UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Departamento de sistemas informáticos y computación
Universidad Politécnica de Valéncia

Desarrollo de técnicas de búsqueda metaheurísticas en problemas de scheduling multiobjetivo

Trabajo fin de máster

MÁSTER UNIVERSITARIO EN INTELIGENCIA ARTIFICIAL, RECONOCIMIENTO DE FORMAS E IMAGEN DIGITAL

Autor: Sergio Ferrer Sánchez

Director: Miguel Ángel Salido Gregorio

Codirectores: Federico Barber Sanchís, Adriana S. Giret Boggino

Curso: 2016-2017

A Federico y Miguel, por la oportunidad que me brindan y la ayuda constante que recibo por su parte.

A Giancarlo, por su paciencia y motivación continua.

A Sergio, sin tu apoyo incondicional, mi trabajo no saldría adelante.

Y sobre todo, a mis padres. Gracias por ser un ejemplo de esfuerzo, superación y honradez. Cualquiera de mis humildes logros es vuestro.

Resumen

Los problemas de scheduling en Inteligencia Artificial, centrados en la asignación de recursos o tiempo a las acciones de un plan, se aplican en diversos entornos reales y su principal objetivo reside en encontrar soluciones optimizadas mediante el uso de técnicas de búsqueda centralizadas, donde, por lo general, se pretende optimizar la duración total del scheduling (makespan). Sin embargo, la realidad dista mucho de este planteamiento, donde muchos problemas de scheduling ocurren en entornos distribuidos y dinámicos. Esta naturaleza distribuida y dinámica de los problemas de scheduling, junto con los nuevos objetivos a optimizar, como la eficiencia energética, hacen pertinente gestionarlos de forma eficiente para mejorar los tiempos de cómputo a la vez que el sistema trata de adaptarse a las nuevas restricciones del problema. Por otro lado, cada recurso disponible suele tener un consumo energético asociado y variable en función del tiempo durante el que se usa o de la potencia de trabajo asignada. En un contexto en el que requieren procesos cada vez más sostenibles, surge la necesidad de obtener soluciones energéticamente eficientes. Esto representa una línea novedosa en la resolución de problemas de scheduling llamada "Green Scheduling".

En este proyecto presentamos un modelo multiagente para la resolución de Green Scheduling offline en sistemas de fabricación de intenso consumo energético. En el modelo propuesto, los agentes, representando cada uno las distintas máquinas del entorno, deben colaborar para conseguir un acuerdo con la finalidad de obtener una solución multiobjetivo basada en minimizar 3 factores: el retraso total de los trabajos, el tiempo total de preparación (o configuración) entre tareas y el consumo total de energía que realiza la maquinaria.

Posteriormente, a este sistema offline le integramos un segundo sistema online que monitoriza la ejecución del plan a la espera de que ocurran incidencias. En caso de que éstas ocurran, el módulo online realiza un rescheduling en tiempo real de la máquina afectada por la incidencia, redistribuyendo las tareas que tenía asignadas en caso de ser posible y tratando de minimizar el impacto que tiene la incidencia sobre los costes globales del entorno completo de fabricación.

Los sistemas desarrollados han sido evaluados sobre diversos casos de prueba y los resultandos obtenidos muestran la utilidad de las propuestas realizadas.

Palabras clave: scheduling, green scheduling, multiagente, control de producción

Abstract

In Artificial Intelligence field, scheduling problems are applied to multiple real environments and its main objective is to find optimized solutions by using centralized and static search techniques. However, the reality is far from this approach, where a lot of scheduling problems take place in distributed and dynamic environments. This dynamic and distributed nature of scheduling problems, along with new objectives to optimize such as energy efficiency, force us to manage them in a efficient way in order to improve computing time while it adapts to new constraints of the problem. Each available resource usually has and associated energy consumption and it is variable depending on its use time or the assigned power work. Given a context in which more and more efficient processes are needed, we need to obtain energy efficient solutions. This represents a new view in the way that we approach scheduling problems called "Green Scheduling".

In this project we present a multi-agent model as a tool to solve Green Scheduling offline problems in production system with high energy consumption. In the proposed model, agents collaborate with each other in order to achieve an agreement in which we optimize a multiobjective function based on minimizing 3 factors: the total tardiness of the jobs, the total setup time and the total energy consumption that machines need to perform all the assigned jobs.

Later, we add an online system integrating it in the offline system previously mentioned. This online system monitors the execution of the plan scheduled offline, waiting until an incidence appears. When an incidence occurs, the online system carries out a rescheduling of the affected machine in real time, redistributing the jobs originally assigned to it, if possible, and trying to minimize the impact of the incidence on the global costs of the whole production context.

The developed systems have been evaluated on several test cases and the obtained results show the utility of the proposals.

Keywords : scheduling, green scheduling, multiagent, production control

Tabla de contenidos

Contenido

1. Introducción 11

1.1 Descripción del problema 12

1.2 Definición formal del problema 13

1.3 Tamaño del espacio de búsqueda 16

1.4 Objetivos 17

2. Descripción del sistema offline propuesto 18

3. Propuesta de algoritmo genético para la resolución de problemas de scheduling multiobjetivo. 21

3.1 Representación del individuo 21

3.2 Fitness 22

3.3 Algoritmos de inicialización propuestos 22

3.4 Estudio del tamaño de la población 24

3.5 Selección , cruce y mutación 26

3.6 Sustitución 27

4. Estructura multiagente propuesta 28

4.1 Políticas de selección de trabajo 31

4.2 Worst Local Job (WLJ) 31

4.3 Best Local Job (BLJ) 33

4.4 Política de selección de máquina 33

4.5 Worst Available Machine (WAM) 34

5. Resultados del entorno en scheduling multiobjetivo 37

5.1 WLJ vs AG 37

5.2 BLJ vs AG 40

5.3 WLJ vs BLJ 43

6. Rescheduling online para problemas de scheduling multiobjetivo. 46

6.1 Trabajo relacionado 46
6.2 Introducción 48
6.3 Estructura del entorno de rescheduling 50
6.4 Definición formal de incidencia 51
6.5 Funcionamiento del sistema 52
6.6 Resultados del sistema online 54
7. Conclusiones 61
Bibliografía 63

Índice de tablas

Tabla 1: Nomenclatura y definiciones 14
Tabla 2: Algoritmo de inicialización aleatoria 23
Tabla 3: Algoritmo de inicialización voraz 24
Tabla 4: Inicialización aleatoria vs inicialización voraz 24
Tabla 5: Estudio del tamaño de la población 25
Tabla 6: Algoritmo de tratamiento de incidencias 53
Tabla 7: Ejemplo. Posibles máquinas asignables a cada trabajo 55
Tabla 8: Tiempo de respuesta del solver 60

Índice de figuras

Figura 1: Ejemplo de posibles asignaciones 16
Figura 2: Estructura completa del sistema 20
Figura 3: Estudio del tamaño de la población 25
Figura 4: Operador de cruce en un punto 26
Figura 5: Operador de cruce en dos puntos 27
Figura 6: Comparativa WLJ vs AG, instancia muy grande 38
Figura 7: Comparativa WLJ vs AG, instancia grande 38
Figura 8: Comparativa WLJ vs AG, instancia mediana 39
Figura 9: Comparativa WLJ vs AG, instancia pequeña 39
Figura 10: Comparativa BLJ vs AG, instancia muy grande 40
Figura 11: Comparativa BLJ vs AG, instancia grande 41
Figura 12: Comparativa BLJ vs AG, instancia mediana 41
Figura 13: Comparativa BLJ vs AG, instancia pequeña 42
Figura 14: Comparativa BLJ vs WLJ, instancia muy grande 43
Figura 15: Comparativa BLJ vs WLJ, instancia grande 44
Figura 16: Comparativa BLJ vs WLJ, instancia mediana 44
Figura 17: Comparativa BLJ vs WLJ, instancia pequeña 45
Figura 18: Comparativa nº de reschedulings 46
Figura 19: Sistemas offline y online integrados 50
Figura 20: Esquema ejemplo de asignación 52
Figura 21: Ejemplo. Scheduling original que sufre una incidencia durante su ejecución 56

Figura 22: Ejemplo. Técnica de resolución propuesta como baseline: propagación 56
Figura 23: Ejemplo. Resolución de la incidencia mediante el solver propuesto 57
Figura 24: Evaluación del solver, instancia muy grande 58
Figura 25: Evaluación del solver, instancia grande 58
Figura 26: Evaluación del solver, instancia mediana 59
Figura 27: Evaluación del solver: instancia pequeña 59

1. Introducción

En el área de Inteligencia Artificial (IA) existe un notable interés por encontrar métodos automáticos capaces de dar respuesta a problemas combinatorios (incluyendo soluciones bioinspiradas, metaheurísticas, basadas en restricciones, etc.). Dentro de este campo, los problemas de scheduling representan uno de los mayores retos y tienen asociada una gran relevancia práctica al estar relacionados con el consumo de recursos (asignación de recursos, temporización de acciones, planificación de turnos, etc.).

Tradicionalmente los problemas de scheduling en procesos industriales y logísticos se centran en la minimización de los tiempos de proceso con el objetivo de optimizar sus beneficios. Sin embargo, cada vez más se necesita el uso de técnicas multiobjetivo que combinen los criterios habituales con las necesidades actuales de sostenibilidad energética. Actualmente, tanto los procesos industriales como logísticos incluyen entre sus objetivos la reducción de emisiones de efecto invernadero. La energía es un recurso imprescindible que la humanidad necesita para su supervivencia y desarrollo. Prácticamente la mitad de la energía que se consume en el planeta la utiliza el sector industrial (Newman, Nassehi, Imani-Asrai, & Dhokia, 2012).

La mayoría de la investigación realizada en el contexto de los problemas de scheduling donde se gestiona un consumo energético, se centra en buscar soluciones optimizadas que satisfagan las restricciones estáticas del problema. Sin embargo, en problemas dinámicos, las restricciones, variables o dominios cambian a lo largo del tiempo, por lo que es imprescindible considerar otros planteamientos (Wallace, Grimes, & Freuder, 2009). Al respecto, las técnicas reactivas se centran en reusar la solución original para producir una nueva solución a partir de cambios de la solución original, mientras que las técnicas proactivas persiguen obtener soluciones capaces de absorber los cambios del problema o minimizar los cambios necesarios para obtener una solución válida (Verfaillie & Schiex, 1994).

En concreto, en este proyecto nos enfrentamos a un problema de scheduling multiobjetivo donde cada uno de los trabajos a realizar puede asignarse a un subconjunto del total de máquinas disponibles. Con el fin de resolver este problema se han desarrollado diversas aproximaciones multiagente para resolver problemas de scheduling. (Agnetis, Billaut, Gawiejnowicz, Pacciarelli, & and Soukhal, Multiagent Scheduling: Models and Algorithms, 2014) presenta una solución multiagente en el cual subconjuntos de trabajos que comparten los mismos recursos son evaluados con criterios diferentes para el scheduling en una o varias máquinas. En (Liu, Abdelrahman, & Ramaswamy, 2007) se propone un framework multiagente para el scheduling de trabajos de forma dinámica. Lo hace mediante la aplicación de solvers distribuidos en múltiples agentes para fijar trabajos en un entorno dinámico conociendo información mínima del estado global. Recientemente, estos autores han propuesto una generalización de un sistema basado en agentes donde cada agente implementa una combinación de metaheurística/búsqueda local para cooperar con los diferentes agentes a través del envío de mensajes asíncronos. Sin embargo, cada agente trabaja en el problema completo.

