UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Máster Universitario en Ingeniería de Análisis de Datos, Mejora de Procesos y Toma de Decisiones

Departamento de estadística e investigación operativa aplicadas y calidad

UNA METAHEURÍSTICA PARA UN PROBLEMA DE SECUENCIACIÓN CON NECESIDAD DE RECURSOS EN LOS AJUSTES

Juan Camilo Yepes Borrero

Tutores:
Federico Perea Rojas-Marcos y María Fulgencia Villa Julià

Valencia, 10 de julio 2017

Resumen

En este trabajo fin de máster trata el problema de máquinas paralelas no relacionadas con tiempos de ajuste dependientes de la secuencia y recursos escasos asignados a los tiempos de ajuste (UPMSR), teniendo como objetivo principal buscar un procedimiento heurístico que logre encontrar buenas soluciones para este problema. Se empieza por introducir diferentes problemas de secuenciación de máquinas paralelas y algunos trabajos relacionados a estos problemas, para luego introducir el problema UPMSR y finalmente presentar el algoritmo heurístico propuesto. Este algoritmo se verifica a partir de una experiencia computacional sobre instancias del problema generadas aleatoriamente.

Índice

1. Introducción

Actualmente, debido a las exigencias que tienen las empresas, los modelos de programación de la producción tienen un papel muy importante en la investigación operativa. La programación de la producción consiste, entre otras cosas, en la asignación de recursos a tareas durante el tiempo que este recurso procese la tarea buscando optimizar uno o varios objetivos. Por ejemplo, los recursos pueden ser grúas para el desembarco de contenedores en un puerto y las tareas pueden ser los barcos que deben descargar los contenedores en el puerto. En otro ejemplo, los recursos pueden ser máquinas en una fábrica y las tareas pueden ser pedidos de clientes a procesar (problemas de secuenciación).

A lo largo de este trabajo, se introducirá un tipo de problema de secuenciación, específicamente los problemas de máquinas paralelas, donde los recursos son máquinas y las tareas son trabajos a ser procesados por estas máquinas, y se buscarán métodos heurísticos para solucionar el problema de máquinas paralelas no relacionadas con tiempos de ajuste dependientes de la secuencia y recursos escasos asignados a los tiempos de ajuste (UPMSR). En estos problemas, conociendo las características básicas del problema, como qué tareas se van a realizar, con qué recursos se cuenta, los tiempos de proceso, tiempos de ajuste (SETUP) entre trabajos y recursos requeridos para procesar los ajustes, se busca la mejor asignación y secuenciación de trabajos para optimizar un objetivo. En este trabajo, el objetivo será la minimización del tiempo de terminación de la última máquina o makespan.

1.1. Motivación

Encontrar buenas soluciones buscando minimizar el makespan en el problema UPMSR puede ser de gran interés en la planificación de la producción, sin embargo, debido a su complejidad, no es posible obtener buenas soluciones de manera rápida. Garey and Johnson (1979) ya demostraron que el problema de máquinas paralelas, en su forma más simple, que es el de máquinas paralelas idénticas, es parte de los

problemas llamados NP-Hard, es decir, la complejidad computacional va aumentando de forma polinomial con el tamaño del problema, por lo que los métodos exactos dejan de funcionar muy rápidamente a medida que aumentan los parámetros del problema. Por ello, es necesario buscar procedimientos rápidos para encontrar buenas soluciones según el criterio que se desee optimizar. Existen múltiples trabajos en los que se proponen procedimientos heurísticos para diversas variaciones del problema de máquinas paralelas (Sección 3). Sin embargo, para el problema a tratar en este trabajo, no se encuentra ningún trabajo previo, por lo que es de gran interés encontrar un algoritmo eficiente para este problema.

1.2. Objetivos

De acuerdo a lo comentado previamente, los objetivos de este trabajo son los siguientes:

- Introducir el problema UPMSR y presentar un modelo matemático para éste.
- Diseñar un algoritmo heurístico que mediante algunas reglas de asignación y reparación, logre encontrar buenas soluciones factibles al UPMSR.
- Aleatorizar el algoritmo para conseguir mayor variedad de soluciones factibles, y así mejorar las soluciones encontradas previamente.

A continuación se describe de forma general la estructura y el contenido de este trabajo:

- Sección 2. Problemas de secuenciación: Se definirán los problemas de secuenciación y se describirán las diferentes variaciones del problema de máquinas paralelas.
- Sección 3. Revisión bibliográfica: Se hará un resumen de los trabajos previos de problemas de máquinas paralelas que han sido consultados a la hora de hacer este trabajo.
- Sección 4. Definición formal del problema de máquinas paralelas no relacionadas con tiempos de ajuste dependientes de la secuencia y recursos escasos asignados a los tiempos de ajuste (UPMSR

por sus siglas en inglés): Se describirá formalmente el problema con la notación utilizada, y un modelo matemático para su resolución.

- Sección 5. Algoritmo propuesto: Se explicarán las fases del algoritmo heurístico propuesto para la solución del problema UPMSR.
- Sección 6. Experimentos computacionales: Se presentarán los resultados obtenidos por el algoritmo propuesto sobre un conjunto de instancias generadas aleatoriamente.
- Sección 7. Conclusiones y trabajo futuro: Se comentarán las conclusiones del trabajo y se hablará de los objetivos alcanzados. Además, se plantearán posibles líneas de trabajo futuras.
- Por último, una lista de la bibliografía consultada y el apéndice.

2. Problemas de secuenciación

En la actualidad, se pueden encontrar problemas de secuenciación y asignación en muchos ámbitos prácticos, como en plantas de producción, transporte, operaciones portuarias, entre muchos otras. Los problemas de asignación consisten en determinar de manera óptima, qué recursos (máquinas, operarios, herramientas, entre muchos otros) deben ser asignados a cada tarea para su realización, con el fin de optimizar un cierto objetivo. Por otro lado, los problemas de secuenciación consisten en determinar el orden en el que se realizan las tareas de tal forma que se optimice el criterio deseado. Cabe decir que en el caso de tener una única máquina, el problema de asignación es trivial, ya que todos los trabajos tendrían que ser procesados por esta máquina. Sin embargo, la secuencia puede no ser trivial, ya que el orden en el que se procesen los trabajos, podría afectar al tiempo final de terminación en caso de tener que ajustar la máquina entre dos trabajos. En este trabajo se considera el problema de secuenciación de máquinas paralelas con tiempos de ajuste dependientes de la secuencia y con recursos escasos necesarios para realizar esos ajustes.

2.1. Máquinas paralelas

En esta sección se hará una explicación más detallada sobre los problemas de secuenciación de máquinas paralelas, desde los modelos más simples hasta los modelos más complejos, que son el motivo de estudio en este trabajo. En estos problemas se tendrá un conjunto $N = \{1, \ldots, n\}$ de trabajos a procesar, los cuales serán procesados en una de las $M = \{1, \ldots, m\}$ máquinas disponibles. Un trabajo $j \in N$ procesado en la máquina $i \in M$ tendrá un tiempo de proceso p_{ij}. El tiempo de terminación de una máquina i, es decir, el instante de tiempo en el que termina de procesar todos los trabajos, lo llamaremos C_i. Si bien existen varios criterios que se pueden optimizar en este tipo de problemas, en este trabajo, el criterio que se busca optimizar es la minimización del makespan ($C_{\text{máx}}$) o tiempo en el que se procesa la última tarea en la máquina que termina más tarde, es decir, el máximo C_i.

2.1.1. Máquinas paralelas idénticas

El problema menos complejo es el de máquinas paralelas idénticas sin tiempos de ajuste. En este caso, el tiempo de proceso de un trabajo es igual en todas las máquinas, es decir, $p_{1j} = p_{2j} = \ldots = p_{mj}$, por lo que minimizar el makespan equivaldría a equilibrar la carga de trabajo en las m máquinas Fanjul-Peyró (2010) .

Ejemplo 2.1. A continuación, se muestra un ejemplo con tres máquinas paralelas idénticas. La Tabla 1 muestra los tiempos de proceso p_{ij} donde se puede ver que los tiempos de proceso para cada trabajo j son iguales en las tres máquinas.

	J1	**J2**	**J3**	**J4**
M1	3	4	2	7
M2	3	4	2	7
M3	3	4	2	7

Tabla 1: p_{ij} para tres máquinas idénticas.

La Figura 1 muestra diferentes soluciones factibles al problema en las que se asignan los trabajos con la intención de minimizar el makespan. Se puede observar que al ser idénticas las máquinas, es indiferente si los trabajos 2 y 3 se procesan en la máquina 2 o en la máquina 3, así como si el trabajo 1 se procesa en la máquina 3 o en la 2. Otro detalle importante a destacar en este problema, es que no importa el orden en el que se procesen los trabajos en cada máquina, ya que el objetivo a lo largo de este trabajo es el de minimizar el makespan, y en este caso, procesar un trabajo antes que otro en una máquina, no cambiaría el tiempo de terminación C_i en ésta. Por ejemplo, en la segunda solución de la Figura 1, el tiempo de proceso total de la máquina 3 es igual si se procesa primero el trabajo 3 que si se procesa el trabajo 2. En este ejemplo, el makespan es de 7 en ambas soluciones y se puede ver fácilmente que son soluciones óptimas, ya que el tiempo de proceso del trabajo 4 es de 7, y en ningún caso el makespan podría ser menor a este valor.

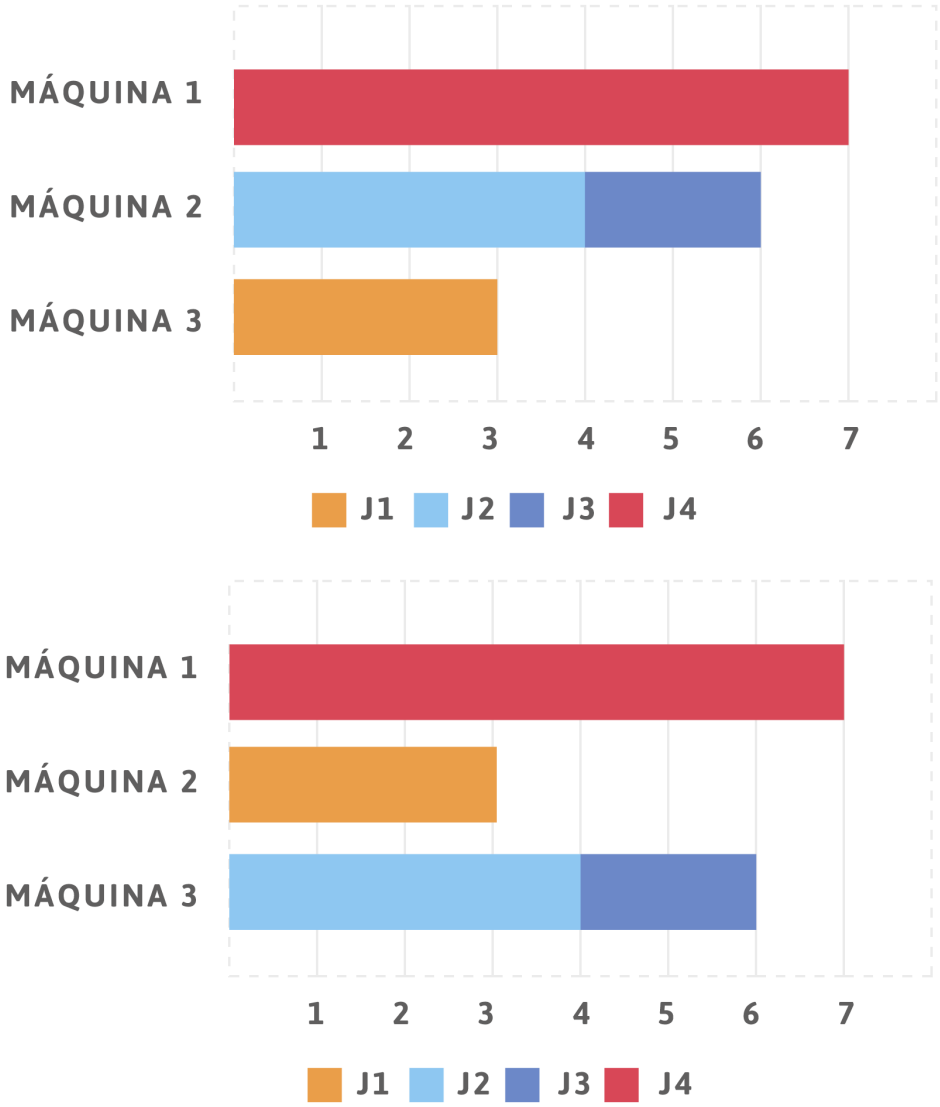

Figura 1: Ejemplo de tres máquinas idénticas.

