de la normalización que tiene en cuenta los recursos explicada antes:

		GAP					
		TRABAJOS					
TIEMPO	RCL	10	20	30	40	50	**TOTAL**
(n*m*1)/2	2	3,37	5,66	4,81	4,05	1,31	**3,84**
	3	3,30	6,32	5,78	5,48	2,12	**4,60**
	4	2,62	7,02	6,61	6,16	3,39	**5,16**
(n*m*2)/2	2	3,37	4,54	4,09	3,19	0,82	**3,20**
	3	3,06	5,24	4,70	3,57	2,01	**3,72**
	4	2,58	6,29	6,07	5,03	2,42	**4,48**
(n*m*3)/2	2	3,25	5,09	3,79	3,25	0,55	**3,18**
	3	2,56	4,84	4,46	4,09	1,43	**3,48**
	4	1,76	5,70	5,48	4,58	2,06	**3,92**
(n*m*4)/2	2	3,11	4,26	3,53	2,92	0,80	**2,93**
	3	2,27	4,72	4,23	3,44	1,00	**3,13**
	4	1,75	5,31	5,18	4,14	1,77	**3,63**
(n*m*5)/2	2	3,28	4,56	3,77	2,84	0,51	**2,99**
	3	2,53	4,78	4,10	3,32	1,11	**3,17**
	4	1,27	5,10	4,84	4,20	1,94	**3,47**

Tabla 26: GAPs obtenidos para la versión aleatorizada con pesos y teniendo en cuenta los recursos en la fase constructiva

Es en esta última versión donde se encuentra el mejor GAP promedio y por tanto la mejor solución. La gran ventaja es que ésta ha sido hallada en $t = 4$, sin tener que utilizar el máximo tiempo disponible que hubiera tenido con $t = 5$, donde ha empeorado unas centésimas (esto último es debido a la aleatoriedad). Por otro lado, el mayor inconveniente es que la mejora ha sido muy leve (2.95 -> 2.93), pese a esto se puede observar como en la mayoría de los otros gaps esta última versión mejora a la segunda.

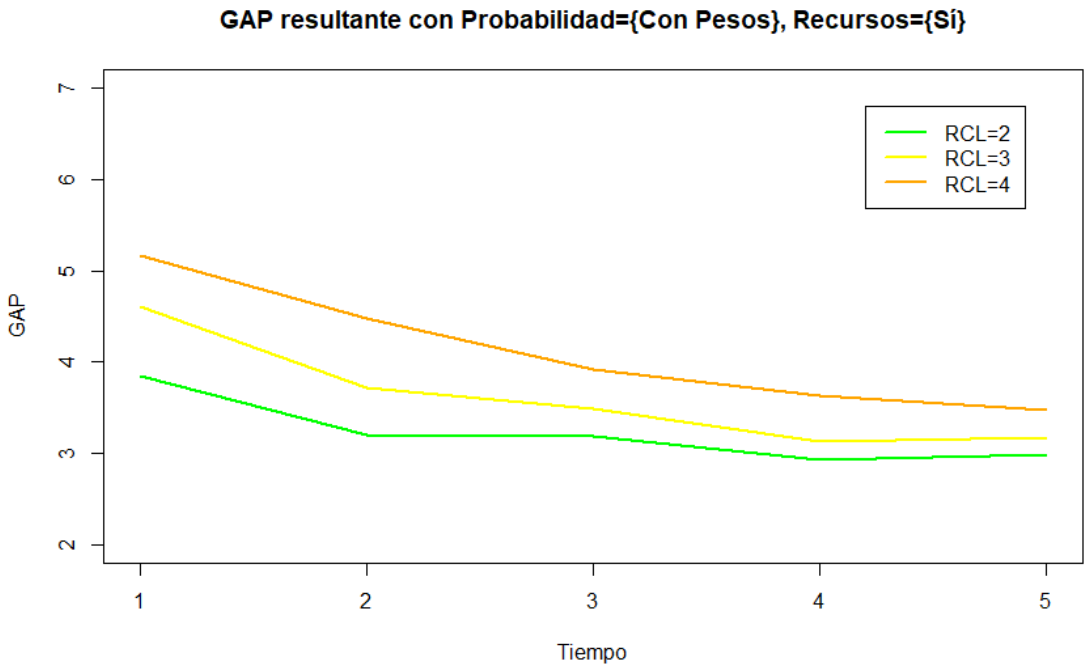

Figura 26: GAP resultante con probabilidad con pesos y teniendo en cuenta recursos