En este trabajo utilizaremos un enfoque multiagente donde, en nuestro caso, cada agente trabajara sobre una parte del problema y donde será necesario un acuerdo para construir la solución global. Posteriormente, añadiremos un módulo de rescheduling a las soluciones obtenidas para lograr generar una mayor adaptabilidad al entorno pudiendo gestionar posibles incidencias.

1.1 Descripción del problema

El problema consiste en el scheduling de un conjunto de trabajos en un conjunto de prensas de moldeo por inyección. Para cada trabajo existe un conjunto de prensas disponibles en las que se puede llevar a cabo. El tiempo de procesado y el consumo de energía dependen de la máquina y el trabajo. Ya que se requiere limpiar las prensas entre trabajo y trabajo, se tiene que considerar un tiempo de *setup* entre trabajos (configuración entre trabajos).

De forma general, el problema consiste en asignar un conjunto de tareas en un conjunto de máquinas (prensas de moldeo por inyección) y, dentro de

cada una de las máquinas, establecer el orden de cada una de las tareas asignadas.

Cada tarea tiene una fecha de liberación, la cual es la fecha más temprana en la que puede ser dispuesta en una máquina para llevarse a cabo, y una fecha de entrega. Los trabajos tienen distintas prioridades, controladas por unos pesos para calcular la penalización si estos son finalizados fuera de la fecha de entrega. Nótese que esto implica que un trabajo puede terminar después de su fecha prevista, asumiendo una penalización.

Cada trabajo tiene que ser procesado por una máquina del conjunto de alternativas existentes para ese trabajo. El tiempo de proceso y consumo de energía de los trabajos depende de la máquina seleccionada, ya que máquinas diferentes necesitan una cantidad de energía distinta para hacer la misma tarea. Para procesar una tarea, la máquina ha de ser preparada (debe ser limpiada, se debe haber cambiado el molde de inyección, etc.). Por ello, se considera un tiempo de *setup* (o configuración) entre dos trabajos consecutivos en la misma máquina. El tiempo de *setup* depende de la máquina y de la secuencia de trabajos.

1.2 Definición formal del problema

Una solución S al problema se define como una especificación de qué trabajo debe realizarse en qué máquina y del orden en el cual deben realizarse estos trabajos dentro de la máquina asignada. En la tabla 1 se muestra la nomenclatura a utilizar para la definición del problema, extraída de (Nicoló, Salido, Ferrer, Giret, & Barber, 2017).

A la hora de evaluar una solución, hay que tener en cuenta 3 factores, dado que la evaluación de la solución se hace de forma multiobjetivo. En concreto, los 3 factores que pretendemos minimizar, expresados como funciones objetivo, son:

-El retraso total de los trabajos en la solución s: TT(s)

$$TT(s) = \sum_{j \in J} W_j * t_j \qquad (1)$$

-La energía total consumida en la solución s: EN(s)

$$EN(s) = \sum_{j \in J} \sum_{k \in M_j} E_{jk} \sum_{i \neq j,\ i \in J_k} x_{ijk} \qquad (2)$$

-El tiempo total de configuración entre trabajos en la solución s: ST(s)

$$ST(s) = \sum_{k \in M} \sum_{i \in J_k} \sum_{i \neq j, j \in J_k} S_{ijk} x_{ijk} \qquad (3)$$

Variable	Definición
$J = \{1, 2 \dots, n\}$	El conjunto de todos los trabajos.
$M = \{1, 2, \dots, m\}$	El conjunto de todas las máquinas
M_j , $\forall j \in J$	Conjunto de máquinas que pueden ejecutar el trabajo j
J_k , $\forall k \in M$	Conjunto de trabajos que puede ejecutarse en la máquina k
D_j, $\forall j \in J$	Tiempo de finalización a partir del cual se desea que el trabajo esté terminado (Due date)
R_j, $\forall j \in J$	Tiempo en el que el trabajo se libera para su realización. Antes de R_j , el trabajo j no puede realizarse
W_j, $\forall j \in J$	Penalización que sufrirá el sistema por cada intervalo de tiempo que haya en el retraso de la tarea j.
P_{jk}, $\forall j \in J, \forall k \in M_j$	Tiempo de procesamiento de la tarea j en la máquina k
E_{jk}, $\forall j \in J, \forall k \in M_j$	Energía que consume el trabajo j en la máquina k
S_{ijk}, $\forall j \in J, \forall k \in M_j, i \neq j$	Tiempo de configuración necesario para preparar la máquina k entre las tareas i y j
s_j	Tiempo de inicio de la tarea j
c_j, $\forall j \in J$ $c_j = sj + Pjk$	Tiempo de finalización de la tarea j (Completion time)
t_j, $\forall j \in J$ $t_j = \max(0, c_j - D_j)$	Retraso con el que terminamos la tarea j
$x_{ijk} \in \{0,1\}$	Una variable binaria utilizada para indicar la secuenciación de tareas. Si su valor es 1, indica que el trabajo i precede de forma inmediata al trabajo j en la máquina k.

Tabla 1: Nomenclatura y definiciones

Por lo tanto, dada una solución s al problema, obtendremos su evaluación mediante una combinación de estos 3 factores, es decir:

$$\text{Puntuación} = F(s)$$

Donde F(s) es una función que combina las ecuaciones (1),(2) y (3) con el fin de encontrar una evaluación de s que tenga en cuenta los 3 factores que pretendemos minimizar. En concreto, F(s) se define como:

$$F(S) = \sum_{g=1}^{3} \pi_g \frac{f_g(s) - f_g^-}{f_g^+ - f_g^-} \qquad (4)$$

Donde:

- $f_g(s)$, $g \in \{1,2,3\}$, representa cada una de las 3 funciones objetivo que pretendemos minimizar: TT(s), EN(s) y ST(s), definidas en (1), (2) y (3).
- f_g^- representa el mejor valor (el mínimo) encontrado de forma experimental para la componente g-ésima cuando se optimiza de forma individual.
- f_g^+ representa el peor valor (el máximo) encontrado de forma experimental para la componente g-ésima cuando se optimiza de forma individual cualquiera de las otras 2 funciones objetivo
- π_g representa el peso de la componente g-ésima sobre el valor final de la función objetivo, sujeto a:

$$\sum_{g=1}^{3} \pi_g = 1$$

$$0 \leq \pi_g \leq 1, \forall g \in \{1,2,3\}$$

Los valores f_g^+ y f_g^- se han obtenido mediante estudio exhaustivo y analítico que se presenta en (Paolucci, Anghinolfi, & Tonelli, 2016)

Dadas estas definiciones, buscamos una solución s* tal que:

$$s^* = argmin_{s \in S} F(s) \qquad (5)$$

Donde S representa el espacio de todas las posibles s soluciones.

1.3 Tamaño del espacio de búsqueda

En este apartado expondremos brevemente el tamaño total del espacio de búsqueda que presenta este problema con la finalidad de dar una idea de la dificultad que presenta debido al alto espacio combinatorio que generan todas las posibles asignaciones de trabajos en cada una de las máquinas.

1.3.1 Combinatoria de las asignaciones

Primero analizaremos el número total de asignaciones existentes entre trabajos y máquinas, sin tener en cuenta el orden en el que se coloquen estos trabajos dentro de las máquinas. Es decir, primero abordaremos el siguiente problema: "Dado un conjunto de máquinas y un conjunto de trabajos, ¿de cuántas formas distintas pueden asignarse los trabajos a las máquinas?".

Para este análisis supondremos que todos los trabajos pueden ejecutarse en todas las máquinas, por lo que calcularemos el número total de combinaciones posibles, sin tener en cuenta restricciones de incompatibilidades entre máquinas y trabajos.

Supongamos que disponemos de un conjunto de N trabajos y M máquinas. Cada uno de los trabajos puede asignarse a cada una de las M máquinas y la asignación de un trabajo a una máquina no condiciona en ningún aspecto la asignación de otro trabajo a cualquier máquina, por lo tanto, el número total de combinaciones asciende a M^N. La figura 1 trata de mostrar de forma intuitiva algunas posibles combinaciones pare que se ilustre el número total de combinaciones existentes. Cada posible asignación es cada uno de los posibles caminos desde el trabajo T1 hasta el trabajo TN.

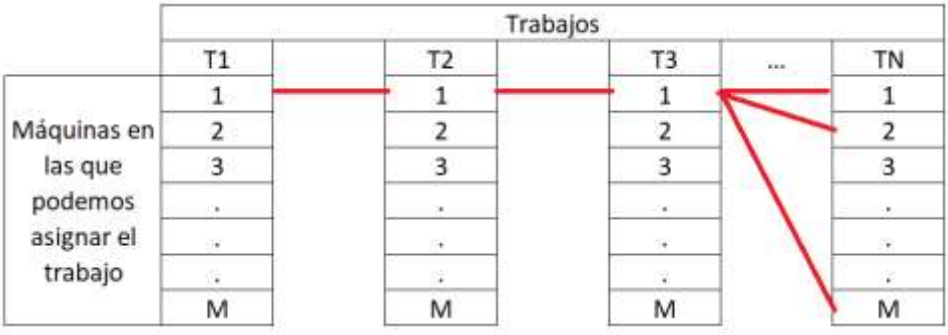

Figura 1: Ejemplo de posibles asignaciones

Tengamos en cuenta que solemos trabajar en contextos que tienen unas 20 máquinas y un conjunto de unos 250 trabajos, lo que genera un espacio total de combinaciones de 20^{250}, es decir, un espacio totalmente inexplorable por búsqueda exhaustiva.

1.3.2 Combinatoria en cada máquina

En este apartado analizaremos las posibles combinaciones que hay en cada una de las máquinas de forma independiente. Es decir, dado un conjunto de T trabajos asignados a una máquina concreta, ¿de cuántas formas se puede ordenar esos trabajos? Recuérdese que el orden es importante en nuestro problema, dado que el tiempo total de *setup* depende de la secuencia en la que se ordenen los trabajos, pudiendo influir en gran medida en la evaluación de la solución.

Dado que estas ordenaciones corresponden a posibles colocaciones de los trabajos sin posible repetición, hablamos de *permutaciones de T elementos*, que se calcula como factorial de T.

1.4 Objetivos

A continuación, listamos los objetivos que nos marcamos en la realización de este proyecto que pretenden ser una representación del estado al que se pretende llegar en la finalización de éste y tratando de ser un enfoque fiel y realista del entorno y la complejidad, tanto computacional como espacial, del problema con el que trabajamos. Dado que en este proyecto se desarrollan dos sistemas, uno offline y otro online, hemos dividido los objetivos en función del sistema al que están destinados:

Objetivos del sistema offline

1º - Lograr soluciones offline competentes al problema descrito en un tiempo de cómputo inferior a 30 minutos. Este tiempo se adapta a las necesidades del contexto y proporciona un buen margen de precaución para resolver cualquier inconveniente que pudiera surgir en el proceso.