2.1.2. Máquinas paralelas uniformemente relacionadas

El problema de máquinas paralelas uniformemente relacionadas es una generalización del caso anterior, variando en que las máquinas presentan velocidades de procesamiento de los trabajos distintas y

proporcionales entre ellas. Por ejemplo, los trabajos pueden tardar la mitad del tiempo en ser procesados en una máquina que sea el doble de rápida que otra. En este caso, una posible solución sería ubicar los trabajos con tiempos de proceso más largos en las máquinas más rápidas para minimizar el makespan.

Ejemplo 2.2. A continuación se muestra un ejemplo para ilustrar la idea:

	J1	**J2**	**J3**	**J4**
M1	2	4	6	8
M2	1	2	3	4
M3	4	8	12	16

Tabla 2: p_{ij} para tres máquinas uniformemente relacionadas.

En la Tabla 2 se muestran los tiempos de proceso de cada trabajo en cada máquina. Se puede observar que la máquina 1 es el doble de rápida que la máquina 3 y a su vez, la máquina 2 es el doble de rápida que la máquina 1.

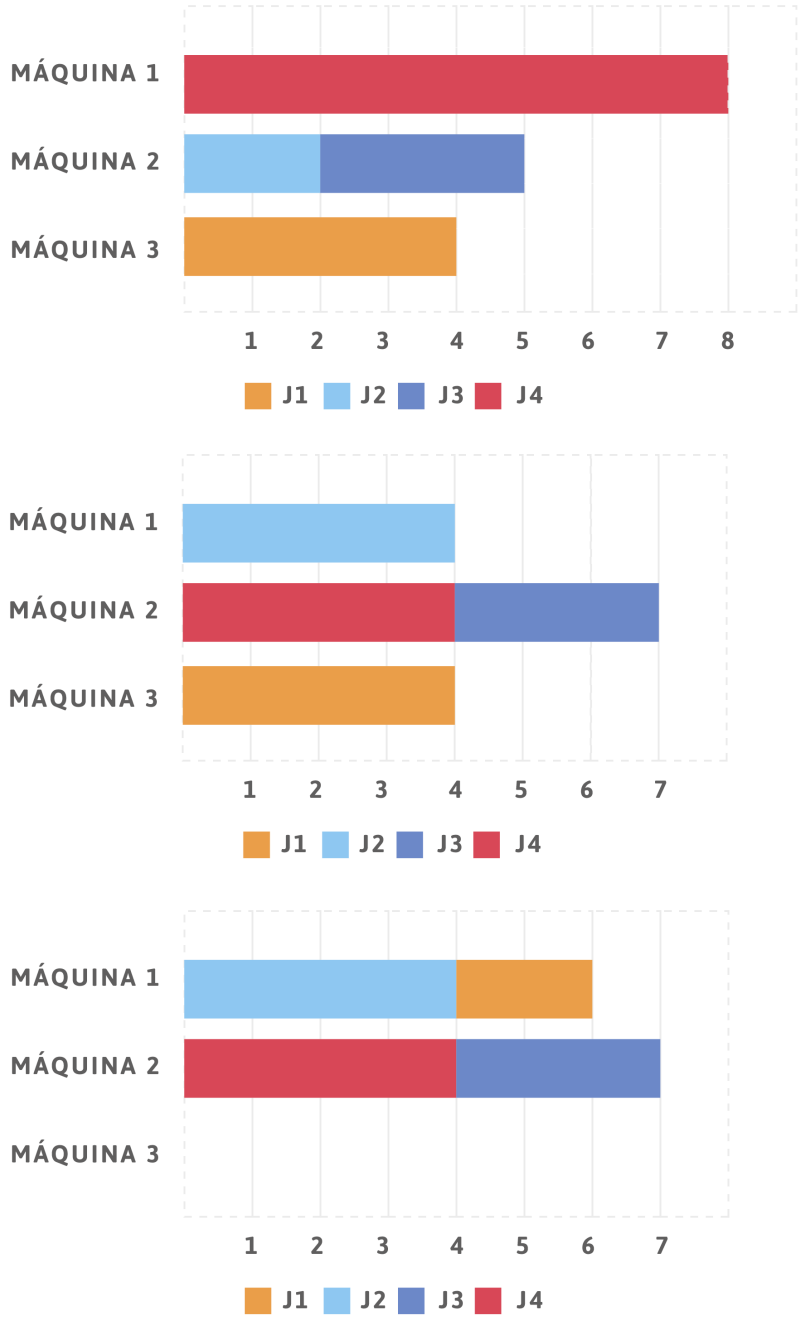

Figura 2: Ejemplo de tres máquinas uniformemente relacionadas.

En la Figura 2 se observa la variación en el tiempo de proceso de los trabajos dependiendo de la máquina en la que se procese. Al igual que en el problema de máquinas idénticas, no importa el orden en el que se procesen los trabajos asignados a cada máquina, pero en este caso si va a importar en que máquina se procese cada trabajo, ya que al ser diferente el tiempo de proceso de un trabajo j dependiendo de la máquina en la que es asignado, se buscará encontrar la asignación óptima de cada trabajo en la que se minimice el makespam. En el ejemplo se observa que el trabajo 4, al ser el trabajo con mayor tiempo de proceso, debe ubicarse en la máquina donde menos tiempo tarda en ser procesado para mejorar la solución.

2.1.3. Máquinas paralelas no relacionadas

En este caso se tienen tiempos de proceso diferentes de los trabajos dependiendo de la máquina en la que sean procesados. En este problema, a diferencia del anterior, los tiempos de proceso no están relacionados y se buscará encontrar la asignación ideal de cada trabajo j para minimizar el makespan.

Ejemplo 2.3. A continuación, la Tabla 3 y la Figura 3 muestran un ejemplo de un problema de secuenciación de máquinas paralelas no relacionadas:

	J1	**J2**	**J3**	**J4**
M1	5	6	7	4
M2	7	7	5	9
M3	4	7	11	9

Tabla 3: p_{ij} para tres máquinas no relacionadas.

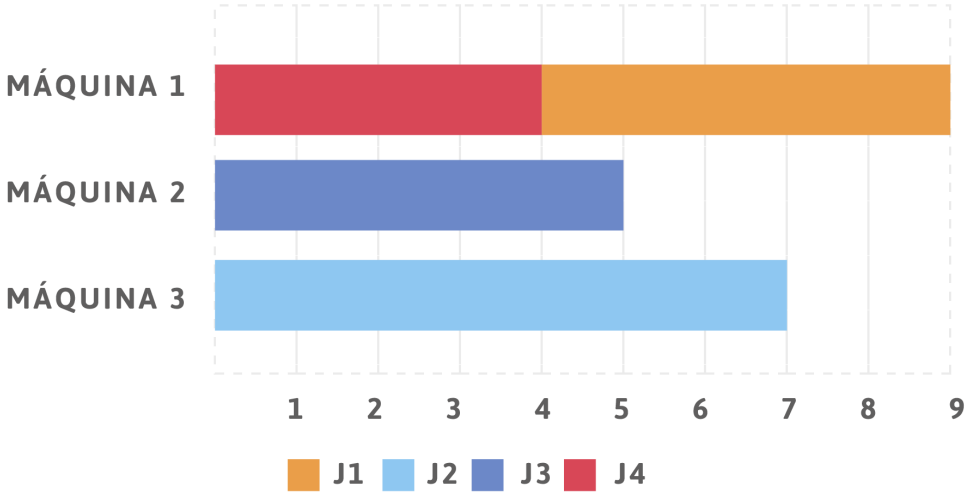

Figura 3: Ejemplo para tres máquinas no relacionadas.

Cabe mencionar que, al igual que en los dos casos anteriores, en este tampoco importará el orden en el que se procesen los trabajos dentro de la máquina en la que han sido asignados.

2.1.4. Máquinas paralelas con tiempo de ajuste dependiente de la secuencia (UPMS)

A diferencia de los problemas anteriores, en este caso se deben ajustar las máquinas entre la realización de dos trabajos consecutivos. Por lo tanto, no sólo hay que decidir qué trabajos se procesan en cada máquina sino también el orden de ejecución, ya que el tiempo de ajuste puede ser diferente dependiendo de qué trabajos se vayan a procesar consecutivamente, es decir, si definimos s_{ijk} como el tiempo de ajuste requerido para procesar el trabajo k inmediatamente después del trabajo j en la máquina i, este tiempo s_{ijk} no necesariamente es igual a s_{ikj}.

Ejemplo 2.4. Para comprender mejor este problema, se muestra a continuación un ejemplo de máquinas paralelas con tiempo de ajuste dependiente de la secuencia:

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

Tabla 4: p_{ij} para dos máquinas no relacionadas con tiempos de ajuste.

	J1	**J2**	**J3**	**J4**
J1	0	1	5	2
J2	6	0	5	6
J3	4	6	0	4
J4	2	5	2	0

Tabla 5: s_{1jk} para cada par de trabajos j y k en la máquina 1.

	J1	**J2**	**J3**	**J4**
J1	0	2	6	3
J2	5	0	2	6
J3	4	3	0	4
J4	6	4	6	0

Tabla 6: s_{2jk} para cada par de trabajos j y k en la máquina 2.

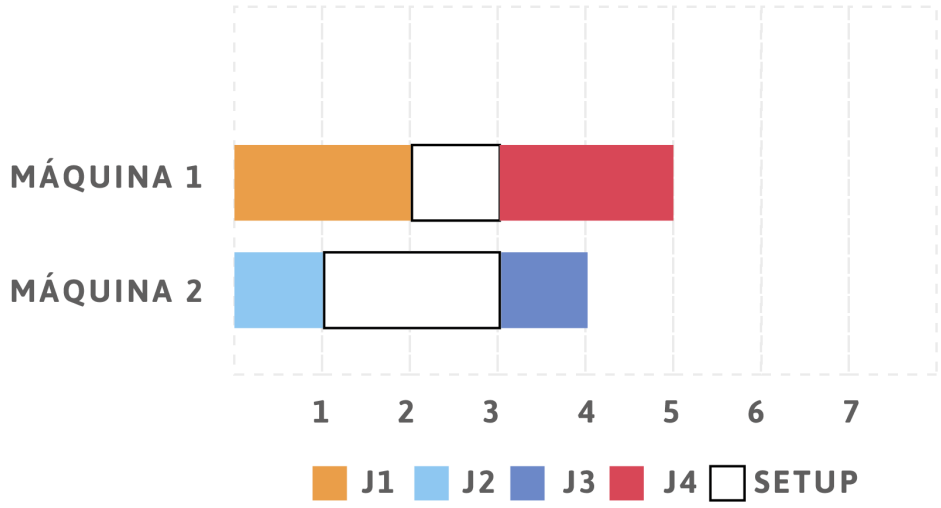

Figura 4: Ejemplo de dos máquinas no relacionadas con tiempos de ajuste.

La Tabla 4 muestra los tiempos de proceso de cada trabajo j en cada máquina i. Las Tablas 5 y 6 muestran los tiempos de de ajuste de la máquina 1 y la máquina 2 respectivamente. En estas tablas se representa el tiempo de ajuste requerido entre dos trabajos consecutivos, en las filas el trabajo que se hará inicialmente y en las columnas el trabajo que se realiza inmediatamente después. Por ejemplo, realizar el trabajo 2 inmediatamente después del trabajo 1 en la máquina 1, requiere un ajuste de 2 unidades de tiempo, mientras que si se procesa el trabajo 1 inmediatamente después del trabajo 2 en esta misma máquina, el tiempo de ajuste necesario será de 6 unidades de tiempo. En la Figura 4 se muestra gráficamente una posible solución a este ejemplo. Señalar que, en este problema, no sólo es importante la asignación de los trabajos a las máquinas, sino también el orden en el que estos sean procesados en ellas, ya que el tiempo de ajuste entre trabajos puede ser diferente dependiendo del orden de los trabajos a procesar.