En esta última Figura 26 representativa de la evolución del GAP obtenido en el último experimento se puede observar la misma tendencia que con el segundo experimento donde no se tenían en cuenta los recursos. A pesar de que las gráficas parecen ser iguales, sí que se observa cierta mejoría en esta última ejecución, donde los GAPs son un poco inferiores como se puede observar en la siguiente Figura 27 si se superponen ambas gráficas. A pesar de que sean inferiores en la mayoría de los valores para t, cuando este valor es el máximo permitido para los experimentos, para el GAP acaba siendo prácticamente indiferente tanto si se tienen en cuenta los recursos como si no.

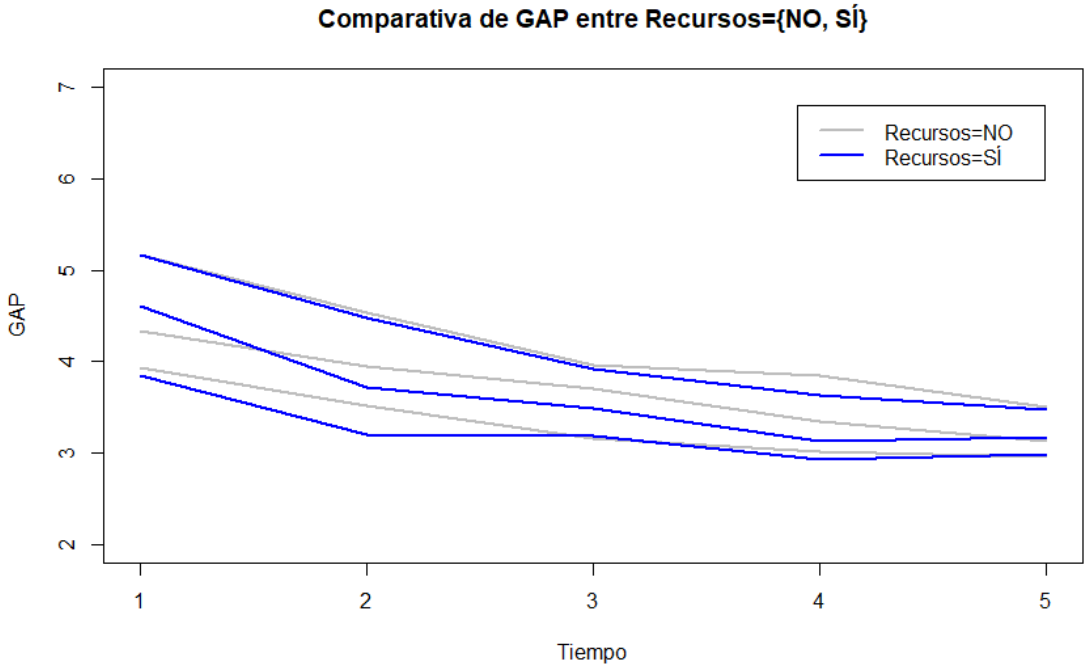

Figura 27: Comparativa de GAP entre tener en cuenta recursos o no en la fase constructiva

Para terminar con los experimentos y observar una mejor comparación entre el mejor GAP obtenido con el mejor $RCL = 2$, se van a superponer los valores de $RCL = 2$ para los cuatro experimentos realizados en la siguiente Figura 28:

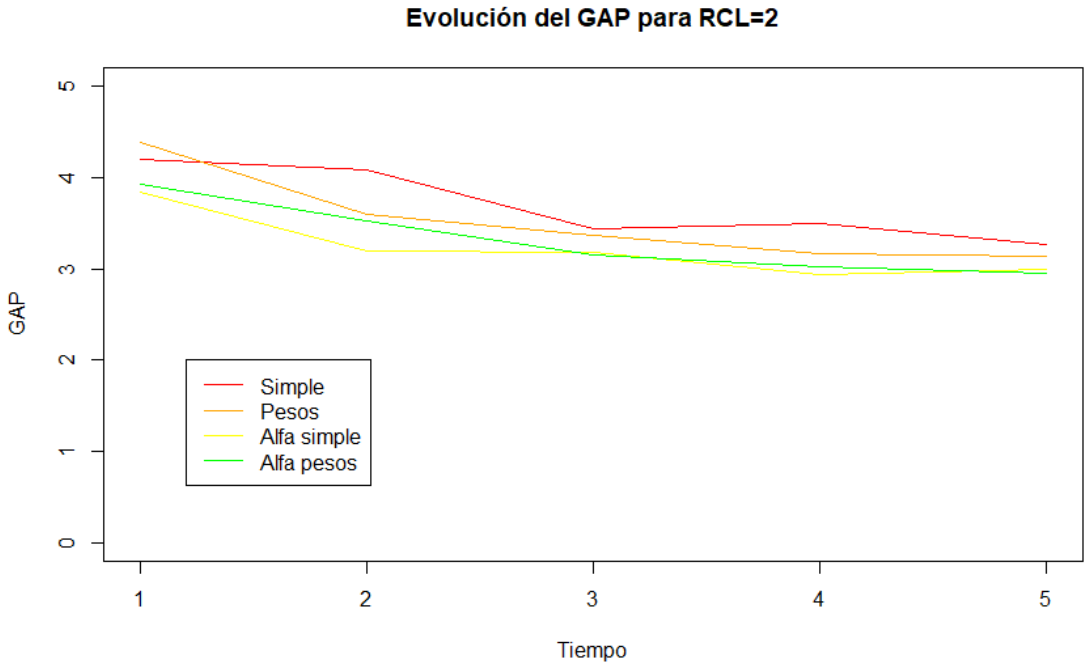

Figura 28: Evolución del GAP para RCL=2

En esta gráfica se muestra en el eje vertical el valor del GAP, y en el eje horizontal el factor tiempo t dividido en los 5 valores que puede tomar t para la fórmula del tiempo explicada anteriormente.

Cada punto de cada línea es la media de GAP para el valor t del tiempo en el eje horizontal y con $RCL = 2$. Estos puntos se unen en la línea de la gráfica que representan las cuatro ejecuciones para $RCL = 2$:

- Simple: ejecución aleatorizada con probabilidad de elección uniforme y sin tener en cuenta recursos.
- Pesos: probabilidad de elección según el valor del *makespan* sin tener en cuenta los recursos.
- Alfa simple: probabilidad de elección uniforme teniendo en cuenta los recursos para seleccionar los trabajos.
- Alfa pesos: probabilidad de elección con pesos teniendo en cuenta el *makespan* y los recursos.

En esta gráfica se puede observar como a pesar de quintuplicar el tiempo para obtener una solución, el GAP apenas baja en una unidad. El hecho de que se elija siempre entre los dos mejores candidatos dificulta la variabilidad de las soluciones obtenidas como muestra la gráfica, a pesar de que haya cuatro variaciones distintas de la misma heurística. Aun siendo poca la diferencia entre resultados, una conclusión importante es que las mejores opciones son las que consideran los recursos en la fase constructiva, no es una mejora muy destacada, pero es uniforme para cualquier valor de t que si se tienen en cuenta los recursos el GAP va a ser inferior en general. Por lo tanto, con una ejecución realizada, se puede concluir que dependiendo del tiempo disponible puede ser mejor la versión de alfa simple (para $t \in \{1,2,4\}$) o alfa con pesos ($t \in \{3,5\}$).

9. Conclusiones

En este proyecto se ha abordado el problema de la secuenciación de máquinas paralelas no relacionadas con tiempos de ajuste y recursos limitados asignados a las máquinas durante los ajustes y durante el procesamiento de los trabajos. En él se han comentado la necesidad de implementar algoritmos eficientes que puedan suponer a una entidad minimizar el *makespan* lo máximo posible para así ahorrar costes en su cadena de producción o en cualquier fase donde se precise el uso de secuenciación de máquinas.