2º - Generar heurísticas e implementar varias metaheurísticas de búsqueda que mejoren de forma considerable la solución que proporciona un

primer algoritmo genético ya implementado. Este objetivo se centra en mejorar el trabajo actual del que ya se dispone al inicio de este proyecto.

Objetivos del sistema online

1º - Lograr la integración completa del sistema online, desarrollado de forma íntegra en este proyecto, en el sistema offline.

2º - Conseguir que el sistema online sea capaz de realizar la gestión de incidencias en un tiempo de respuesta considerablemente bajo, del orden de segundos, para poder integrarlo en un contexto de control en tiempo real.

3º - Conseguir menos de un 5% de degradación de calidad de las soluciones al gestionar una incidencia. Este margen del 5% se establece como *baseline* de forma experimental, dado que es la degradación que aparece en las soluciones cuando las incidencias no se gestionan. Como objetivo, nos proponemos mejorar este comportamiento "por defecto" para lograr una mejoría en la gestión de incidencias.

2. Descripción del sistema offline propuesto

Para encontrar una solución al problema propuesto, se presenta un sistema multiagente con la idea de descentralizar el proceso de búsqueda de la solución, consiguiendo así escalar el poder computacional disponible para abordar el problema en cuestión.

Dada la alta carga y complejidad del problema que estamos manejando, esta descentralización genera un buen recurso extra que permite proporcionar soluciones más optimizadas y explorar un mayor espacio de búsqueda que, de otra manera, necesitaría varios órdenes de magnitud superiores, en cuanto a tiempo computacional se refiere, para ser explorado.

El problema podría ser descompuesto de forma óptima si cada uno de los trabajos pudiera ser asignado únicamente a una máquina, pues de esta forma, todos los trabajos asociados a una máquina podrían ser organizados de forma independiente a todos los demás, por lo que bastaría con realizar el scheduling de cada máquina de forma independiente. Sin embargo, en nuestro entorno, muchos trabajos pueden ser ejecutados en múltiples máquinas distintas (trabajos compartidos) y, evidentemente, dentro de cada una de ellas debemos buscarle la posición óptima.

Por lo tanto, el mayor problema y la mayor parte del esfuerzo computacional radica en encontrar cómo elegir qué máquina es asignada a cada trabajo para tratar de optimizar la función multiobjetivo que se ha descrito en el punto 1.2 (fórmula (4)), pretendiendo minimizar el uso de varios recursos de forma simultánea y ponderados por sus correspondientes pesos. La idea básica que hay tras el framework es ir reduciendo el espacio total de búsqueda, de forma iterativa, tratando que los trabajos compartidos tengan cada vez menos máquinas en las que puedan ser ejecutados hasta que los trabajos compartidos se transformen en "trabajos asignados" (aquellos que sólo pueden ejecutarse en una máquina).

Para conseguir esto, de forma iterativa iremos seleccionando un trabajo compartido de entre el conjunto formado por todos los trabajos compartidos, y al trabajo seleccionado le eliminaremos una de sus posibles máquinas, reduciendo así el espacio de búsqueda. Para operar de esta forma necesitaremos una política de selección de trabajo, que nos indicará cual de los trabajos compartidos tenemos que seleccionar, y una política de selección de máquina, que será el criterio a seguir para que, dado un trabajo compartido, podamos seleccionar qué máquina eliminar. En la sección 4, se presentan varias políticas de selección de trabajo y una de selección de máquina que se han implementado y comparado.

Para lograr este funcionamiento, los agentes de nuestro sistema (cada uno de ellos simulando una máquina distinta) colaborarán en el proceso hasta que lleguen a un acuerdo. Esta tarea se lleva a cabo partiendo de una solución obtenida a partir de un proceso metaheurístico (algoritmo genético) que se da en un paso anterior, ya que el proceso de acuerdo entre los agentes necesita una

solución inicial sobre la que un agente específico deberá razonar para operar cómo se ha descrito anteriormente. El objetivo es mejorar la solución que proporciona este algoritmo genético sin la utilización de nuestro framework.

Partiendo de esta idea, la figura 2 presenta el flujo de trabajo del sistema completo, donde se muestran los módulos "algoritmo genético" y "framework multiagente" que serán descritos en las siguientes secciones.

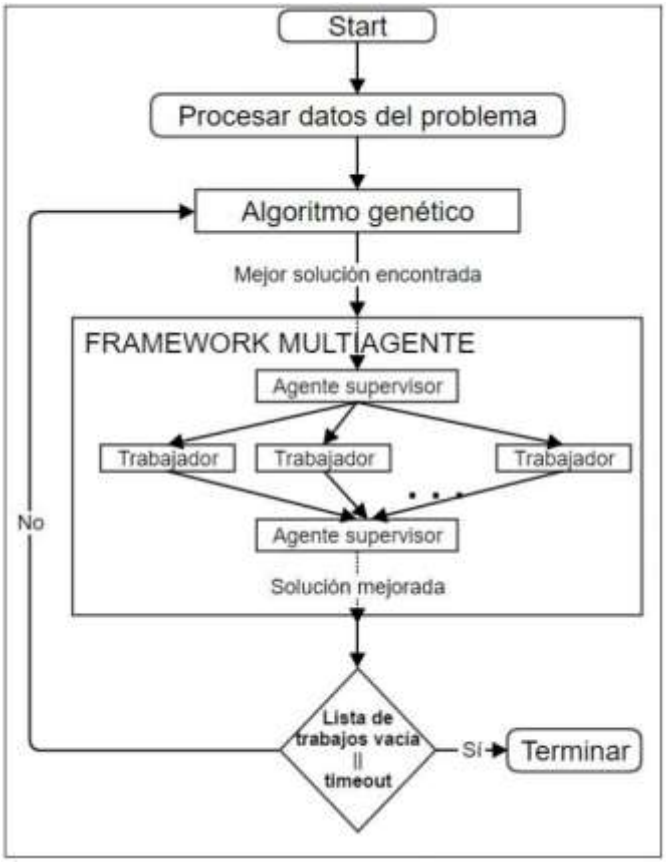

Figura 2: Estructura completa del sistema

3. Propuesta de algoritmo genético para la resolución de problemas de scheduling multiobjetivo.

En esta sección presentaremos el algoritmo genético (AG) que inicia el sistema. La idea principal es ejecutar este AG para encontrar soluciones al problema. En un momento determinado, el AG se detiene y recupera la mejor solución que haya encontrado. Esta solución se mejora por el framework multiagente situado inmediatamente después del AG (ver figura 2) y las conclusiones que haya extraído el framework multiagente durante su ejecución, son introducidas en toda la población del AG, que se vuelve a ejecutar de nuevo por un tiempo limitado hasta detenerse otra vez para ejecutar el framework multiagente. Este comportamiento iterativo es llevado a cabo hasta que se alcanza un *timeout* o ya no existen más trabajos compartidos.

3.1 Representación del individuo

Cada individuo se representa como una lista T de N tuplas de la forma:

$$T = [t_1 , t_2 , \dots , t_N]$$

donde:

- **N:** es el número total de trabajos que tiene el problema

A su vez, cada tupla t_i, $1 <= i <= N$, tiene la forma:

$$t_i = [n, m, M_n]$$

donde:

- **n**: es un número entero que representa el identificador de un trabajo
- **m**: es un número entero que representa el identificador de la máquina en la que se ha decidido colocar al trabajo n

- **M_n** : es el conjunto formado por los identificadores de todas las máquinas en las que puede ejecutarse el trabajo n.

Nótese que el tiempo en el que un trabajo n será procesado en la máquina m asignada, será determinado por la posición de la tupla t_i que contenga a n. Es decir, la primera tupla t_i que contenga el identificador m, estará colocando al trabajo n en la primera posición dentro de m, y de forma general, la q-ésima tupla con la máquina m asignada estará colocando su trabajo en la posición q dentro de m.

3.2 Fitness

El fitness de un individuo se calcula utilizando la expresión (4), por lo que se evalúa cada individuo de la misma forma que se evaluará la solución al problema. De esta forma, nos aseguramos que no existen discrepancias entre las prestaciones que ofrece un individuo y las prestaciones que ofrece la solución que éste representa al entorno real de aplicación.

3.3 Algoritmos de inicialización propuestos

A la hora de crear una población inicial se ha decidido estudiar 2 alternativas: generar individuos de forma aleatoria o generar individuos mediante un algoritmo voraz que genera soluciones subóptimas. A continuación se exponen los 2 algoritmos en pseudocódigo que se han utilizado para inicializar la población:

Algoritmo de inicialización aleatoria:

Entradas: lista T de trabajos, lista M de máquinas, tamaño N de la población, todos los parámetros del problema (ver tabla 1)
Salida: Una lista de N elementos, donde cada elemento es un individuo, es decir, una asignación para todos los trabajos de una máquina en la que ejecutarse (una solución)

```
Algoritmo:
1   población = new List()
2   while población.length < N:
3       individuo = new List(M) //Crea una lista de tamaño M
4       for each t in T: //recorremos todos los trabajos
5           t = new List(3) //creamos su representación
6           t[0] = t //identificador del trabajo
7           t[1] = randomInteger(M) //devuelve un N, 1<=N<=M. M
8                                   // es la máquina asignada
9           t[2] = Mt
10          individuo.append(t)
11      poblacion.append(individuo)
12   return población
```

Tabla 2: Algoritmo de inicialización aleatoria

Como se puede observar en la línea 7, la asignación de la máquina es completamente aleatoria.

Algoritmo de inicialización voraz:

Entradas: lista T de trabajos, lista M de máquinas, tamaño N de la población, todos los parámetros del problema (ver tabla 1)
Salida: Una lista de N elementos, donde cada elemento es un individuo, es decir, una asignación para todos los trabajos de una máquina en la que ejecutarse (una solución)

```
Algoritmo:
1   población = new List()
2   while población.length < N:
3       individuo = new List(M) //Crea una lista de tamaño M
4       for each t in T: //recorremos todos los trabajos
5           t = new List(3) //creamos su representación
6           t[0] = t //identificador del trabajo
```

```
7            t[1] = selectBesGreedyMachine(individuo,t)
8            t[2] = Mt
9            individuo.append(t)
10       poblacion.append(individuo)
11   return población
12
13   function selectBestGreedyMachine(individuo, t):
14      for maquina in individuo: //Recorremos las máquinas
15        individuo.append(t) //añadimos t a todas las máquinas
16      mejorMaquina = seleccionaMáquinaConMejorFitness(individuo)
17      return mejorMaquina
```

Tabla 3: Algoritmo de inicialización voraz

Como se puede observar, en este caso, en la línea 7, en lugar de asignar una máquina aleatoria, llamamos a una función que devuelve la "mejor" (la mejor según el algoritmo voraz presentado) máquina que asignar a ese trabajo. Esta función, que definimos en la línea 13, lo que hace es añadir el trabajo a todas las posibles máquinas y, simplemente, elegir la mejor de todas una vez añadido el trabajo. Por lo tanto, esta opción devuelve la mejor máquina en la que añadir el trabajo al final del *Schedule* que ya tiene asignado.