2.1.5. Máquinas paralelas no relacionados con tiempo de ajuste dependiente a la secuencia y recursos escasos asignados a los tiempos de ajuste (UPMSR)

Por último, el problema que se trata en este trabajo es el de máquinas paralelas con tiempo de ajuste dependiente de la secuencia y recursos escasos asignados a los tiempos de ajuste (UPMSR) que, a diferencia del problema de la Sección 2.1.4, presenta la restricción de recursos limitados para realizar los ajustes. Por lo que ahora, además de tener un tiempo de ajustes entre los trabajos procesados, se tendrá un número de recursos requeridos para realizar este ajustes. Definimos r_{ijk} como la cantidad de recursos requeridos para hacer el ajuste necesario en la máquina i para poder procesar el trabajo k inmediatamente después del trabajo j, y $R_{\text{máx}}$ como el número máximo de recursos disponibles.

Ejemplo 2.5. A continuación, se continúa con el ejemplo anterior agregando los recursos limitados:

	J1	**J2**	**J3**	**J4**
J1	0	2	1	2
J2	3	0	3	1
J3	1	2	0	3
J4	1	1	4	0

Tabla 7: r_{1jk} para dos máquinas no relacionadas con tiempos de ajuste.

	J1	**J2**	**J3**	**J4**
J1	0	1	3	2
J2	2	0	2	2
J3	1	2	0	1
J4	1	2	3	0

Tabla 8: r_{2jk} para dos máquinas no relacionadas con tiempos de ajuste.

Las Tablas 7 y 8 muestran los recursos requeridos para cada ajuste entre cada par de trabajos, en la máquna 1 y en la máquina 2, respectivamente. Por ejemplo, para procesar el trabajo 2 inmediatamente

después del trabajo 1 en la máquina 1 se requieren 2 recursos para hacer el ajustes entre estos dos trabajos, mientras que si se procesan esos mismos trabajos en ese orden en la máquina 2, se requerirá 1 recurso para procesar el ajustes entre ellos.

Ahora se supondrá que el total de recursos disponibles ($R_{\text{máx}}$) es 3 y, al igual que antes, se busca encontrar la secuencia que minimice el makespam. Si se utiliza la misma secuencia que en el caso anterior sin tener en cuenta recursos se obtiene la solución de la Figura 5. En ella se observa que durante el instante de tiempo 2, la restricción de recursos no se cumple, ya que en ese momento se están procesando dos ajustes simultáneos que requieren 2 recursos cada uno, por lo que se necesitarían 4, uno más de los disponibles. En este caso, esta solución no es factible, por lo que hay que buscar otras soluciones que sí lo sean, bien sea cambiando la asignación de los trabajos a las máquinas, cambiando la secuencia de procesamiento de trabajos en cada máquina, o simplemente aplazando la realización del ajustes hasta que se cuente con los recursos necesarios para procesarlo como se muestra en la Figuara 6.

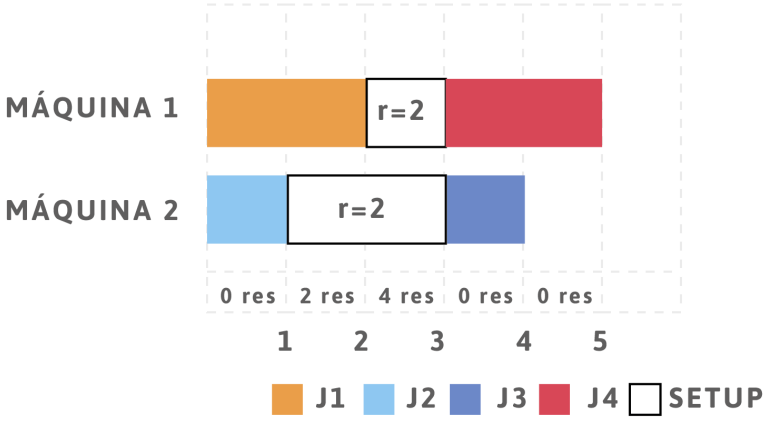

Figura 5: Solución no factible en ejemplo de problema UPMSR.

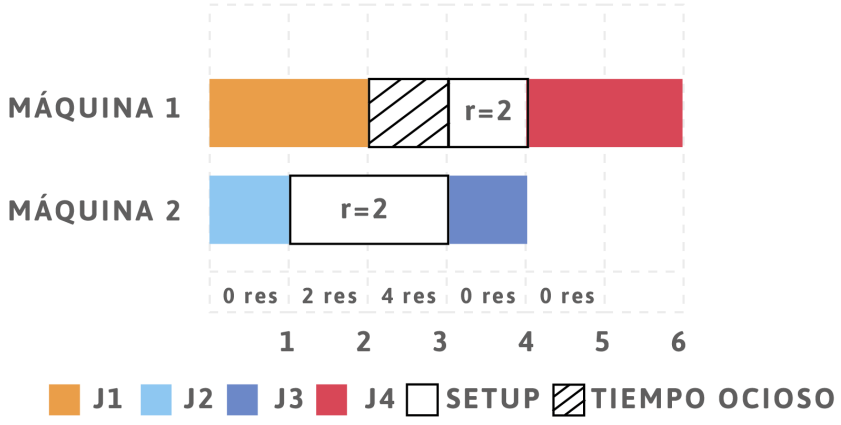

Figura 6: Solución factible en ejemplo de problema UPMSR.

3. Revisión bibliográfica

En esta sección, se hará un repaso sobre la bibliografía utilizada en la redacción del trabajo. Para empezar, sí bien es cierto que para el problema específico que se trabaja en esta memoria no hay mucha información, sí se pueden encontrar muchos trabajos sobre problemas de secuenciación de máquinas paralelas no relacionadas con las diferentes variaciones explicadas en la Sección 2. A continuación se hablará de algunos de los trabajos más relacionados con el problema trabajado en esta memoria.

Inicialmente, Kurz and Askin (2001), proponen cuatro heurísticas para el problema de máquinas paralelas relacionadas con tiempos de ajuste dependientes de la secuencia de trabajos. En la primera heurística proponen un método rápido en el que se soluciona el problema con sólo una máquina y se divide la secuencia obtenida en m (número de máquinas) grupos. La segunda heurística propuesta consiste en una adaptación del algoritmo MULTI-FIT propuesto por E. G. Coffman et al. (1978), con el cual se obtiene una secuencia inicial en cada una de las máquinas. Es importante mencionar que es necesario calcular un nuevo tiempo de proceso para cada trabajo sumando el tiempo de proceso y el mínimo de los tiempos de ajuste, ya que este algoritmo no considera tiempos de ajuste dependientes de la secuencia. Una vez obtenida esa secuencia, se calcula el tiempo de terminación de cada máquina (C_i) con los verdaderos tiempos de ajuste. Posteriormente, con la asignación obtenida en cada máquina, se resuelve para cada una el problema como un TSP (Problema del viajante) y se compara el nuevo C_i de cada máquina con el obtenido anteriormente, quedándose con la secuenciación que tenga menor C_i. La tercera heurística está formada por dos fases: en la primera fase se ordenan de manera no creciente los tiempos de proceso modificados igual que en la heurística anterior (sumando el tiempo de proceso con el mínimo tiempo de ajuste); en la segunda fase se prueba cada trabajo en cada posición de la máquina, ubicándolo en la posición donde el makespan parcial sea menor usando el tiempo real de ajuste. La cuarta y última heurística propuesta por estos autores, es un algoritmo genético en el que se genera de manera aleatoria la población inicial para luego evaluarla

y seleccionar los mejores individuos. Partiendo de estos mejores individuos, se realiza el cruce para obtener la nueva generación a la que luego se le realizará una mutación con una probabilidad p.

Chen (2005) trabaja sobre el problema de máquinas paralelas no relacionadas con tiempos de ajuste con recursos secundarios, en el que se incurre en tiempo de ajuste y en utilización de recurso (en este caso un molde) si el trabajo programado es de un tipo diferente al último trabajo procesado en esa máquina. Adicionalmente, algunos trabajos están restringidos a ser procesados únicamente en ciertas máquinas. El autor propone asignar cada trabajo a la máquina más eficiente, es decir, a la máquina donde el tiempo de proceso del trabajo sea menor, y ordenarlos por orden creciente, poniendo primero los trabajos que menor tiempo tardan en ser procesados. Una vez obtenida la asignación de los trabajos en las máquinas, se inicia una serie de reasignaciones directas, en las cuales se elige el último trabajo procesado por la máquina que tenga mayor tiempo de terminación y se reubica en otra máquina. Esta búsqueda local se hace con una lista tabú para evitar quedarse en un ciclo. Posteriormente, se inicia una serie de reasignaciones indirectas, en las cuales, al igual que en el paso anterior, se elige el último trabajo procesado por la máquina que tenga mayor tiempo de terminación y se reubica en otra máquina. De igual forma al paso anterior, se crea una lista tabú para evitar ciclos mientras se busca mejorar la solución.

Ruiz and Andrés-Romano (2011) trabajan el problema de máquinas paralelas no relacionadas con tiempos de ajustes dependientes de la secuencia y del número de recursos asignados al ajuste, ya que se pueden asignar más recursos a un ajuste para reducir el tiempo en el que se realiza. Los autores proponen tres heurísticas para este problema y posteriormente comparan sus resultados. La primera heurística propuesta consiste, en primer lugar, en comprobar para cada tarea cuál es la máquina en la que dicha tarea tarda menos tiempo en ser procesada, para luego, con esos tiempos, crear un vector ordenado de forma no decreciente e ir asignando en ese orden trabajos a la máquina en la que tarda menos tiempo en ser procesado. Por último, se asignan

los recursos al ajuste que se debe efectuar entre los trabajos procesados. La segunda heurística propuesta es una pequeña variación de la anterior, la diferencia está en que en el paso inicial, para saber a qué máquina asignar cada trabajo, no sólo se calcula el mínimo tiempo de proceso, sino que también se tiene en cuenta el tiempo de ajuste promedio del trabajo en cada máquina. Una vez obtenida la máquina en la que es mínimo el tiempo de proceso más el ajuste promedio de cada trabajo, se continúa con el mismo proceso de almacenar en un vector y ordenarlo en orden creciente para luego ir asignando los trabajos. La tercera heurística propuesta por los autores consiste en tomar un grupo de trabajos pendientes por asignar, e ir probando cada trabajo en cada máquina y calcular el valor de la función objetivo en cada prueba, para al final de cada iteración, después de probar todos los trabajos en todas las máquinas, asignar el trabajo que menos incremente la función objetivo. Por último los autores proponen la redistribución óptima de los recursos mediante un sencillo cálculo en el que se evalúan los ajustes en los que sería más "rentable" asignar más recursos.

Vallada and Ruiz (2011) proponen un algoritmo genético para el problema de máquinas paralelas no relacionadas con tiempos de ajuste dependientes de la secuencia (UPMS). En su algoritmo proponen una población inicial conformada por soluciones obtenidas mediante MI (Multiple Insertion) y otras aleatorias. Una vez seleccionada la población inicial, se procede a hacer los cruces haciendo una adaptación del *One Point Order Crossover* para el problema de máquinas paralelas, en el que para cada máquina del padre uno, se selecciona aleatoriamente un punto p y se copian en el primer hijo los trabajos asignados desde la posición uno hasta la posición p de cada máquina, y los trabajosldesde la posición $p+1$ en adelante, son copiados al segundo hijo. Luego, los trabajos del padre dos que no han sido asignados en el hijo, son asignados en el mismo orden en el que están y aplicando una búsqueda local probando cada trabajo por asignar en cada posición de la máquina en la que será asignado y dejándolo en la posición donde el makespam sea menor. Posterior a esto, se realiza una mutación con probabilidad p en la cual se selecciona al azar una máquina y dentro

de esa máquina un trabajo que será reubicado también al azar en otra posición de la misma máquina. Por último, se efectúa una búsqueda local probando cambiar cada trabajo en cada posición de cada máquina y dejándolo en la posición en la que el makespam sea menor.