Un modelo de programación matemática existente en la literatura resuelve eficientemente en pocos segundos el problema UPMSR para instancias pequeñas encontrando la solución óptima. Pero conforme el tamaño de la instancia va aumentando, las soluciones encontradas son peores y los tiempos de cómputo explotan, siendo el modelo de programación matemática una solución al problema no escalable. Es aquí donde se ha visto la importancia de los algoritmos heurísticos y metaheurísticos, que para instancias grandes y con mucho menor tiempo de cómputo el algoritmo diseñado consigue acercarse e incluso prácticamente igualar al modelo de programación matemática en las instancias con mayor volumen de trabajos y máquinas, con tiempos de cómputo muy reducidos. Y para instancias pequeñas y medianas empeoran un poco, pero siguen de cerca y también consiguen igualar al modelo obteniendo las soluciones óptimas en algunos casos.

Es en los experimentos realizados con la aleatorización donde se han visto las mayores mejoras en cuanto al GAP y de donde se podría seguir investigando para rebajarlo lo máximo posible. Ya que aunque los diferentes experimentos sobre la heurística aleatoria no han rebajado demasiado el GAP obtenido con la aleatorización, sí que ha sido una mejora constante y todos estos experimentos del capítulo “Análisis de resultados” han arrojado resultados positivos.

9.1 Investigación futura

En cuanto a las opciones futuras para ampliar y mejorar el trabajo, estas podrían tener en cuenta los siguientes puntos:

- Mejorar el rendimiento en el futuro, ya que a pesar de que el GAPs es pequeño, en las instancias grandes se realizan muy pocas iteraciones en la aleatorización.
- Probar a realizar la aleatorización para un valor distinto del $\alpha = 0.7$.
- En la fase constructiva los trabajos han sido asignados según el *makespan* generado una vez estén asignados, en vez de por el que menos diferencia de *makespan* generen. Si por ejemplo ninguno de los trabajos por asignar supera el *makespan* existentente, no dará igual asignar uno u otro, se deberá asignar el que menos *makespan* genere a pesar de que haya un trabajo ya asignado que los supere en *makespan*. Lo mismo para la fase de reparación.

- Alternar los valores de los factores que se han propuesto o añadir nuevos para realizar más experimentos y poder ver como evoluciona el problema frente a estas variaciones.
- Plantear la escritura de un artículo científico basado en el trabajo realizado sobre la resolución del problema UPMSR.

10. Bibliografía

Ruíz-Torres, A., López, J., Ho, J., 2007. Scheduling uniform parallel machines subject to a secondary resource to minimize the number of tardy jobs. European Journal of Operational Research 179, 302-315.

Ruiz, R., Andrés-Romano, C., 2011. Scheduling unrelated parallel machines with resource-assignable sequence-dependent setup times. The international Journal of Advanced Manufacturing Technology 57, 777-794.

Edis, E.-B., Oguz, C., 2012. Parallel machine scheduling with flexible resources. Computers & Industrial Engineering 63, 433-447.

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

Fanjul, L., Perea, F., Ruiz, R. (2017) Exact algorithms and matheuristics for scheduling problems with additional resources. European Journal of Operational Research 260(2) 482--493.

Villa, F., Vallada, E., Fanjul-Peyro, L., 2017. Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource. Expert Systems with Applications 93, 28-38.

Arbaoui, T., Yalaoui, F., 2018. Solving the unrelated parallel machine scheduling problem with additional resources using constraint programming. Intelligent Information and Database Systems, 716-725.

Vallada, E., Villa, F., Fanjul-Peyro, L., 2019. Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem. Computers & Operations Research 111, 415-424.

Yepes-Borrero, J.-C., Villa, F., Pera, F., Caballero-Villalobos, J.-P., 2019. GRASP algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. Expert Systems With Applications 131, 112959.

Fanjul-Peyro, L., 2020. Models and an exact method for the unrelated parallel machine scheduling problem with setups and resources. Expert Systems with Applications 5, 100022.