Se ha ejecutado el algoritmo genético con ambos algoritmos de inicialización, con el mismo *timeout*, y se ha repetido la ejecución 10 veces. A continuación, se muestran los resultados medios obtenidos:

	Algoritmo aleatorio	**Algoritmo voraz**
Fitness de la solución	0.39	0.27

Tabla 4: Inicialización aleatoria vs inicialización voraz

Recordemos que nuestro objetivo era minimizar la función objetivo, por lo que el algoritmo voraz ha proporcionado mejores resultados que la inicialización aleatoria.

3.4 Estudio del tamaño de la población

A la hora de elegir el tamaño inicial para la población hemos realizado un barrido entre distintos valores, para ambos algoritmos de inicialización. Cada

uno de los experimentos ha tenido el mismo *timeout* y se ha realizado 10 veces. A continuación, presentamos los resultados medios obtenidos. En negrita se expresa el mejor resultado de cada algoritmo y sombreado en verde el mejor resultado de todos.

Tamaño de la población	Inicialización aleatoria	Inicialización voraz
50	0.47	0.36
100	0.44	0.31
200	0.41	0.29
400	0.39	0.27
600	**0.33**	**0.26**
800	0.34	0.28
100	0.38	0.28
1400	0.34	0.33
1800	0.35	0.29
2000	0.35	0.3

Tabla 5: Estudio del tamaño de la población

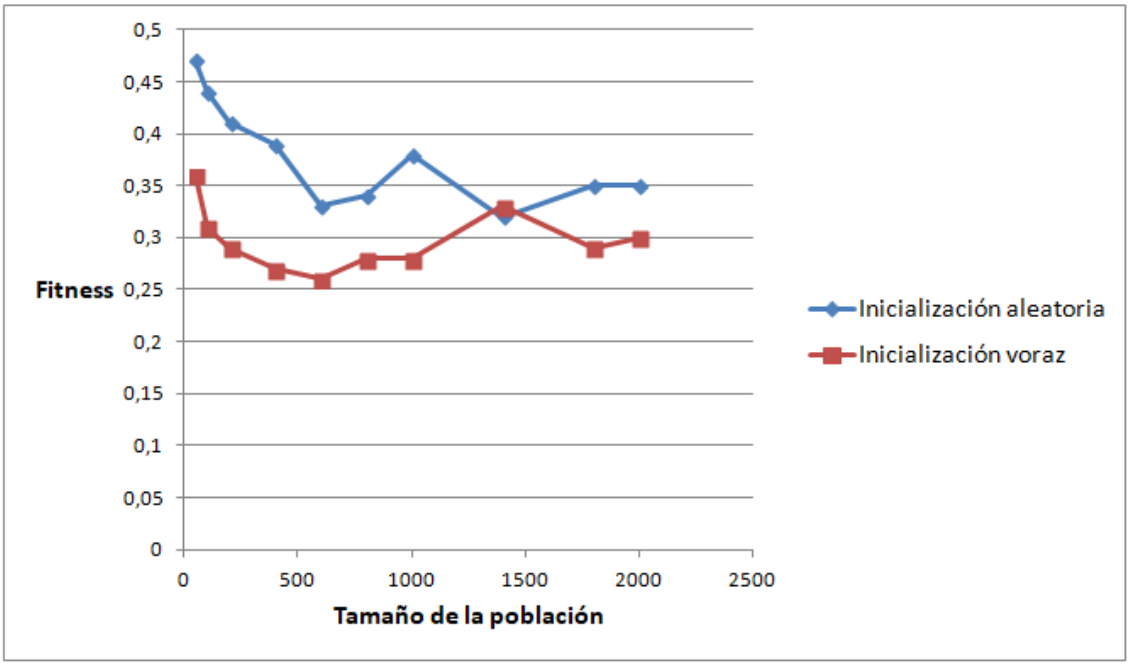

Figura 3: Estudio del tamaño de la población

Como se puede observar, de forma generalizada, la inicialización voraz proporciona mejores resultados, independientemente del tamaño de la población. Además, en ambos algoritmos se ha conseguido el mejor resultado cuando la población tiene tamaño 600. Por esto, el algoritmo genético que integraremos en nuestro framework tendrá tamaño de población 600 y se inicializará de forma voraz.

3.5 Selección, cruce y mutación

Para seleccionar los individuos que serán padres se ordena toda la población por su fitness. Después, se selecciona el 50% del total (300 individuos) de forma aleatoria ponderada, mediante "rueda por ruleta", priorizando la selección sobre aquellos individuos que tienen mejor fitness. Los individuos seleccionados se ordenan de forma aleatoria y se recorren de 2 en 2. Cada par de individuos tiene una probabilidad de 0.5 de cruzarse. Si no se cruzan, se añaden a la lista de la siguiente generación en su estado actual. En caso contrario, tiene lugar el cruce.

El cruce se implementa de 2 formas distintas: cruce de 1 punto y cruce de 2 puntos. Las figuras 4 y 5 muestran un ejemplo de cada uno de los cruces. Cada uno de los cruces tiene un 50% de probabilidades de ser elegido, por lo que se usa uno u otro de forma arbitraria. Una vez seleccionado el tipo de cruce que llevaremos a cabo, en ambos individuos se seleccionan los puntos de cruce necesarios de forma aleatoria, y los hijos, así como sus padres, se añaden a la lista de individuos de la siguiente generación.

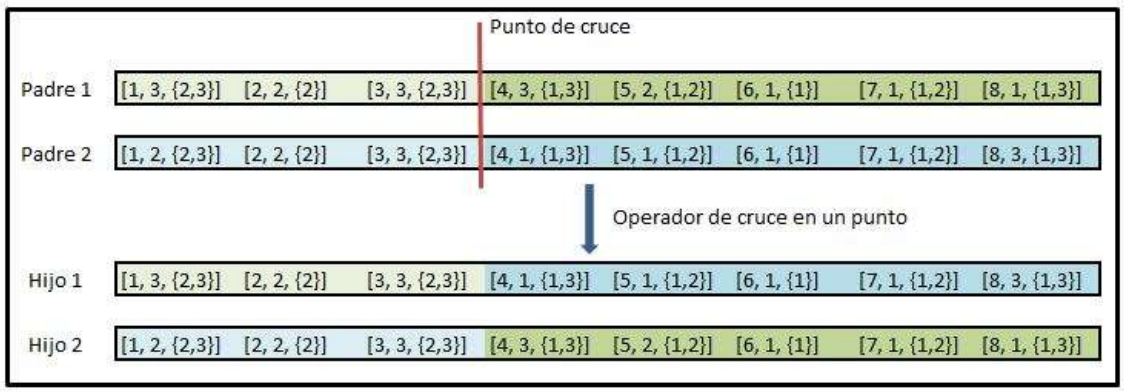

Figura 4: Operador de cruce en un punto

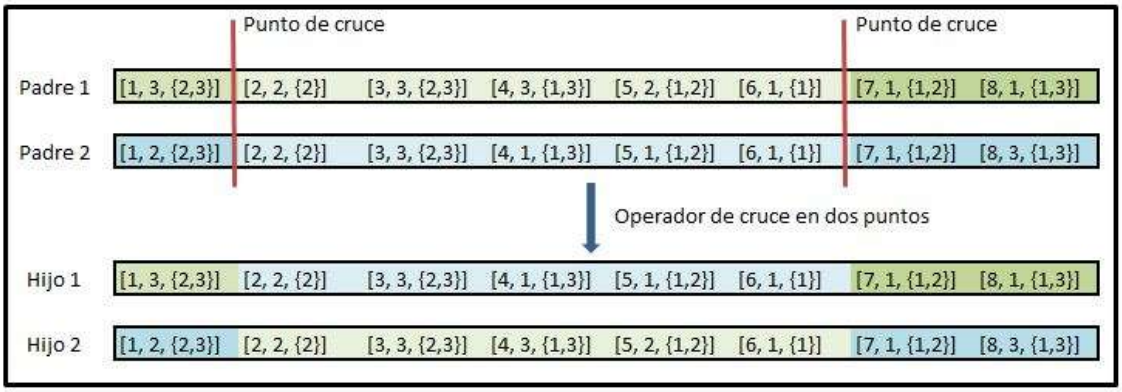

Figura 5: Operador de cruce en dos puntos

Antes de añadirse a la siguiente generación, los hijos tienen una probabilidad de mutar del 30%. Este porcentaje de mutación es un valor realmente alto teniendo en cuenta los valores normales que se le suele dar a este parámetro, sin embargo, diversas pruebas empíricas nos han mostrado que un factor de mutación tan agresivo nos ayuda a encontrar, por lo general, mejores soluciones debido a que se producen más saltos entre zonas dispersas del espacio de búsqueda.

El operador de mutación se implementa mediante una sustitución simple, esto es, se seleccionan dos tuplas t_i y t_k y se intercambian sus posiciones. Además, cada una de las tuplas tiene una probabilidad del 50% de cambiar su elemento m interno, esto es, cambiar la máquina en la que se asigna el trabajo n de la tupla en cuestión.

3.6 Sustitución

Recordemos que en la lista de la nueva generación hemos introducido tanto los padres, cruzados o sin cruzar, como los hijos. Para mantener una población de tamaño constante es necesario realizar un filtrado sobre esta lista para quedarnos únicamente con los 300 individuos que se juntarán con los individuos que no fueron seleccionados para el cruce, logrando así que la población siempre conste de 600 individuos. Para realizar esta sustitución se ordenan todos los individuos a valorar y se eligen los 300 mejores para volver a la población; los restantes morirán en este paso de la ejecución.

4. Estructura multiagente propuesta

A continuación, expondremos el framework multiagente utilizado para la resolución offline del problema de scheduling que nos ocupa. La ejecución de este framework se encuentra situado tras la ejecución del algoritmo genético (ver figura 2) y su misión es mejorar la solución que proporciona el AG y transmitir los conocimientos que ha adquirido al mejorar la solución a la población del AG, para que cuando éste vuelva a ponerse en funcionamiento, esté más informado y oriente la dirección de búsqueda de forma más inteligente.

El sistema multiagente propuesto cuenta con dos tipos de agentes: agente supervisor y agente trabajador, encargándose cada uno de ellos de diferentes tareas:

- **Agente supervisor**: El agente supervisor es el encargado de recibir la solución que ha generado el algoritmo genético, de crear tantos agentes trabajadores como se necesiten y también será el encargado de recibir el trabajo realizado por cada uno de los agentes y combinarlo de acuerdo a algún criterio para lograr construir una solución mejor que la que recibió como entrada.
- **Agente trabajador**: estos agentes son creados por el agente supervisor. Los trabajadores sólo se encargan de realizar una búsqueda local en una máquina en función de lo que les ordene el supervisor. Cuando los trabajadores han encontrado lo que el supervisor les ha pedido, le envían el resultado al supervisor y se autodestruyen, terminando así su tiempo de ejecución.

El funcionamiento completo del sistema es el siguiente:

- **Fase 1**: el algoritmo genético se ejecuta hasta que se consuma un tiempo máximo indicado. Cuando el tiempo termina, el algoritmo genético se detiene y le proporciona la mejor solución encontrada a nuestro

framework multiagente. El resto de la población se mantiene en memoria para ser utilizada durante la fase seis.