Avalos-Rosales et al. (2014) proponen un algoritmo multipartida para el UPMS. La metodología multipartida propuesta consiste en generar varias soluciones iniciales diferentes y aplicar sobre ellas procesos de mejora. En primer lugar, para generar las soluciones iniciales, se ordena de forma no creciente un vector conformado por los tiempos de proceso promedio de cada trabajo en las máquinas disponibles y se eligen los s primeros elementos. A continuación, entre los s trabajos seleccionados, se escoge al azar uno de ellos y se asigna a la máquina donde antes acabe. Este proceso se repite hasta que todos los trabajos sean asignados. Por último, teniendo ya la secuenciación de los trabajos se realiza el proceso de mejora tomando cada trabajo y probándolo en cada una de las posiciones de las otras máquinas y ubicándolo en la posición donde el makespam sea mínimo.

Diana et al. (2015), diseñan un algoritmo basado en el algoritmo Clonal Selection propuesto por Castro y Von Zuben (2000). En su trabajo, proponen generar soluciones iniciales de la mejor calidad posible mediante un algoritmo GRASP introducido por Feo and Resende (1989), para luego hacer un proceso de clonación de soluciones iniciales en el que, cuanto mejor sea la solución inicial, más probabilidades de ser clonada y luego, con estas soluciones, aplicar mutaciones en busca de mejorar la solución.

Fanjul-Peyró et al. (2017) proponen dos programaciones lineales enteras para el problema de máquinas paralelas no relacionadas con recursos para procesar los trabajos. Estos modelos son capaces de resolver problemas pequeños en tiempos de computación razonables. Para problemas más grandes, ayudan a los modelos diseñando tres matheurísticas. La primera en la que inicialmente se resuelve el problema sin tener en cuenta los recursos para obtener la asignación luego secuenciar los trabajos de la mejor manera posible en cada máquina.

La segunda matheuristica propuesta se basa en que cada trabajo, solo puede ser asignado a las l mejores máquinas, mientras que la tercera matheuristica propuesta, soluciona el problema dividiéndolo y solucionando con un algoritmo greedy por grupos de trabajos.

Las matheurísticas diseñadas por Fanjul-Peyró et al. (2017) no son eficientes para resolver problemas grandes. Por ello, Villa et al. (2017) abordan la resolución del problema diseñando diferentes heurísticas utilizando dos estrategias. En la primera para generar una solución se tiene en cuenta la restricción de recursos en todo momento. En la segunda estrategia se utilizan diferentes heurísticas sin tener en cuenta la restricción de recursos y la solución se factibiliza al final mediante un proceso de reparación.

Por último, Perea et al. (2016) presentan la definición formal del problema estudiado en este trabajo, el UPMSR. En su trabajo presentan la ampliación del modelo con respecto al problema sin tener en cuenta recursos, pero no presentan ningún método heurístico para la resolución del problema. Encontar algoritmos eficientes para la resolución del problema UPMRS es el objetivo principal de este trabajo fin de máster.

La tabla 9 muestra las principales características de los trabajos citados en esta sección.

Autor	Máquinas paralelas	Setup	Recursos
Kurz and Askin (2001)	relacionadas	sí	no
Chen (2005)	no relacionadas	sí	Secundarios en los ajustes
Ruiz and Andrés-Romano (2011)	no relacionadas	sí	Secundarios en los ajustes
Vallada and Ruiz (2011)	no relacionadas	sí	no
Avalos-Rosales et al. (2014)	no relacionadas	sí	no
Diana et al. (2015)	no relacionadas	sí	no
Perea et al. (2016)	no relacionadas	sí	sí- asignados a los ajustes
Fanjul-Peyró et al. (2017)	no relacionadas	no	sí- asignados a los trabajos
Villa et al. (2017)	no relacionadas	no	sí- asignados a los trabajos

Tabla 9: Tabla resumen de la revisión bibliográfica

4. Definición formal del problema

En esta sección, se dará una definición más rigurosa del problema de máquinas paralelas con tiempos de ajuste dependientes de la secuencia y con recursos escasos asignados a los tiempos de ajuste (UPMSR). Además, se presentará el modelo matemático introducido en Perea et al. (2016), para una mejor comprensión del problema.

En primer lugar, es importante definir los conceptos de trabajos sucesores y predecesores.

Definición 4.1. Se dice que un trabajo k es sucesor de un trabajo j si los dos trabajos son procesados por la misma máquina i y entre j y k, la máquina i no procesa otro trabajo. Se dice que un trabajo j es predecesor de k si k es sucesor de j.

Definición 4.2. Dos trabajos j y k son sucesivos, en ese orden, si j es predecesor de k (o k es sucesor de j).

A continuación se presentan los datos necesarios para formular el problema UPMSR:

Conjuntos

- $M = \{1, \ldots, m\}$: Conjunto de máquinas indexadas por la letra i.
- $N = \{1, \ldots, n\}$: Conjunto de trabajos a procesar indexados por las letras j y k.

Parámetros

- p_{ij}: Tiempo de proceso del trabajo j, en la máquina i.
- s_{ijk}: Tiempo de ajuste requerido en la máquina i entre los trabajos j y k si k es sucesor de j.
- r_{ijk}: Número de recursos requeridos para hacer el ajuste necesario en la máquina i entre los trabajos sucesivos j y k.
- $R_{\text{máx}}$: Número de recursos disponibles.

- $t_{\text{máx}}$: Límite superior que puede tener el makespan.

Por motivos de modelado, es necesario definir el conjunto $N_0 = N \cup \{0\}$ donde 0 es un trabajo ficticio en el que inician y terminan todas las máquinas i. Nótese que $s_{i0k} = s_{ik0} = r_{i0k} = r_{ik0} = p_{i0} = 0$, $\forall i \in M, \forall k \in N_0$.

Una vez introducidos los datos de entrada de este problema, pasamos a modelarlo utilizando programación lineal entera. Para ello, hacen falta las siguientes variables:

- $C_{\text{máx}}$: Tiempo de terminación de la última tarea procesada en la última máquina en terminar.
- Y_{ij}: Variable binaria que toma valor 1 si el trabajo j es procesado en la máquina i, toma valor 0 en caso contrario.
- X_{ijk}: Variable binaria que toma valor 1 si el trabajo k es sucesor del trabajo j en la máquina i, toma valor 0 en caso contrario.
- H_{ijkt}: Variable binaria que toma valor 1 si el ajuste, en la máquina i, entre los trabajos sucesivos j y k, termina en el instante de tiempo t. Vale 0 en caso contrario.

El siguiente programa de programación lineal entera ($MILP$) modela el UPMSR:

$$\text{mín}\, C_{\text{máx}} \tag{1}$$

$$\text{s.t.:} \sum_{k \in N} X_{i0k} \leq 1,\ i \in M \tag{2}$$

$$\sum_{i \in M} Y_{ij} = 1,\ j \in N \tag{3}$$

$$Y_{ij} = \sum_{k \in N_0, j \neq k} X_{ijk},\ i \in M, j \in N \tag{4}$$

$$Y_{ik} = \sum_{j \in N_0, j \neq k} X_{ijk},\ i \in M, k \in N \tag{5}$$

$$\sum_{t \leq t_{\text{máx}}} H_{ijkt} = X_{ijk}, \forall i \in M, j \in N_0, k \in N, k \neq j \tag{6}$$

$$\sum_{t} t H_{ijkt} \geq \sum_{k' \in N_0} \sum_{t \leq t_{\text{máx}}} H_{ik'jt}(t + s_{ijk} + p_{ij}) - M(1 - X_{ijk}),$$
$$\forall\ i \in M, j \in N_0, k \in N, k \neq j \tag{7}$$

$$\sum_{i \in M, j \in N_0, k \in N, k \neq j, t' \in \{t, \ldots, t + s_{ijk} - 1\}} r_{ijk} H_{ijkt} \leq R_{\text{máx}}, \forall\ t \leq t_{\text{máx}} \tag{8}$$

$$\sum_{t \leq t_{\text{máx}}} t H_{ijkt} \leq C_{\text{máx}}, \forall i \in M, j \in N_0, k \in N_0, k \neq j \tag{9}$$

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

La expresión (1) representa el objetivo a minimizar (makespan). La restricción (2) garantiza que a lo más un trabajo sea asignado en la primera posición en cada máquina. La restricción (3) asegura que cada trabajo sea asignado a una y sólo una máquina. La restricción (4) asegura que cada trabajo j que se procesa en la máquina i tiene un único sucesor k. (5) asegura que cada trabajo k que se procesa en la máquina i, tiene un único predecesor j. La restricción (6) impone que para toda máquina i y para cada par de trabajos sucesivos j y k en i, el tiempo de ajuste entre j y k debe de terminar en un y sólo un instante anterior a $t_{\text{máx}}$. La restricción (7) asegura que el ajuste entre dos trabajos sucesivos j y k, tiene que terminar como muy pronto, cuando termine el ajuste anterior más el tiempo de proceso del trabajo j en la máquina correspondiente, más el tiempo de ajuste entre los trabajos j y k en la misma máquina. La restricción (8) asegura

que para todo instante de tiempo, el número de recursos utilizado no supera $R_{\text{máx}}$. Por último, la restricción (9) impone que el $C_{\text{máx}}$ tiene que ser igual o superior al instante final de todos los ajustes realizados, incluyendo el ajuste ficticio final entre el último trabajo procesado y el trabajo ficticio 0. Nótese que, por la estructura del problema, X y Y se pueden relajar.

5. Algoritmo propuesto

Debido a que el modelo propuesto en la Sección 4 sólo resuelve instancias de tamaño muy pequeño, en esta sección se presenta un algoritmo heurístico para la resolución del problema UPMSR. El algoritmo se divide en dos partes principales; una parte constructiva, en la cual se busca una buena solución inicial y en la que no se tienen en cuenta los recursos, y otra parte de reparación, en la cual se hace factible la solución obtenida en la parte constructiva en caso de no serlo.

En las Secciones 5.1 y 5.2 se explican las dos fases del algoritmo, mientras que en la Sección 5.3 se explica la aleatorización en la asignación de los trabajos para obtener mayor variedad de soluciones.

5.1. Método constructivo

El método constructivo propuesto en este trabajo está basado en un algoritmo GREEDY dividido en dos fases. En la primera fase se asigna en la primera posición de cada máquina, el trabajo que, en promedio, requiera mayor tiempo de ajuste para ser procesado en esa máquina. La segunda fase consiste en un proceso MI (Multiple Insertion) en el que se va probando cada trabajo en cada máquina. Una vez probados todos los trabajos en todas las máquinas, se asigna el trabajo que, al ser asignado, termina de ser procesado lo antes posible. Este proceso se repite hasta que la lista de trabajos por asignar esté vacía.

El Algoritmo 1 muestra el pseudocódigo de la primera fase del proceso constructivo. Llamaremos $\bar{s}_{ik}$ al tiempo promedio de ajuste en la máquina i del trabajo k con sus predecesores, $\bar{C}_i$ al tiempo de terminación de la máquina i y k_i al último trabajo procesado por la máquina i.

Primera asignación;
Crear Conjunto de trabajos a asignar N con todos los trabajos
for $i \in M$ **do**
  for $k \in N$ **do**
    Calcular $\bar{s}_{ik} = \frac{\sum_{j \in N} s_{ijk}}{n-1}$
  end
  Asignar a i el trabajo con mayor $\bar{s}_{ik} \rightarrow k_i$
  $\bar{C}_i = p_{ik_i}$
  $N = N - \{k_i\}$
end

Algoritmo 1: Primera fase proceso constructivo.

Ejemplo 5.1. A continuación se muestra un ejemplo con 8 trabajos y 3 máquinas ilustrando el proceso constructivo del algoritmo. La Tabla 10 muestra los tiempos de proceso, y las Tablas 11, 12 y 13 los tiempos de ajuste entre el trabajo fila y el trabajo columna en cada máquina. La última fila de las tablas muestra el tiempo de ajuste promedio entre el trabajo columna y sus predecesores .

	J1	**J2**	**J3**	**J4**	**J5**	**J6**	**J7**	**J8**
M1	18	18	13	9	11	10	5	1
M2	12	10	15	19	6	17	6	10
M3	18	17	2	19	3	3	17	8

Tabla 10: p_{ij} para tres máquinas y ocho trabajos.

	J1	**J2**	**J3**	**J4**	**J5**	**J6**	**J7**	**J8**
J1	0	7	8	9	5	5	8	5
J2	6	0	7	8	7	6	5	9
J3	8	9	0	7	6	8	5	9
J4	6	9	9	0	9	5	7	8
J5	7	8	8	6	0	7	7	9
J6	8	5	7	8	8	0	9	8
J7	8	7	6	5	8	6	0	9
J8	7	9	6	8	5	6	9	0
$\bar{s}_{ik}$	7.14	7.71	7.28	7.28	6	6.86	7.43	8.14

Tabla 11: Tiempos de ajuste en la máquina 1.

	J1	**J2**	**J3**	**J4**	**J5**	**J6**	**J7**	**J8**
J1	0	5	9	6	5	6	9	8
J2	5	0	6	7	8	7	8	6
J3	9	6	0	7	5	6	5	9
J4	7	5	8	0	7	6	7	8
J5	6	9	5	7	0	6	9	8
J6	9	5	6	7	6	0	6	6
J7	6	7	9	6	6	7	0	5
J8	9	6	6	8	7	8	6	0
$\bar{s}_{ik}$	7.28	6.14	7	6.86	6.28	6.57	7.14	7.14

Tabla 12: Tiempos de ajuste en la máquina 2.

	J1	J2	J3	J4	J5	J6	J7	J8
J1	0	7	7	6	5	7	8	6
J2	5	0	7	8	9	5	6	7
J3	6	6	0	5	5	6	7	6
J4	7	5	6	0	9	7	8	8
J5	8	5	9	6	0	8	6	8
J6	8	8	5	8	8	0	9	6
J7	9	8	7	7	9	8	0	6
J8	9	9	5	8	7	8	6	0
$\bar{s}_{ik}$	7.43	6.86	6.57	6.86	7.43	6.99	7.14	6.70

Tabla 13: Tiempos de ajuste en la máquina 3.

Para empezar, se busca en la máquina 1 el trabajo que mayor tiempo de ajuste promedio requiere. En este caso, sería el trabajo 8, por lo que se asigna en la primera posición de la máquina 1. Este trabajo se extrae de la lista N de trabajos por asignar. Una vez actualizada la lista N de trabajos por asignar, se repite el mismo procedimiento con las máquinas 2 (se le asigna J1) y 3 (se le asigna J5) obteniendo la asignación que se muestra en la Figura 7.

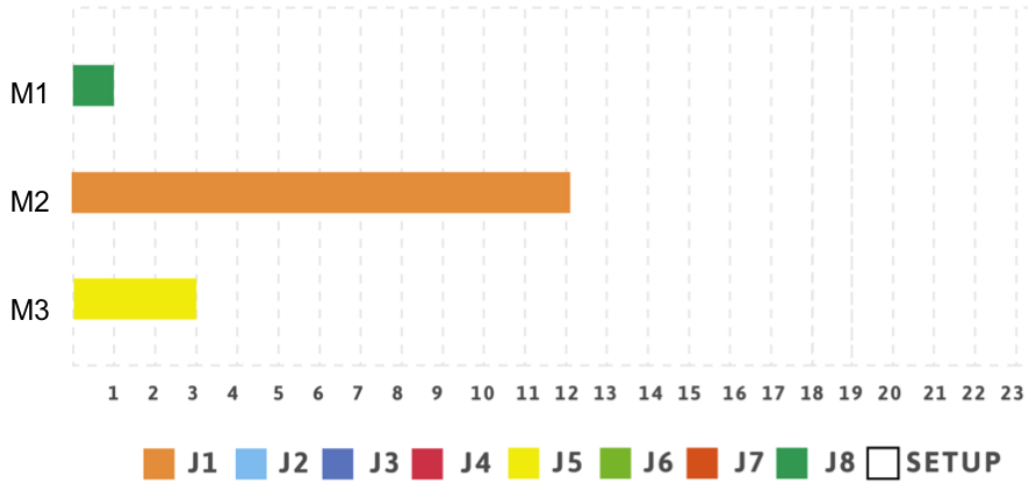

Figura 7: Primer trabajo asignado en cada máquina.

Una vez asignado el primer trabajo a cada máquina, se continúa con la segunda fase del proceso constructivo del algoritmo. A continuación se muestra el pseudocódigo de la segunda fase del proceso constructivo, donde C_{ij} es el tiempo de terminación de la máquina i asignando el trabajo j en ella.

while $N \neq \emptyset$ **do**
  for $j \in N$ **do**
    for $i \in M$ **do**
      Asignar j a i
      $C_{ij} = \bar{C}_i + s_{ik_ij} + p_{ij}$
    end
  end
  Asignar el par (i, j) que menor C_{ij} tiene $\rightarrow (i^*, k^*)$
  $N = N - \{k^*\}$
  $\bar{C}_i^* = \bar{C}_i^* + s_{i^*k_{i^*}k^*} + p_{ik_{i^*}}$
  $k_{i^*} = k^*$
end

Algoritmo 2: Segunda fase proceso constructivo.

Ejemplo 5.2. Siguiendo con el ejemplo anterior, una vez terminada la primera fase del proceso constructivo y se ha asignado el primer trabajo de cada máquina, se empieza a probar cada trabajo aún no asignado del conjunto N en cada máquina, y se busca el trabajo que termine de ser procesado más pronto. En este caso, al probar todos los trabajos restantes en cada máquina, se obtiene que el trabajo que terminaría más pronto sería el trabajo 3 asignado en la máquina 3, que tendría un tiempo de terminación de 14. A continuación se asigna este trabajo a esa máquina como se muestra en la Figura 8.

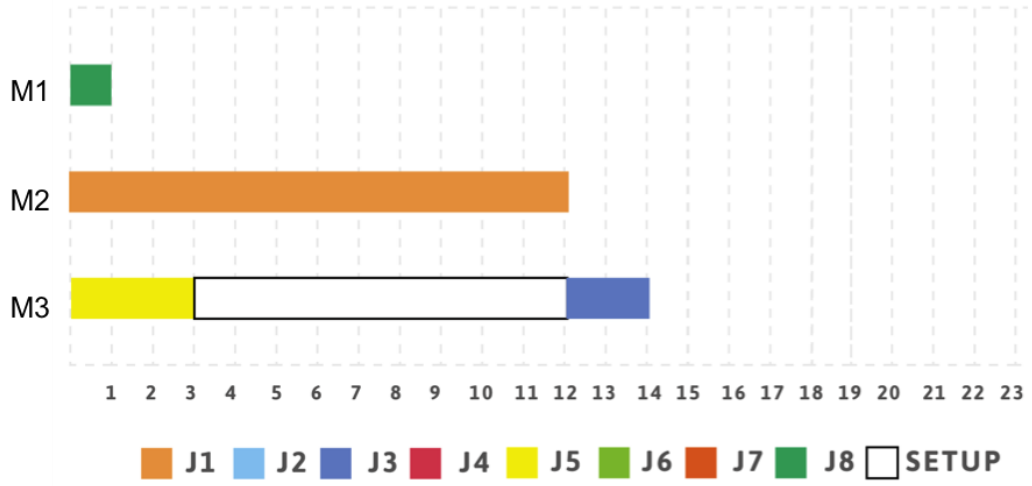

Figura 8: Paso 2 del método constructivo.

En este caso el trabajo 3 se quita de la lista de trabajos por asignar y se repite el proceso hasta que no quede ningún trabajo por asignar en la lista N. El resultado de esta fase viene expresado en la Figura 9, con un makespan de 29.

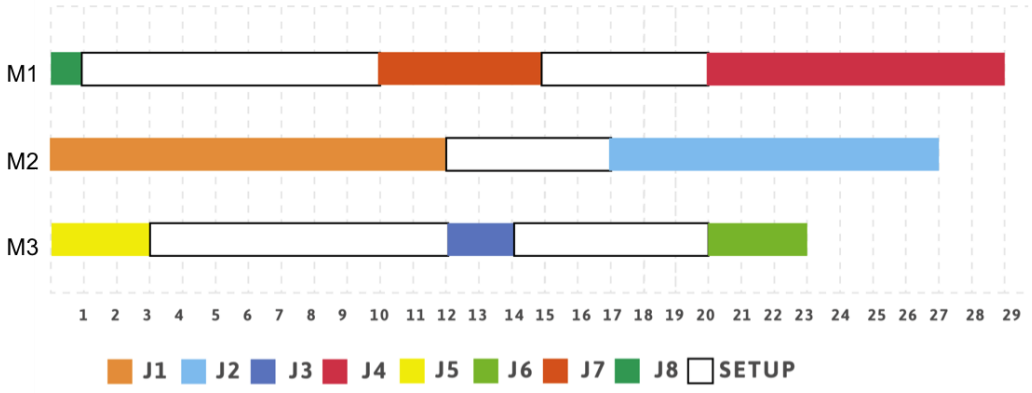

Figura 9: Final método constructivo.

5.2. Método de reparación

Una vez obtenida la solución inicial, se debe revisar cuántos recursos consume cada ajuste procesado en cada instante de tiempo. En caso de que los recursos utilizados en todos los instantes de tiempo no superen en ningún momento $R_{\text{máx}}$, la solución dada por el algoritmo constructivo es una solución factible y no será necesario repararla. Por otro lado, si esta restricción no se cumple en algún momento, se procederá a efectuar el método de reparación de la solución.

El proceso de reparación consiste en evaluar en cada instante de tiempo, la suma de recursos utilizados por las M máquinas, para que en caso de necesitar más recursos de los disponibles, posponer la ejecución del ajuste que al ser retrasado suponga un menor C_i. Una vez hecho este movimiento, se repite el proceso de evaluación desde el instante de tiempo t donde se había generado el problema.

El Algoritmo 3 muestra el pseudocódigo del proceso de reparación de la solución, donde $\bar{M}_t$ es el conjunto de máquinas que realizan ajuste en t:

$t = 0$; $C_{\text{máx}}$ = makespan de la solución inicial
while $t < C_{\text{máx}}$ **do**
  R_t =número de recursos usados en el instante de tiempo t
  while $R_t > R_{\text{máx}}$ **do**
    for $i \in \bar{M}_t$, **do**
      Mover el inicio del ajuste de i hasta que haya un solape menos.
    end
    Guardar movimiento con menor C_i
    Actualizar R_t
    Actualizar $C_{\text{máx}}$
  end
  $t++$;
end

Algoritmo 3: Proceso de reparación

Ejemplo 5.3. Para ilustrar el proceso de reparación, se continúa con el ejemplo anterior suponiendo $R_{\text{máx}} = 3$. Las Tablas 14, 15 y 16 muestran los recursos necesarios para realizar el ajuste de cada máquina

entre el trabajo fila y el trabajo columna

	J1	**J2**	**J3**	**J4**	**J5**	**J6**	**J7**	**J8**
J1	2	3	1	3	1	1	3	1
J2	3	3	3	1	1	1	1	3
J3	1	3	3	3	1	1	1	1
J4	3	2	3	1	2	2	2	3
J5	3	3	1	3	3	3	2	2
J6	2	3	3	3	1	1	2	3
J7	1	2	1	3	1	3	3	1
J8	1	2	3	2	2	2	1	1

Tabla 14: Recursos requeridos en la máquina 1.

	J1	**J2**	**J3**	**J4**	**J5**	**J6**	**J7**	**J8**
J1	2	2	2	1	3	3	2	1
J2	2	2	3	1	1	1	3	2
J3	3	2	2	2	3	2	2	3
J4	2	1	2	3	2	1	2	3
J5	3	1	3	1	3	3	3	1
J6	1	1	2	3	2	2	3	3
J7	2	1	1	2	3	2	2	1
J8	3	2	3	2	3	3	3	1

Tabla 15: Recursos requeridos en la máquina 2.

	J1	J2	J3	J4	J5	J6	J7	J8
J1	0	1	1	1	3	3	1	3
J2	1	0	1	3	1	2	3	1
J3	1	2	0	2	1	3	1	2
J4	1	1	1	0	2	1	3	1
J5	1	3	1	2	0	1	3	1
J6	3	1	2	2	1	0	1	1
J7	3	3	2	3	3	3	0	3
J8	3	2	2	2	2	2	1	0

Tabla 16: Recursos requeridos en la máquina 3.