- **Fase 2**: el agente supervisor recibe la solución que proporciona el algoritmo genético. En este momento, el agente cuenta el número N de máquinas que tiene la instancia recibida del problema y crea N trabajadores, cada uno de ellos simulando cada una de las N máquinas. A cada uno de ellos le envía la lista de trabajos de una de las máquinas. Cada uno de los agentes se encarga de analizar una máquina. No existe ningún par de trabajadores que reciba la misma máquina como entrada. Además, el supervisor les ordena a los trabajadores que elijan un trabajo compartido de entre toda la lista que han recibido, en función de alguna política de selección de trabajo que detallaremos en la sección 4.1. Todos los trabajadores reciben la misma política de selección de trabajo, por lo que todos los trabajadores se basarán en los mismos criterios para seleccionar el trabajo adecuado.

- **Fase 3**: durante esta fase, cada uno de los trabajadores realiza una búsqueda local sobre el *schedule* de la máquina que ha recibido. Cuando encuentran el trabajo compartido que satisface el criterio que le ha sido indicado por el supervisor, se lo envían a éste y se autodestruyen, terminando así su ejecución y haciendo que no haga falta consumir más recursos computacionales ni de memoria para mantenerlos en el sistema.

- **Fase 4**: el supervisor recibe todos los trabajos compartidos que le envían sus trabajadores. De entre todos ellos, el supervisor elige uno en función de los mismos criterios (la misma política de selección de trabajo) que le ha enviado a sus trabajadores. De esta forma, los trabajadores han realizado una búsqueda local en una máquina extrayendo un sólo trabajo por máquina y de todos los trabajos encontrados por los trabajadores, el supervisor elige tan sólo uno, que llamaremos T*. Éste trabajo compartido elegido es al que se le debe de eliminar una de las posibles

máquinas en las que puede ejecutarse, haciendo que el espacio de búsqueda total sea más pequeño.

- **Fase 5:** en esta fase el supervisor eliminará una máquina M* de la lista de posibles máquinas en la que puede ejecutarse el trabajo T*. La máquina debe ser elegida en función de alguna política de selección de máquina, que detallaremos en la sección 4.4.

- **Fase 6:** si se ha alcanzado el timeout, el proceso termina, y se proporciona como solución la salida que ha construido el supervisor como salida del framework multiagente, es decir, la misma solución que proporcionó el algoritmo genético pero con la máquina M* eliminada de la lista de posibles máquinas para el trabajo T*. Por el contrario, si el timeout no ha vencido, se procede a modificar toda la población de individuos que mantiene el algoritmo genético. Esta modificación consiste en realizar los mismos cambios que se han realizado sobre un individuo en la fase cinco pero sobre toda la población. En concreto, consiste en, para todo individuo contenido en la población del genético, se eliminará la máquina M* de lista de posibles máquinas para el trabajo T*. A continuación, se vuelve a la fase uno y el proceso continúa hasta que el tiempo se consuma. Nótese, que a la hora de eliminar M* de la lista de posibles máquinas para T* en toda la población, es posible que el trabajo T* ya estuviera asignado a la máquina M* en ese individuo. En caso de que esto sucediera, sería necesario reubicar ese trabajo en otra máquina. A pesar de haber probado varias técnicas alternativas de reubicación, aquella que ha presentado mejores resultados ha sido reubicar el trabajo T* en una máquina aleatoria de entre todas las posibles.

La idea que subyace bajo todo este proceso es la de intentar mejorar las soluciones del algoritmo genético. Para esto, detenemos temporalmente el proceso de búsqueda llevado a cabo por el algoritmo genético y tratamos de modificar la población de una forma inteligente y a la vez reducimos el conjunto total de posibilidades que debe explorar el genético, por lo que además estamos reduciendo el espacio de búsqueda.

A medida que se realizan más iteraciones del framework propuesto, el espacio de búsqueda se va haciendo más pequeño, y si las políticas de selección de trabajo y máquina son adecuados, es de suponer que el espacio de búsqueda que se va podando en el proceso (las máquinas M* que se van eliminando) es un espacio en el que se encuentran peores soluciones que en el espacio que se queda dentro del proceso de búsqueda. De esta forma, orientamos al algoritmo genético a buscar en un espacio de búsqueda donde las soluciones se suponen mejores, en lugar de buscar sobre el espacio completo de búsqueda.

4.1 Políticas de selección de trabajo

En esta sección describiremos las distintas políticas de selección de trabajo que se han implementado. Recordemos que estas políticas son criterios de elección que tienen los agentes trabajadores para decidir qué trabajo eligen de entre todos aquellos que les ha enviado el agente supervisor. Posteriormente, cuando el agente supervisor recibe todas las elecciones de sus trabajadores, éste elige un trabajo de entre todos ellos, siguiendo también la misma política que hubiera ordenado seguir a sus trabajadores.

4.2 Worst Local Job (WLJ)

Cuando un trabajador recibe del supervisor una lista de trabajos pertenecientes a una máquina, éste debe elegir un trabajo compartido de esa máquina. Qué trabajo elegir depende de qué criterios de elección tenga el trabajador. Usando el criterio *Worst Local Job* (WLJ), los trabajadores seleccionan el trabajo que es considerado peor de entre todos los que les han sido asignados. Se considera que un trabajo A es peor que otro B si el efecto de introducir A en el schedule S empeora la función fitness de la máquina más de lo que lo empeora introducir el trabajo B en el schedule S. Es decir, el trabajo A es peor que el trabajo B si y sólo si:

$$F(S+A) > F(S+B)$$

donde:

- **S+X:** Representa que al schedule S se le añade el trabajo X en la mejor posición dentro de S. Se considera la mejor posición aquella que menos empeora (incrementa) el valor del fitness.

Llamamos WLJ al peor trabajo que existe dentro de cada máquina. Cada máquina tiene su propio WLJ, ya que cada máquina tiene dentro de sí misma un trabajo que sea el peor de todos los que contiene. En caso de empate entre trabajos, sólo uno de ellos, de forma aleatoria, es considerado peor. Así pues, utilizando el criterio WLJ, cada una de las máquinas elige el WLJ que tiene dentro de su schedule. Cada trabajador, para elegir cuál de todos sus trabajos es el WLJ, realiza los pasos que se describen a continuación.

Dados un schedule S asociado a una máquina y todos los parámetros necesarios para el problema (ver tabla 1), se procede de la siguiente forma:

1) El trabajador calcula la función fitness F(S) (ecuación 4) sobre el schedule S que ha recibido del supervisor.

2) Para todos los trabajos j_i contenidos en S, calcula:

$$\Delta i = F(S - j_i) - F(S) , \ \forall j_i \in S \qquad (6)$$

donde:

- **$S - j_i$:** Representa el schedule resultante de eliminar el trabajo j_i en el schedule S

Es decir, de forma iterativa se eliminan todos los trabajos del schedule uno a uno, y se calcula el fitness de la máquina sin ese trabajo. Posteriormente, el trabajo se vuelve a añadir al schedule y se elimina el siguiente. De esta forma obtenemos una lista de elementos Δi, donde cada Δi representa en cuanto empeora el fitness de la máquina al añadirle el trabajo j_i (en negativo. Cuanto más negativo es Δi, el trabajo j_i es considerado más malo). Así pues, definimos el WLJ de una máquina como:

$$WLJ = \text{argmin}_{Ji} \Delta i \qquad (7)$$

Cada trabajador le envía al supervisor su WLJ y también su Δi asociado. De esta forma, el supervisor recibe tantos trabajos y Δi como máquinas tiene la

instancia del problema. De todos los trabajos recibidos, el supervisor debe elegir uno. Siguiendo la misma política que se describe en la ecuación (7), el supervisor elige el trabajo que se considera peor de entre todos los peores de cada máquina recibidos. Este trabajo es el llamado *Worst Global Job* (WGJ). En resumen, cada trabajador elige el trabajo que es peor dentro de su máquina y se lo envía al supervisor. Después, el supervisor elige el peor de todos ellos, denominado WGJ. De esta forma, se elige el trabajo que de forma global es considerado peor.

4.3 Best Local Job (BLJ)

La idea de este criterio es similar a la del punto anterior, pero utilizando este criterio de selección, cada trabajador elige el trabajo que se considera mejor de entre todos los trabajos recibidos del supervisor. Reescribiendo la ecuación (7) podemos definir el BLJ como:

$$BLJ = \text{argmax}_{Ji}\ \Delta i \qquad (8)$$

En este caso, se elige el trabajo que tiene un Δi mayor, es decir, aquel trabajo que al insertarlo en el schedule de la máquina menos empeora el fitness de esa máquina.

De nuevo, el supervisor recibe todos los BLJ que le envían todos los trabajadores y siguiendo el mismo criterio que en (8) elige el Best Global Job (BGJ) sobre todos los trabajos que le han sido enviados. De esta forma, se elige el trabajo que de forma global es considerado mejor.

4.4 Política de selección de máquina

Llegados a este punto, ya hemos elegido un trabajo de entre todos los trabajos que tiene el problema, ya sea el WGJ o el BGJ. Una vez elegido un trabajo, hay que eliminarle una de las posibles máquinas en las que puede

ejecutarse. Recordemos que la idea principal del framework es ir reduciendo el tamaño del espacio de búsqueda de forma iterativa. Para ello, elegimos un trabajo de acuerdo a algún criterio (puntos 4.2 y 4.3) y posteriormente, al trabajo elegido, le eliminamos una de las posibles máquinas en las que puede ejecutarse. De esta forma, como el trabajo puede ejecutarse en una máquina menos, el número de combinaciones posibles es más pequeño. A continuación, proponemos una política de selección de máquina.

4.5 Worst Available Machine (WAM)

La idea de esta política de selección de máquina es elegir la peor máquina de entre todas las que puede ejecutarse el trabajo elegido. Ya que la máquina elegida será eliminada de la posible lista de máquinas para el trabajo elegido, seleccionar la peor máquina debería recortar el espacio de búsqueda sin sacrificar excesiva calidad en las soluciones.

De una manera más formal: Dado un trabajo compartido j buscaremos su peor máquina para eliminarla. Hay que tener en cuenta que el trabajo j, antes de este proceso, ya está asignado a una máquina k'. Para encontrar la peor máquina disponible realizaremos los siguientes cálculos:

$$\Delta_k = F(S_k + j) - F(S_k), \forall k \neq k' \in M_j \qquad (9)$$

donde:

- **M_j:** es el conjunto de máquinas en las que puede ejecutarse el trabajo j
- **S_k:** es el schedule asociado a la máquina k
- **$F(S_k)$:** es el fitness del schedule S_k (ecuación 4)
- **$S_k + j$:** es el schedule resultante de introducir el trabajo j en el schedule S_k en la mejor posición (aquella en la que menos empeora el fitness).
- **$F(S_k + j)$:** es el fitness del schedule S_k +j
- **Δk**: representa cuánto empeora el fitness de la máquina k si le introducimos el trabajo j.

Lo que hacemos en la ecuación (9) es calcular cuánto empeora cada máquina distinta a k' (cada máquina en la que el trabajo j no está asignado) si le introducimos el trabajo j en el schedule que ya tiene asignado en su mejor posición posible (aquella que menos empeora el fitness).

Posteriormente elegimos una máquina k* como:

$$k^* = \text{argmax}_{k \neq k'} \Delta k \qquad (10)$$

k* representa la máquina que peor recibe el trabajo j. Es decir, de todas las máquinas en la que el trabajo j no está asignado, k* es aquella a la que si le asignamos el trabajo j más empeora. Por lo tanto, k* tiene mucha probabilidad de convertirse en la peor máquina posible para alojar el trabajo j. Sin embargo, recordemos que el trabajo j ya está asignado en una máquina k', y es posible que k' fuera la peor máquina en lugar de k*. Si lo que empeora la máquina k* al recibir el trabajo j es menos que lo que empeora la máquina k', en la que el trabajo j ya se encuentra alojado, la peor máquina resultará ser k'. Por ello, es necesario calcular cuánto empeora la máquina k' a causa de tener alojado el trabajo j. Para ello, calculamos:

$$\Delta_{k'} = F(S_{k'}) - F(S_{k'} - j) \qquad (11)$$

Si $\Delta_{k^*} > \Delta_{k'}$ quiere decir que la máquina k* empeora más al recibir el trabajo j que lo que empeora la máquina k' al tenerlo incluido en su schedule, y por lo tanto, k* es la peor máquina posible para alojar el trabajo j. Por el contrario, si $\Delta_{k^*} < \Delta_{k'}$, nos encontramos en el caso en el que la peor máquina para alojar el trabajo j es aquella en la que ya está alojado. Así pues, podemos definir la WAM como:

$$\text{WAM} = k \in \{k', k^*\} : k = \text{argmax}\, \Delta_k \qquad (12)$$

Si WAM = k*, basta con eliminar k* de la lista de posible máquinas en las que se puede ejecutar el trabajo j y no es necesario realizar ninguna otra acción, ya que el trabajo j no está asignado en k*. Sin embargo, si WAM = k', resulta que el trabajo j ya está incluido en su peor máquina y por lo tanto no basta con eliminar k' de la lista de posibles, sino que además es necesario mover el trabajo j a otra máquina. Para ello, necesitamos elegir una nueva máquina para recibir

el trabajo j. Como nueva máquina receptora del trabajo j, el sistema elige la *Best Available Machine* (BAM), reescribiendo la ecuación (12) como:

$$BAM = k \in M_j : k = \operatorname{argmin} \Delta_k$$

Es decir, elegimos como BAM aquella máquina que menos empeora su fitness al recibir el trabajo j, ya que es considerada la mejor para incluirlo en su schedule. Una vez realizado este cálculo, el supervisor mueve el trabajo j a la máquina BAM y elimina la máquina k' de la lista de posibles máquina para el trabajo j.

Es decir, tras todo este proceso se ha elegido un trabajo, se le ha eliminado una máquina y quizá se le haya movido de lugar, si la máquina eliminada ha resultado ser en la que ya estaba situado. Estas acciones ahora se trasladan al conjunto de toda la población del algoritmo genético, que continuará su ejecución con normalidad tras cambiar su población (fase 6 de la ejecución del sistema, punto 4). Esto implica que hay que recorrer todos los individuos de la población y aplicarle exactamente los mismos cambios que se han aplicado en estos pasos, lo que significa eliminar de la lista de posibles máquinas de j la máquina WAM. Nótese que en la población del algoritmo genético el trabajo j puede estar alojado en la WAM independientemente de si en la solución con la que se ha trabajado (la mejor, recogida en la fase 2) estuviera o no en la WAM. Por lo tanto, si mientras se realiza la transformación de la población se encuentra un individuo en el que j esté en la WAM, éste se mueve a cualquier otra máquina elegida de forma aleatoria, aunque dentro de la máquina elegida, se mueve a la mejor posición posible dentro de ella.

De esta forma, conseguimos dotar al AG de algún conocimiento que quizá no supiera o que aún no había conseguido alcanzar debido al alto espacio de búsqueda que debe explorar, consiguiendo que la búsqueda esté algo más guiada y reduciendo de forma progresiva el espacio de búsqueda en el que el AG debe buscar soluciones. Téngase en cuenta que si la política de selección de trabajo así como la de selección de máquina a eliminar son aceptablemente buenas, el espacio de búsqueda que se va recortando no debería representar una pérdida significativa de calidad en las soluciones.

5. Resultados del entorno en scheduling multiobjetivo

A continuación, mostraremos los resultados obtenidos con las distintas políticas de selección de trabajo y mostraremos si consiguen cumplir nuestro objetivo de mejorar las soluciones que proporciona el algoritmo genético. Los experimentos se han llevado a cabo sobre un conjunto de instancias de problemas de 4 tamaños distintos: pequeño (30 trabajos), mediano (50 trabajos), grande (100 trabajos) y muy grande (250 trabajos). Estas instancias del problema no se han generado de forma aleatoria, sino que son el fruto de un meticuloso estudio (Paolucci, Anghinolfi, & Tonelli, 2016) que asegura que estas instancias comparten las propiedades de un caso real, lo que se ha hecho con la intención de poder suponer que el comportamiento de nuestro sistema sobre estas instancias es similar al comportamiento que tendrá sobre el caso real de estudio.

Como ya hemos indicado, tenemos 4 tamaños distintos de instancias, y para cada tamaño tenemos 125 instancias. Por lo tanto, en los resultados que se muestran a continuación se expresan los resultados medios de 125 experimentos distintos. A cada experimento se le ha dejado un timeout de 10 minutos.

5.1 WLJ vs AG

En la figuras 6, 7. 8 y 9 se muestra la comparativa de la política de WLJ con el algoritmo genético (AG). Como se puede observar, en instancias grandes la política WLJ comienza proporcionando resultados muy similares a los del algoritmo genético, sin embargo, a medida que avanza el tiempo, consigue ir mejorando las soluciones consiguiendo una notable diferencia con respecto al AG. Por otro lado, en la instancia más pequeñas se observa cómo, desde el inicio, los resultados de WLJ mejoran notablemente los resultados del AG.