La Figura 10 representa la solución obtenida con el método constructivo. En la solución se observa en blanco los ajustes que se procesan con los recursos requeridos para su proceso. Si se va evaluando el cumplimiento de los recursos desde el instante de tiempo 0 en adelante, en el instante 14 se están utilizando 5 recursos (dos en la máquina 2 y tres en la máquina 3), por lo que estamos ante una solución no factible y hay que repararla. El proceso de reparación propuesto prueba aplazando el inicio de cada ajuste que se solapa en el instante de tiempo donde no se cumple la restricción de recursos. El ajuste aplazado no inicia hasta que termine el ajuste de la otra máquina y se guardará el movimiento de la máquina que suponga un menor C_i.

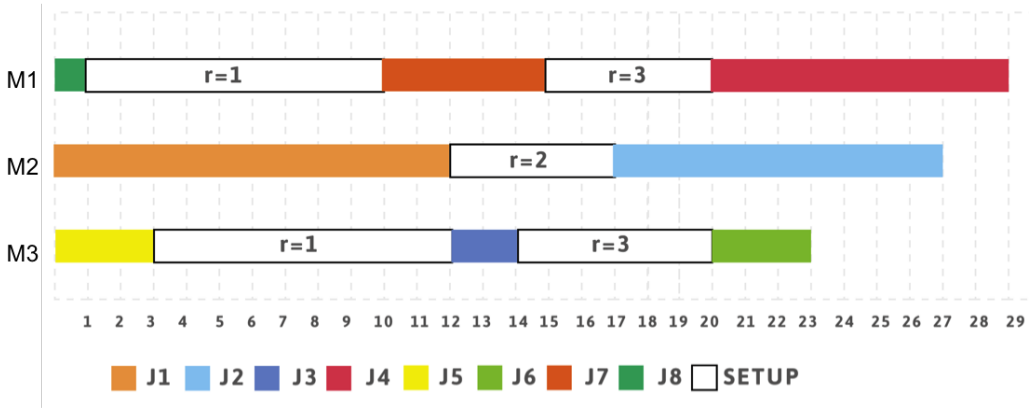

Figura 10: Solución inicial no factible.

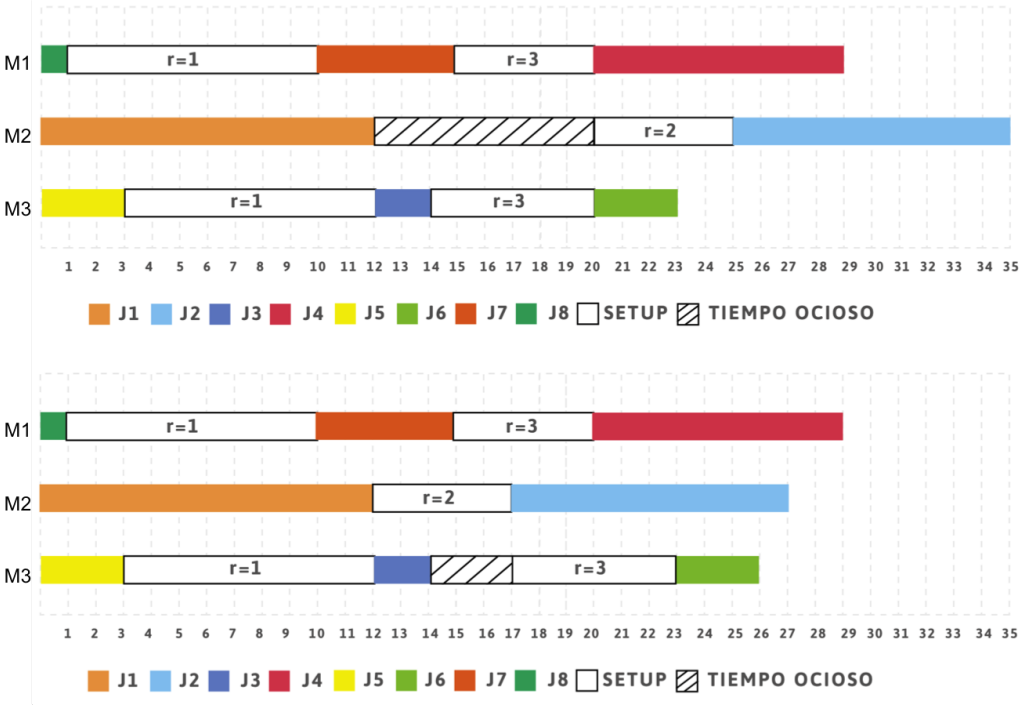

Figura 11: Primer movimiento de reparación.

La Figura 11 muestra los dos movimientos que realizan para saber que ajuste posponer. En los gráficos se observa que, si se pospone el inicio del ajuste entre los trabajos 1 y 2 en la máquina 2, este no se iniciaría hasta el instante de tiempo 20 (que es cuando termina el ajuste de la máquina 3), lo que haría que C_2 sea 35. Por otro lado, el movimiento en la máquina 3, aplazaría el inicio del ajuste entre los trabajos 3 y 6 en esta máquina hasta el instante de tiempo 17, teniendo C_3 igual a 26, por lo que este es el mejor movimiento y es el que se guarda (parte inferior de la Figura 11). Con esta nueva solución, se repite el proceso de evaluación de recursos desde el instante de tiempo donde se había encontrado el problema. Sin embargo, en la nueva solución se puede ver que al volver a evaluar los recursos, se incumple la restricción de recursos en el instante de tiempo 16, pero ahora las máquinas que se solapan son las maquinas 1 y 2. Se debe repetir el proceso de reparación como se muestra a continuación.

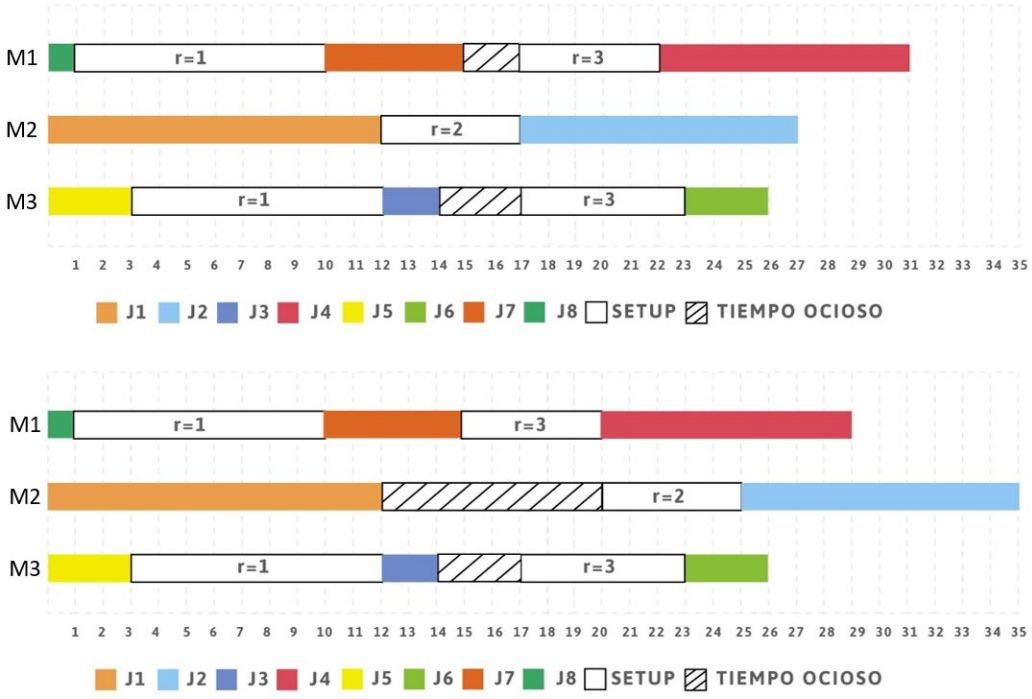

Figura 12: Segundo movimiento de reparación.

La Figura 12 muestra los movimientos de los ajustes de las máquinas 1 y 2 respectivamente. Al igual que en el movimiento anterior, se prueba aplazando el inicio de los ajustes que se solapan y se guarda el movimiento de la máquina que después de éste, tenga menor C_i. En este caso se observa que si se mueve el ajuste de la máquina 1, C_1 es 31, mientras que si se mueve el ajuste de la máquina 2, C_2 es 35, por lo que el ajuste que se aplaza es el ajuste entre los trabajos 7 y 4 en la máquina 1 que se ilustra en la parte superior de la Figura 12. Al igual que antes, se toma esta nueva solución y se reanuda la evaluación de recursos desde el instante de tiempo donde se había encontrado el incumplimiento de la restricción y se observa que en el instante de tiempo 18 vuelve a incumplirse la restricción de recursos, por lo que debe repetirse el aplazamiento de ajustes para reparar la solución como se muestra a continuación.

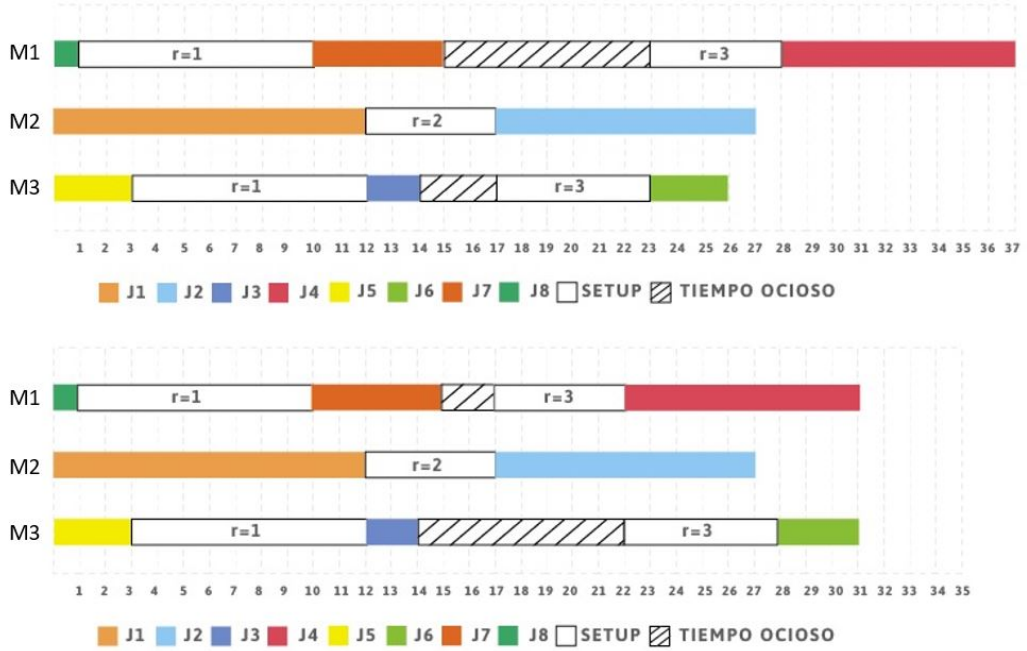

Figura 13: Tercer movimiento de reparación. Final algoritmo, solución factible.

En este caso los ajustes que se solapan son los realizados por las máquinas 1 y 3, por lo que se prueba aplazando el inicio de los ajustes en estas máquinas. En la primera gráfica de la Figura 13 se aplaza el inicio del ajuste entre los trabajos 7 y 4 de la máquina 1 hasta el instante de tiempo 24, obteniendo un C_1 de 37. Por otro lado, en la gráfica inferior Figura 13 se aplaza el inicio del ajuste entre los trabajos 3 y 6 en la máquina 3 hasta el instante de tiempo 23, obteniendo que C_e es 31, que sería el mejor movimiento en este caso. Una vez realizado este movimiento, se repite el proceso de evaluación de recursos y se observa que en todos los instantes de tiempo se cumple con la restricción, por lo que esta es una solución factible y finaliza el algoritmo con un $C_{\text{máx}}$ de 31.