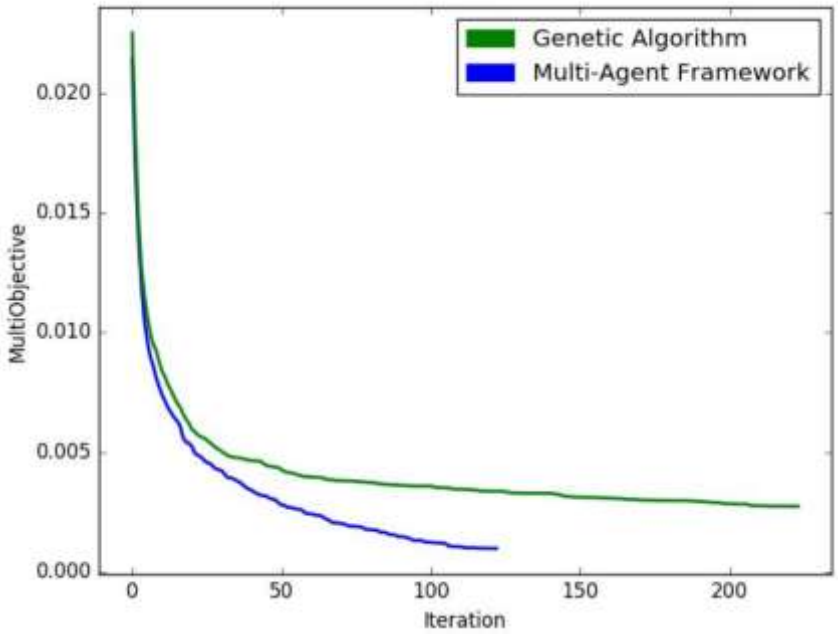

Figura 6: Comparativa WLJ vs AG, instancia muy grande

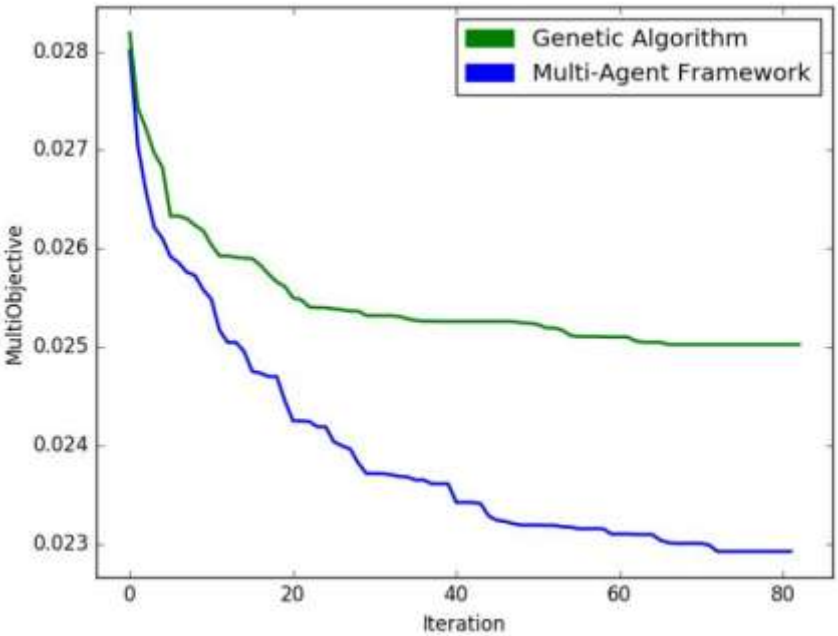

Figura 7: Comparativa WLJ vs AG, instancia grande

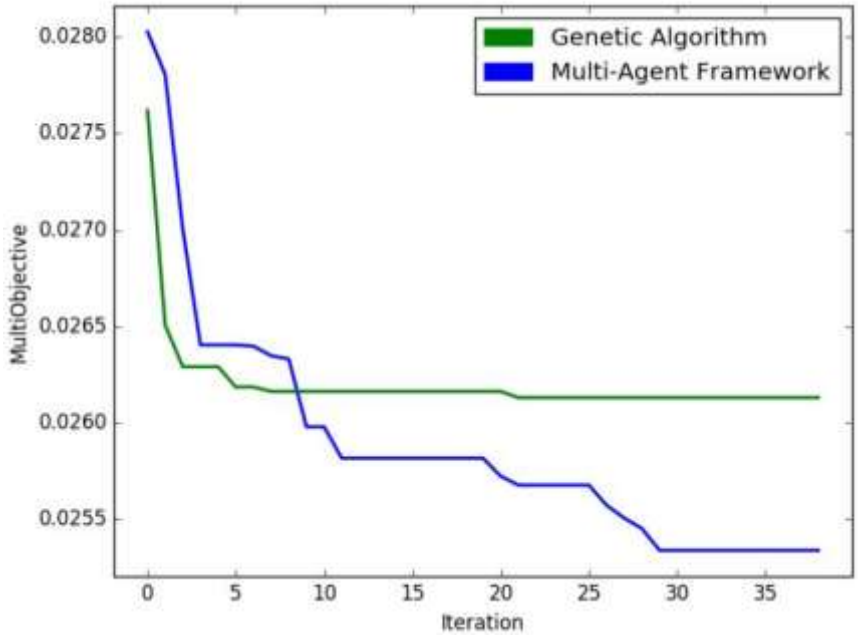

Figura 8: Comparativa WLJ vs AG, instancia mediana

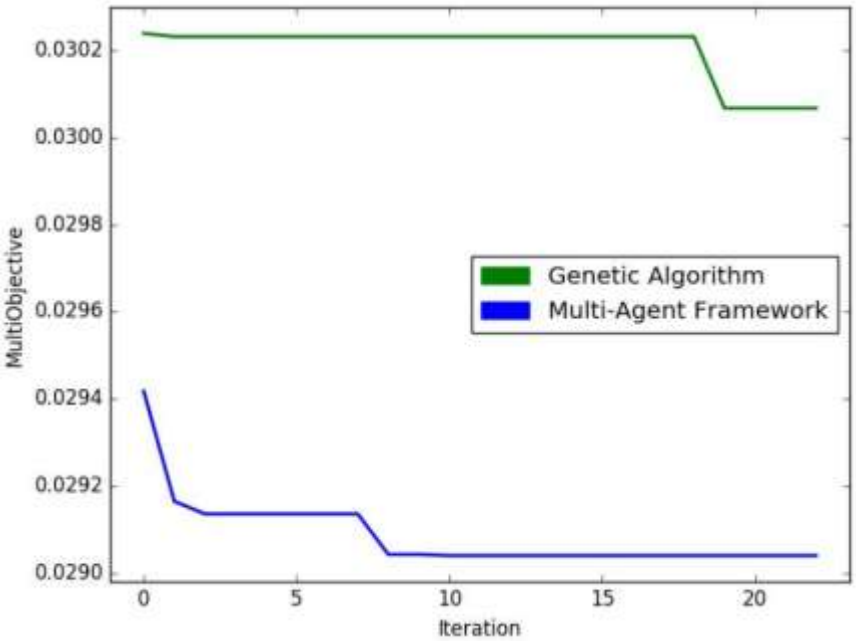

Figura 9: Comparativa WLJ vs AG, instancia pequeña

Además, se aprecia un fenómeno interesante. Como se puede observar si analizamos las instancias más grandes, en el mismo tiempo (10 minutos), nuestro sistema multiagente ha realizado muchas menos iteraciones, porque

cada iteración es mucho más costosa. Sin embargo, a pesar de realizar menos iteraciones, el resultado es mejor que el presentado por el AG con más iteraciones. Esto evidencia que la poda del espacio de búsqueda que estamos haciendo es una poda correcta, mientras que el AG al tener un espacio de búsqueda tan grande le es más difícil dar con soluciones mejores.

5.2 BLJ vs AG

En las figuras 10, 11, 12 y 13 se muestra la comparativa de resultados de la política BLJ con los resultados que proporciona el AG. De nuevo, se puede observar como el framework propuesto proporciona mejores resultados en el mismo tiempo que el AG.

A pesar de que en este caso también se observa que el framework mejora los resultados del algoritmo genético, en este caso se aprecia cómo, en un número bajo de iteraciones, el AG proporciona mejores resultados que la política BLJ, o incluso que supera notablemente a BLJ, como se puede apreciar en la instancia mediana.

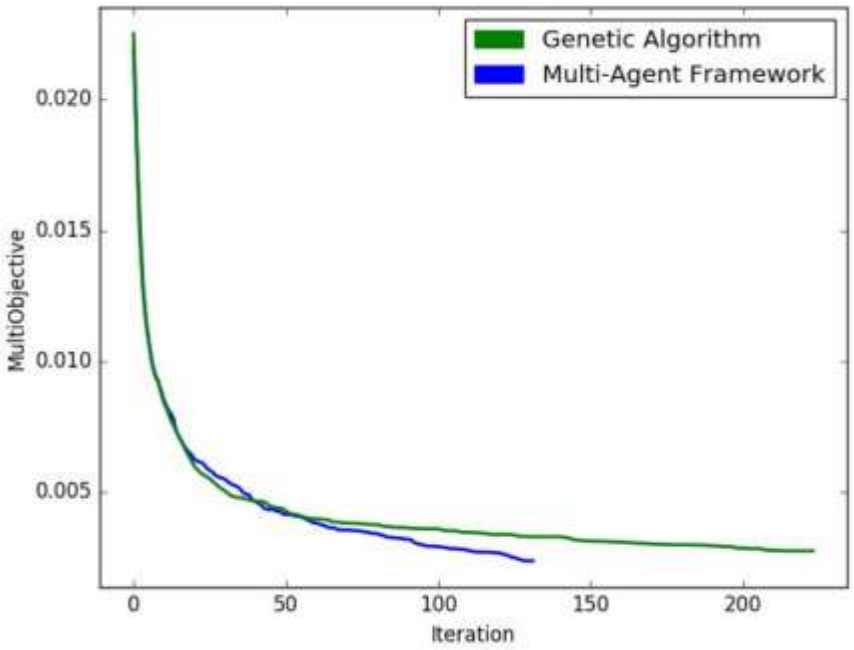

Figura 10: Comparativa BLJ vs AG, instancia muy grande

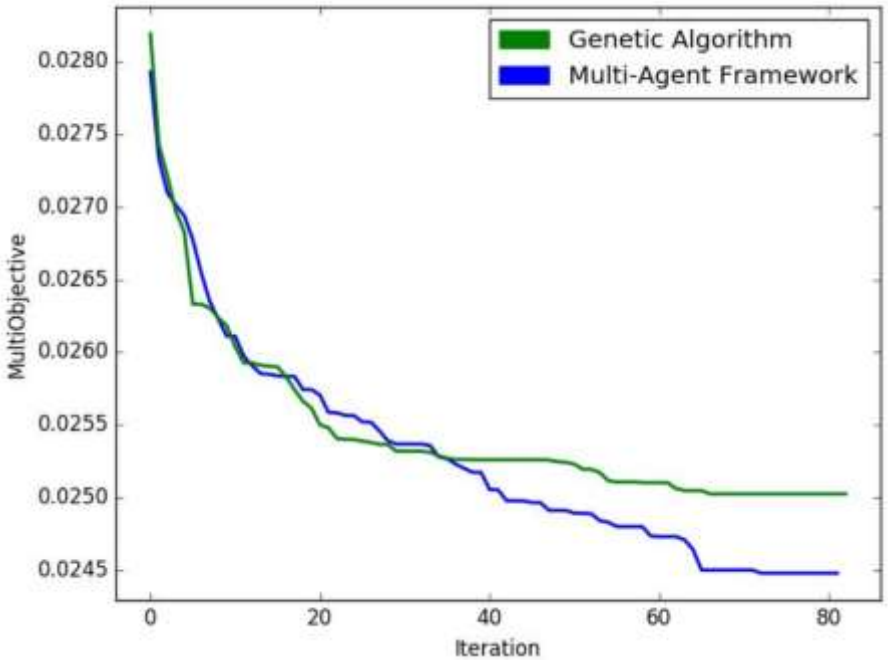

Figura 11: Comparativa BLJ vs AG, instancia grande

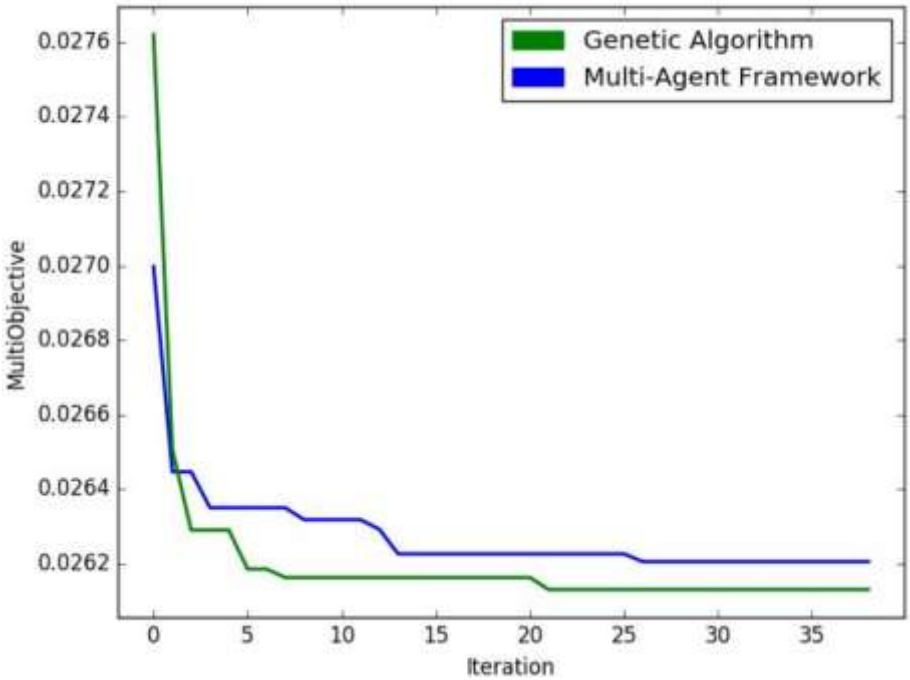

Figura 12: Comparativa BLJ vs AG, instancia mediana

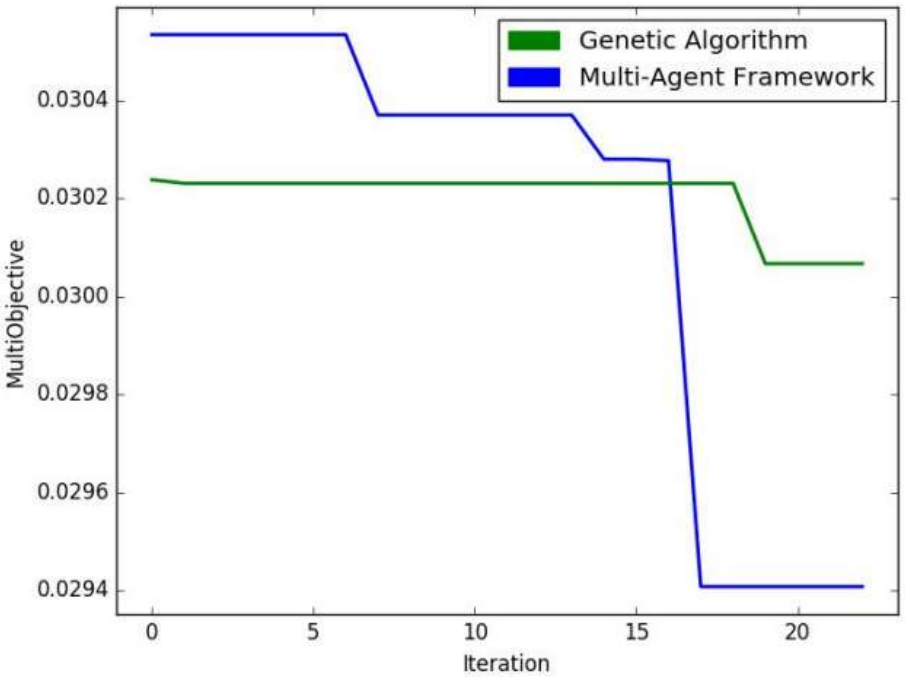

Figura 13: Comparativa BLJ vs AG, instancia pequeña

Esto puede deberse a que el criterio del BLJ consume mucho tiempo sin que realmente influya sobre la solución, debido a que si un trabajo era bueno, seguramente lo fuera porque se encontraba en una buena máquina, y por lo tanto, no sea necesario hacerle rescheduling a otra máquina mejorando así la solución. Por otro lado, si el trabajo ya estaba en una buena máquina, el AG, probablemente, ya lo tendría en cuenta y mantuviera esta propiedad entre las distintas generaciones, por lo que quitarle una mala máquina a un buen trabajo, no aporta gran cosa al AG que éste no supiera ya.