5.3. Aleatorización del algoritmo

Con la finalidad de obtener mayor variedad de soluciones, se aleatoriza la parte constructiva del algoritmo. En lugar de elegir siempre el mejor candidato según las reglas de asignación de las dos partes del método constructivo explicadas en la Sección 5.1, se elige aleatoriamente uno de una lista con los $G \in \mathbb{Z}^+$ mejores candidatos, teniendo así la fase constructiva de un algoritmo GRASP. Una vez aleatorizado el algoritmo, para obtener varias soluciones a cada ejemplo, se ejecutará el algoritmo un número de iteraciones (que llamaremos $it_{\text{máx}}$) y se guardará la mejor solución encontrada entre esas iteraciones. Es importante notar que si $G = 1$, obtenemos el algoritmo sin aleatorizar (GREEDY).

5.3.1. Pseudocódigo de la aleatorización

A continuación se muestra el pseudocódigo del algoritmo aleatorizado:

$it = 0$;
while $it < it_{max}$ **do**
    Fase constructiva con aleatoriazación
    Fase de reparación
    $it = it + +$;
end

Algoritmo 4: Pseudocódigo general del algoritmo aleatorizado.

6. Experimentos computacionales

En esta sección se presentan los resultados obtenidos por el algoritmo propuesto al probarlo en instancias de diferente magnitud. Primero se explicará la forma en la que se generaron las instancias de prueba, y después se presentarán los resultados comparando los resultados del algoritmo propuesto sin aleatorizar y aleatorizado, con los resultados obtenidos por el $MILP$.

6.1. Generación de instancias

Para este trabajo, se cuenta con dos grupos de instancias de prueba, un grupo de instancias pequeñas, las cuales tienen 8 trabajos, y otro grupo de instancias medianas que tienen 32 trabajos. Ambos grupos de instancias tienen dos factores que varían, el número de máquinas (entre 2, 3, 4 y 5 máquinas en el caso de las instancias pequeñas, y entre 3, 4, 5 y 6 en las instancias medianas), y la forma forma de asignar los tiempos de ajuste entre trabajos (s_{ijk}), que se calcula de forma aleatoria mediante una distribución uniforme discreta entre 1 y 9, entre 1 y 49, entre 1 y 99, y entre 1 y 124. Estas variaciones nos dan 16 casos para cada grupo de instancias. Los tiempos de proceso p_{ij} se generaron mediante una distribución uniforme continua entre 1 y 100. Se generó una instancia por cada combinación de los factores a variar, teniendo así 16 instancias pequeñas y 16 medianas. Para cada instancia, el número máximo de recursos ($R_{\text{máx}}$) fue elegido aleatoriamente entre 3 y 4, mientras que el número de recursos requeridos para los ajustes (r_{ijk}) fue elegido mediante una distribución uniforme discreta entre 1 y $R_{\text{máx}}$. Por último, el límite superior $t_{\text{máx}}$ para el modelo de de programación matemática se ha generado como $\text{mín}_i \sum_{j=0}^{n-1} p_{ij} + s_{i,j,j+1}$, que es el mínimo makespan obtenido al asignar todos los trabajos en el orden marcado por su numeración a la misma máquina. En resumen tenemos:

Instancias pequeñas

- $n = 8$
- $m \in \{2, 3, 4, 5\}$

- $s_{ijk} \in \{U(1,9), U(1,49), U(1,99), U(1,124)\}$
- $R_{\text{máx}} = U(3,4)$
- $r_{ijk} = U(1, R_{\text{máx}})$
- $t_{\text{máx}} = \text{mín}_i \sum_{j=0}^{n-1} p_{ij} + s_{i,j,j+1}$

Instancias medianas

- $n = 32$
- $m \in \{3,4,5,6\}$
- $s_{ijk} \in \{U(1,9), U(1,49), U(1,99), U(1,124)\}$
- $R_{\text{máx}} = U(3,4)$
- $r_{ijk} = U(1, R_{\text{máx}})$
- $t_{\text{máx}} = \text{mín}_i \sum_{j=0}^{n-1} p_{ij} + s_{i,j,j+1}$

6.2. Lista de candidatos restringida

En el algoritmo aleatorizado propuesto, el tamaño de la lista de candidatos restringida para las instancias pequeñas se calcula como $G = \frac{n_h}{4}$, redondeando hacia arriba, donde n_h es el número de trabajos que quedan por asignar en la iteración h. Cabe mencionar que también se probó dividiendo n_h por 2, obteniendo peores soluciones, ya que al dividir por 2, la lista es demasiado grande, haciendo que la solución final esté poco dirigida y sea demasiado aleatoria. Para las instancias medianas, la lista de candidatos restringida se calcula como $G = \frac{n_h}{8}$. En estas instancias, también se probó con diferentes tamaños de lista, en este caso dividiendo por 2 y por 4, obteniendo peores soluciones, ya que al igual que con las instancias pequeñas, la lista queda muy grande y las soluciones son muy aleatorias.

6.3. Resultados

Para las pruebas realizadas en este trabajo, el algoritmo propuesto fue programado en Java versión 1.8.0_111−b14 en un ordenador con sistema operativo macOs Sierra versión 10.12.5 con procesador Intel Core i5 de 2.4 GHz y 16 GBytes de memoria Ram. El $MILP$ fue

resuelto con CPLEX 12.7.0, en un ordenador con sistema operativo Windows 10 de 64 bits, con procesador AMD 2.60 GHz y 8 GBytes de memoria Ram.

En esta sección, para medir la calidad de las soluciones obtenidas por el algoritmo propuesto, se calcula el porcentaje de desviación relativa (RPD por sus siglas en inglés) entre el algoritmo propuesto y los resultados obtenidos con el modelo de programación matemática, que se calcula de la siguiente manera:

$$RPD = \frac{C_{\text{máx}}(Alg) - C_{\text{máx}}(MILP)}{C_{\text{máx}}(MILP)} \cdot 100, \tag{10}$$

donde $C_{\text{máx}}(Alg)$ es la solución obtenida por el algoritmo propuesto, y $C_{\text{máx}}(MILP)$ es la solución obtenida con el modelo de programación matemática.

6.3.1. Resultados del algoritmo en instancias pequeñas

En la Tabla 17 se muestran los resultados del algoritmo propuesto sin aleatorizar (GREEDY) y aleatorizado para las instancias pequeñas. Para asegurar mayor variedad de soluciones obtenidas por el algoritmo aleatorizado, se ejecuta el algoritmo 1000 iteraciones guardando la mejor solución encontrada. La primera columna de la tabla muestra el nombre de la instancia, en la que el primer número es el número de trabajos a procesar, seguido del número de máquinas disponibles y la forma en la que se calcularon los tiempos de ajuste entre trabajos explicado previamente (009 equivale a $U(1, 9)$, 049 equivale a $U(1, 49)$, 099 equvale a $U(1, 99)$ y 124 equvale a $U(1, 124)$). Las siguientes columnas de la tabla se dividen en tres grupos que se detallan a continuación:

Resultados obtenidos por el $MILP$:

- $C_{\text{máx}}(MILP)$: Muestra la mejor solución encontrada por el $MILP$ tras una hora de cómputo.

- GAP: Muestra el GAP de la solución obtenida por el $MILP$.
- $C_{\text{máx}}(NR)$: Muestra la solución óptima obtenida por el $MILP$ para el problema sin tener en cuenta los recursos (UPMS), para ello se ha utilizado el $MILP$ en Avalos-Rosales et al. (2014) y que se encuentra detallado en el apéndice.

Resultados obtenidos por el algoritmo sin aleatorizar:

- $C_{\text{máx}}$: Muestra la solución obtenida por el algoritmo sin aleatorizar (GREEDY).
- t: Muestra el tiempo de cómputo en milisegundos del algoritmo GREEDY.
- RPD_{MILP}: Muestra el RPD entre los resultados obtenidos por el algoritmo GREEDY y la solución encontrada por el $MILP$. Sólo se puede calcular para las instancias en las que el $MILP$ encontró alguna solución factible.
- RPD_{NR}: Muestra el RPD entre los resultados obtenidos por el algoritmo GREEDY y la solución óptima obtenida por el $MILP$ para el problema sin tener en cuenta los recursos.

Resultados obtenidos por el algoritmo aleatorizado:

- $C_{\text{máx}}$: Muestra la mejor solución obtenida por el algoritmo aleatorizado después de las 1000 iteraciones.
- t: Muestra el tiempo de cómputo en milisegundos para ejecutar las 1000 iteraciones.
- BI: Muestra la iteración en la que el algoritmo encontró la mejor solución.
- RPD_{MILP}: Muestra el RPD entre los resultados obtenidos por el algoritmo aleatorizado y la solución encontrada por el $MILP$. Al igual que con el algoritmo sin aleatorizar, sólo se puede calcular para las instancias en las que el $MILP$ encontró alguna solución factible.
- RPD_{NR}: Muestra el RPD entre los resultados obtenidos por el algoritmo aleatorizado y la solución óptima obtenida por el

$MILP$ para el problema sin tener en cuenta los recursos.

Por último, la columna llamada RPD muestra el RPD entre el algoritmo sin aleatorizar y el algoritmo aleatorizado.

La Tabla 17 muestra los resultados obtenidos en los experimentos computacionales realizados. La fila llamada Promedio* representa los promedios teniendo en cuenta sólo las instancias en las que el $MILP$ encontró alguna solución factible, mientras que la fila llamada Promedio, tiene en cuenta todas las instancias. En los resultados presentados en la tabla, se puede ver que, en promedio, las soluciones encontradas por el algoritmo sin aleatorizar (GREEDY), con un tiempo promedio de cómputo de 3 milisegundos, son un 9.61 % peores que las encontradas por el modelo $MILP$. Sin embargo, es importante decir que, de las 16 instancias, después de una hora de cálculo, el $MILP$ sólo encontró 2 soluciones óptimas y no pudo encontrar ninguna solución factible para 3 de las 16 instancias. La comparación de los resultados entre este modelo y los algoritmos propuestos sólo se está haciendo para las instancias en las que el $MILP$ encontró alguna solución. Adicional a esto, es interesante ver que a pesar de ser peores en promedio las soluciones del algoritmo GREEDY, para algunas instancias encuentra mejores soluciones, y el makespan promedio de las instancias (calculado sólo con las instancias en las que el $MILP$ encontró alguna solución), es menor con respecto al makespan promedio del $MILP$. Con respecto al modelo sin recursos, las soluciones encontradas por el algoritmo GREEDY son un 62.46 % peores.

Por otro lado es interesante ver que al aplicar la aleatorización del algoritmo, se obtienen mejores soluciones en todas las instancias excepto en una, en la que casualmente, la solución encontrada por el algoritmo GREEDY es la misma que la mejor solución encontrada por el aleatorizado. Para comparar estos dos algoritmos, sí se tienen en cuenta las 16 instancias, en las que, en promedio, el algoritmo aleatorizado es un 21.62 % mejor que el algoritmo GREEDY. Es interesante ver que, al ser instancias pequeñas, a pesar de haber ejecutado el algoritmo 1000 iteraciones buscando mejores soluciones, en promedio, las

				GREEDY				Aleatorizado					
Instancia	$C_{\text{máx}}(MILP)$	GAP	$C_{\text{máx}}(NR)$	$C_{\text{máx}}$	t	RPD_{MILP}	RPD_{NR}	$C_{\text{máx}}$	t	BI	RPD_{MILP}	RPD_{NR}	RPD
8×2 −009	90	30.40 %	88	118	3	31.11	34.09	95	1588	18	5.56	7.95	-19.49
8×2 −049	256	80.16 %	167	207	3	**-19.14**	23.95	178	1901	3	**-30.47**	6.59	-14.01
8×2 −099	294	67.46 %	263	317	2	7.82	20.53	305	2446	6	3.74	15.97	-3.79
8×2 −124	260	75.00 %	211	305	2	17.31	44.55	283	3089	3	8.85	34.12	-7.21
8×3 −009	287	98.46 %	72	122	1	**-57.49**	69.44	85	2478	4	**-70.38**	18.06	-30.33
8×3 −049	186	71.81 %	103	119	2	**-36.02**	15.53	119	2234	6	**-36.02**	15.53	0.00
8×3 −099	488	99.39 %	126	243	2	**-50.20**	92.86	160	2796	3	**-67.21**	26.98	-34.16
8×3 −124	171	62.79 %	140	234	2	36.84	67.14	184	2377	3	7.60	31.43	-21.37
8×4 −009	62	0.00 %	62	118	2	90.32	90.32	75	2150	5	20.97	20.97	-36.44
8×4 −049	98	45.45 %	87	189	3	92.86	117.24	111	2631	31	13.27	27.59	-41.27
8×4 −099			74	133	6		79.73	129	2580	3		74.32	-3.01
8×4 −124			106	164	3		54.72	116	2699	4		9.43	-29.27
8×5 −009	43	0.00 %	43	69	13	60.47	60.47	49	2183	12	13.95	13.95	-28.99
8×5 −049	63	96.87 %	42	83	3	31.75	97.62	57	2768	8	**-9.52**	35.71	-31.33
8×5 −099			50	124	2		148.00	83	2675	52		66.00	-33.06
8×5 −124	508	100.00 %	55	98	1	**-80.71**	78.18	86	3018	14	**-83.07**	56.36	-12.24
Promedio*	215.85	63.68 %	112.23	170.92	3.00	**9.61**	62.46	137.46	2435.31	8.92	**-17.13**	23.94	**-21.59**
Promedio	215.85	63.68 %	105.56	165.19	3.13	**9.61**	68.40	132.19	2475.81	10.94	**-17.13**	28.81	**-21.62**

Tabla 17: Resultados para instancias pequeñas

mejores soluciones encontradas por el algoritmo para cada instancia, en las que el $MILP$ encontró alguna solución, aparecen en la iteración 8.92 con un tiempo de 2435.31 milisegundos para ejecutar las 1000 iteraciones. Con respecto a los resultados obtenidos por el modelo de programación matemática, el algoritmo aleatorizado, además de encontrar soluciones con menor makespan en promedio, mejora un 17.13 % las soluciones obtenidas por el $MILP$, y las soluciones encontradas son un 28.81 % peores que las soluciones óptimas del modelo sin tener en cuenta los recursos.

6.3.2. Resultados del algoritmo en instancias medianas

Debido a que, para este tamaño de instancias, no se cuenta con soluciones factibles obtenidas por el $MILP$, en este caso, compararemos los resultados de los algoritmos sin aleatorizar y aleatorizado entre sí, además, de compararlas con las soluciones del $MILP$ para el UPMS, que servirán como cota inferior del $C_{\text{máx}}$ del UPMSR. Los resultados obtenidos para las instancias medianas se muestran en la Tabla 18. Al igual que para las instancias pequeñas, la primera columna muestra el nombre de la instancia, en la que el primer número es el número de trabajos a procesar, seguido del número de máquinas disponibles y la forma en la que se calcularon los tiempos de ajuste entre trabajos. Después, se presentan las soluciones obtenidas por el $MILP$ para el UPMS, seguido de los resultados del algoritmo sin aleatorizar con el tiempo de cómputo en milisegundos, seguido de las mejores soluciones obtenidas por el algoritmo aleatorizado después de 1000 iteraciones con el tiempo de cómputo de estas iteraciones en milisegundos y de la iteración en la que el algoritmo encuentra la mejor solución. Por último, la columna RPD muestra el RPD entre el algoritmo aleatorizado y el algoritmo sin aleatorizar.

		GREEDY			Aleatorizado				
Instancias	$C_{\text{máx}}(NR)$	$C_{\text{máx}}$	t	RPD_{NR}	$C_{\text{máx}}$	t	BI	RPD_{NR}	RPD
$32 \times 3 - 009$	293	336	12	14.68	332	7804	635	13.31	-1.19
$32 \times 3 - 049$	321	500	6	55.76	401	7920	678	24.92	-19.80
$32 \times 3 - 099$	377	578	14	53.32	482	8406	790	27.85	-16.61
$32 \times 3 - 124$	432	704	7	62.96	589	10095	274	36.34	-16.34
$32 \times 4 - 009$	190	285	9	50.00	227	8030	946	19.47	-20.35
$32 \times 4 - 049$	295	464	11	57.29	371	9958	160	25.76	-20.04
$32 \times 4 - 099$	242	506	11	109.09	421	9648	329	73.97	-16.80
$32 \times 4 - 124$	269	590	6	119.33	484	9773	19	79.93	-17.97
$32 \times 5 - 009$	119	202	5	69.75	168	8713	229	41.18	-16.83
$32 \times 5 - 049$	127	321	8	152.76	261	9648	984	105.51	-18.69
$32 \times 5 - 099$	180	381	4	111.67	291	9350	422	61.67	-23.62
$32 \times 5 - 124$	209	645	14	208.61	452	9847	155	116.27	-29.92
$32 \times 6 - 009$	95	181	5	90.53	151	11132	693	58.95	-16.57
$32 \times 6 - 049$	118	297	4	151.69	219	10292	243	85.59	-26.26
$32 \times 6 - 099$	124	401	8	223.39	278	12259	149	124.19	-30.67
$32 \times 6 - 124$	147	462	9	214.29	245	10286	707	66.67	-46.97
Promedio	221.13	428.31	8.31	109.07	335.75	9572.56	463.31	60.10	**-21.17**

Tabla 18: Resultados para instancias medianas

En la Tabla 18 se puede observar que, comparando los dos algoritmos, el algoritmo aleatorizado, presenta resultados 21,17 % mejores que los del algoritmo sin aleatorizar y que en promedio, tarda 9,5 segundos en ejecutar las 1000 iteraciones del algoritmo. También es importante anotar que las mejores soluciones encontradas por el algoritmo aleatorio, se encuentran en promedio en la iteración 463. Al no tener soluciones factibles con las que comparar los resultados obtenidos con el algoritmo propuesto, es difícil saber si son buenas o malas las soluciones encontradas, sin embargo, se pueden comparar con una cota inferior que serían las soluciones óptimas obtenidas por el $MILP$ para el problema sin recursos. Al hacer esta comparación, observamos que, en promedio, el algoritmo propuesto encuentra soluciones 60,10 % peores, que no es mucho si se tiene en cuenta el tiempo de cómputo, y que estamos comparando con una cota inferior de un problema con menos restricciones y que en las instancias pequeñas la solución del $MILP$ UPMS es, aproximadamente, la mitad que las soluciones del $MILP$ UPMSR.

7. Conclusiones y trabajo futuro

En esta sección se comentan las conclusiones generales obtenidas a lo largo del desarrollo del trabajo, así como las líneas de trabajo futuras que quedan abiertas.

7.1. Conclusiones

En este trabajo se ha tratado el problema de máquinas paralelas no relacionadas con tiempos de ajuste dependientes de la secuencia y recursos escasos asignados a los tiempos de ajuste (UPMSR) con el objetivo de minimizar el makespan. Se ha expuesto la necesidad y la importancia de buscar métodos heurísticos que encuentren buenas soluciones a este problema y se ha presentado una propuesta de algoritmo heurístico.

El primer algoritmo propuesto, sin aleatorizar, para instancias pequeñas, donde el $MILP$ puede encontrar soluciones, encuentra soluciones un poco peores a las encontradas por el $MILP$, sin embargo, las encuentra con un tiempo de cómputo promedio de 3 milisegundos, por lo que ya es interesante, ya que las soluciones del $MILP$ son obtenidas después de una hora de cómputo, o incluso más. Sin embargo, al aplicar la aleatorización en el proceso constructivo del algoritmo, y ejecutarlo varias veces, se obtienen soluciones mejores que las obtenidas con el algoritmo sin aleatorizar y mejores que las encontradas por el $MILP$ sin requerir mucho tiempo de cómputo, lo que hace de gran interés aplicar esta aleatorización al algoritmo.

Este comportamiento se observa igualmente en las instancias medianas, donde no se pueden comparar los resultados con los obtenidos por el $MILP$, pero comparando los algoritmos sin aleatorizar y aleatorizado, el segundo encuentra soluciones mucho mejores después de ejecutar el algoritmo durante varias iteraciones con un tiempo de cómputo pequeño.

Esto muestra la importancia de la aleatorización en el proceso constructivo del algoritmo, ya que se le da mayor variedad de opciones para

construir la solución y al no ser muy costoso computacionalmente, se puede implementar sin problemas para este problema.

7.2. Trabajo futuro

El problema tratado en este trabajo, al ser un problema que no ha sido muy estudiado, podría continuar trabajándose de diversas maneras. A continuación se proponen algunos trabajos futuros para este problema:

- Aleatorizar la fase de reparación del algoritmo propuesto en este trabajo.
- Agregar un procedimiento de búsqueda local al algoritmo propuesto para intentar mejorar las soluciones obtenidas.
- Plantear variaciones del problema, por ejemplo, que los tiempos de ajuste o de proceso de tareas no tengan que ser seguidos, sino que se puedan interrumpir en algún momento para continuar con ellos después, o agregando recursos a los tiempos de proceso.
- Escribir un artículo sobre el problema UPMSR tratado en este trabajo.

Estas opciones de trabajo futuro pueden ser de gran interés para continuar con una futura tesis doctoral.

Referencias

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

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

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

E. G. Coffman, J., Garey, M. R., Johnson, D. S., 1978. An application of bin-packing to multiprocessor scheduling. SIAM Journal on Computing 7 (1), 1–7.

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

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

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

Garey, M. R., Johnson, D. S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA.

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

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

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 (5), 777–794.

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

Villa, F., Vallada, E., Fanjul-Peyró, L., February 2017. Heuristic algorithms for the unrelated parallel machines scheduling problem with scarce additional resources. Tech. rep., Universitat Politècnica de València.

Apéndice

El MILP más eficiente para el UPMS es el que encontramos en Avalos-Rosales et al. (2014). La formulación usa las variables X, Y, C, donde C es el tiempo de completación de la máquina i, y X y Y son las definidas en la Sección 4 de esta memoria. A continuación, veremos que la variable Y puede se relajada. El $MILP$ presentado en Avalos-Rosales et al. (2014) consiste en:

$$\text{mín } C_{\text{máx}} \tag{11}$$

$$\text{s.t.:} \sum_{j \in N_0, k \in N, k \neq j} s_{ijk} X_{ijk} + \sum_{j \in N} p_{ij} Y_{ij} \leq C_{\text{máx}},\ i \in M. \tag{12}$$

$$\sum_{k \in N} X_{i0k} \leq 1,\ i \in M \tag{13}$$

$$\sum_{i \in M} Y_{ij} = 1,\ j \in N \tag{14}$$

$$Y_{ij} = \sum_{k \in N_0, j \neq k} X_{ijk},\ i \in M, j \in N \tag{15}$$

$$Y_{ik} = \sum_{j \in N_0, j \neq k} X_{ijk},\ i \in M, k \in N \tag{16}$$

$$C_k - C_j + V(1 - X_{ijk}) \geq s_{ijk} + p_{ik},\ j \in N_0, k \in N, j \neq k, i \in M \tag{17}$$

$$C_0 = 0 \tag{18}$$

$$C_j \leq C_{\text{máx}} \tag{19}$$

La restricción (12) define el makespan (nótese que la parte izquierda de esas restricciones define que la parte del tiempo que la máquina correspondiente está ocupada). La restricción (13) asegura que como mucho, un trabajo está programado de primero en cada máquina. La restricción (14) asegura que cada trabajo va a ser procesado por una, y sólo una máquina. La restricción (15) asegura que todos los trabajos tienen sólo un sucesor (posiblemente el trabajo ficticio) para la máquina en la que son procesados. Análogamente, la restricción (16) asegura que todos los trabajos tienen un predecesor para la máquina en la que son procesados. La restricción (17) provee un orden de procesamiento correcto y rompe los ciclos. Se impone que, si k es el sucesor de j

para la máquina i, entonces k debe ser completado al menos $s_{ijk} + p_{ik}$ unidades de tiempo después de que j sea completado. La restricción (18) asegura que el tiempo de proceso del trabajo ficticio es cero. Las restricciones (19) son cortes factibles que se demostraron ser eficientes en Avalos-Rosales et al. (2014). El lector debe notar que la estructura del problema permite relajar las variables Y_{ij}, y, además, son definidas como positivas. En Avalos-Rosales et al. (2014), los autores explican que este modelo MILP resuelve eficientemente algunos casos de hasta 60 trabajos.