Sin embargo, de nuevo, a medida que avanzan las iteraciones, se aprecia como el AG se pierde en un espacio de búsqueda tan grande, ralentizando la convergencia a soluciones mejores, mientras que BLJ, a pesar de haber avanzado más lentamente al principio, consigue superar al AG debido a la poda progresiva del espacio de búsqueda que va realizando de forma iterativa, a excepción de lo que ocurre en la instancia de tamaño mediano.

5.3 WLJ vs BLJ

Las 2 políticas de selección de trabajo han mostrado, en mayor o menor medida, que son capaces de mejorar los resultados del algoritmo genético. Sin embargo, falta decidir cuál de ellas resulta más efectiva.

A continuación veremos cuál de las 2 políticas de trabajo implementadas ha proporcionado mejores resultados. Como se muestra en las figuras 14, 15, 16 y 17, los resultados que ha proporcionado WLJ han sido notablemente mejores que los resultados proporcionados por la política BLJ

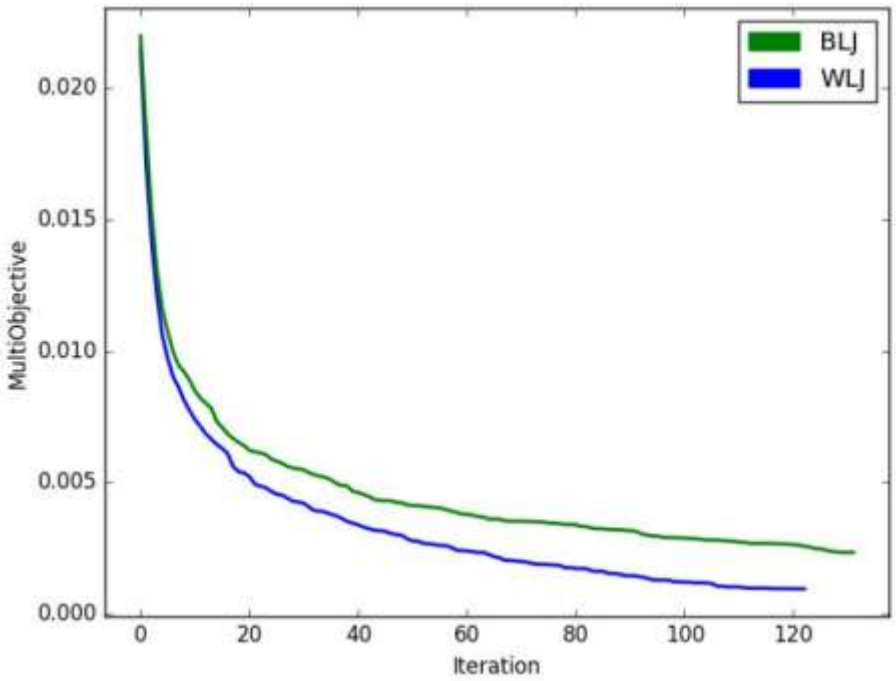

Figura 14: Comparativa BLJ vs WLJ, instancia muy grande

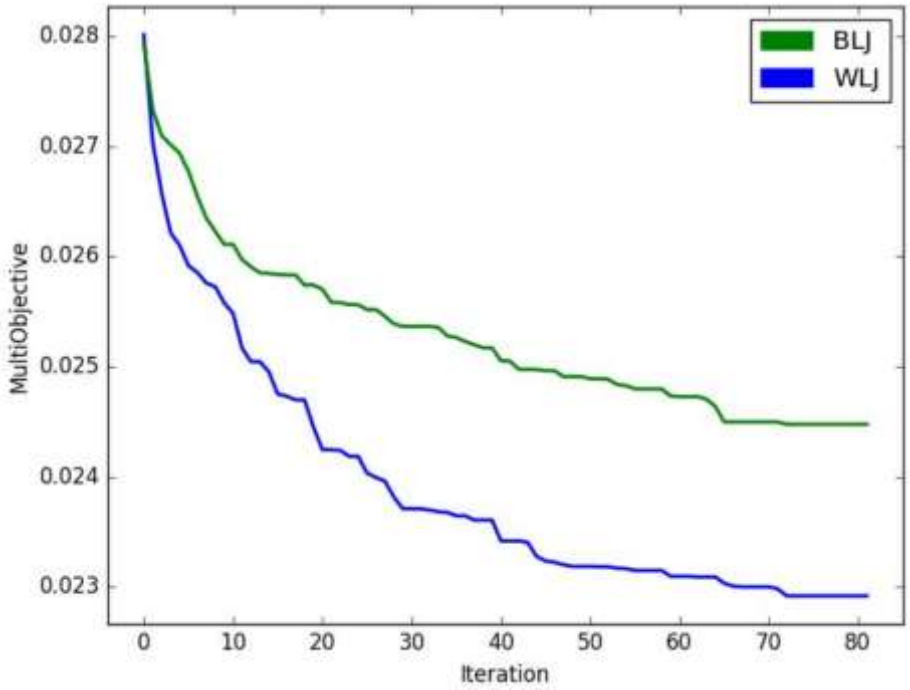

Figura 15: Comparativa BLJ vs WLJ, instancia grande

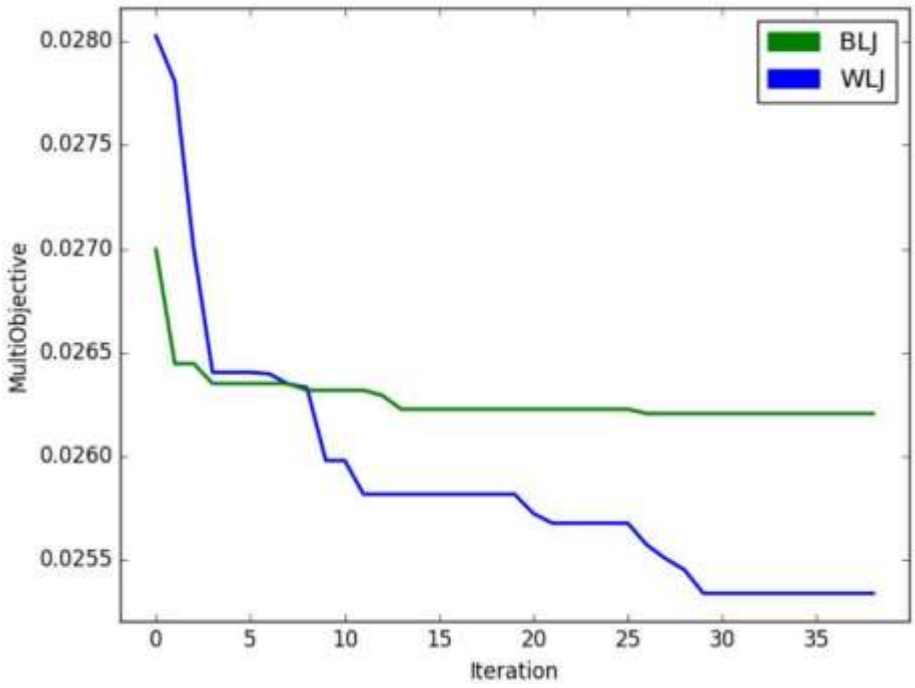

Figura 16: Comparativa BLJ vs WLJ, instancia mediana

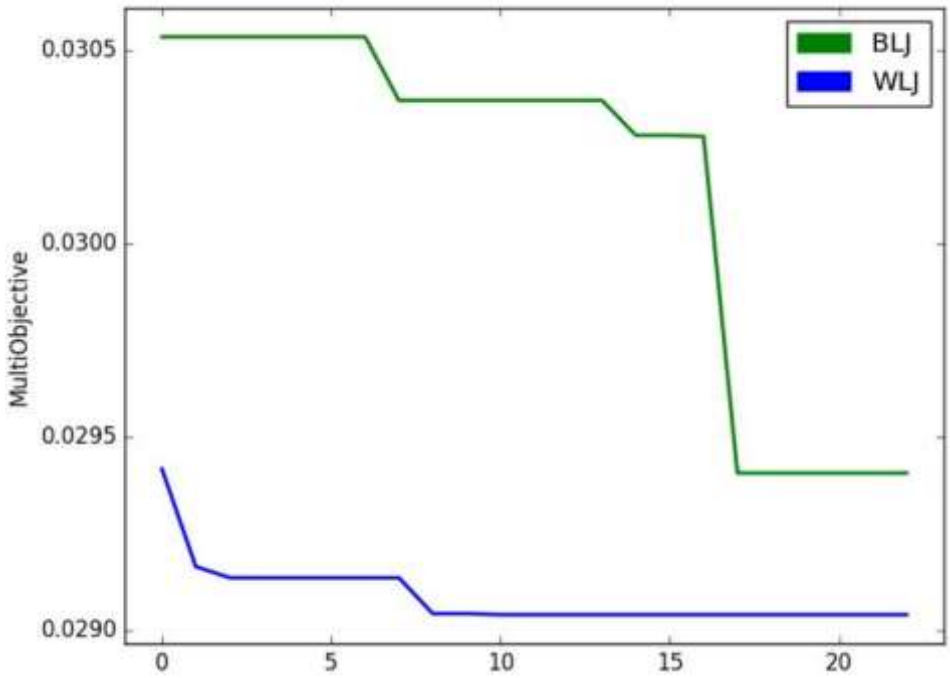

Figura 17: Comparativa BLJ vs WLJ, instancia pequeña

Estos resultados pueden parecer incoherentes, ya que, de forma intuitiva puede parecer que BLJ debería proporcionar mejores resultados que WLJ, ya que, BLJ elige un trabajo que es bueno e impide que se mueva, por lo que si un trabajo es bueno, se mantendrá como bueno sin cambiarse de máquina. Es decir, mantener inmóviles los buenos trabajos debería proporcionar mejores resultados que mantener inmóviles los malos trabajos.

Sin embargo, la clave para explicar los resultados obtenidos reside en el rescheduling. Recordemos que cuando al trabajo elegido se le selecciona una máquina para ser eliminada, si el trabajo ya está en esa máquina, se le mueve a otra. En concreto, se le mueve a la mejor máquina disponible para él (BAM), por lo que siempre que realizamos rescheduling, estamos mejorando la solución de forma notable, ya que movemos un trabajo desde su peor máquina a su mejor máquina. Dicho de otra forma, cuantos más rescheduling hagamos, más mejoraremos la solución. Además, como ya dijimos en la sección anterior, recordemos que dejar inmóvil un trabajo que ya es bueno, puede no influir demasiado en el AG, pues probablemente éste ya conociera esa propiedad y la mantuviera entre las distintas generaciones.

La figura 18 muestra la cantidad de reschedulings que se llevan a cabo según cada política de selección trabajo. Como se observa, cuando se utiliza la política WLJ se realiza el rescheduling más veces, lo que, como ya mencionamos, aporta 2 cosas: mejoramos de forma inmediata la solución, y además trasladamos a la población del AG una nueva propiedad que quizá éste no hubiera identificado como positiva.

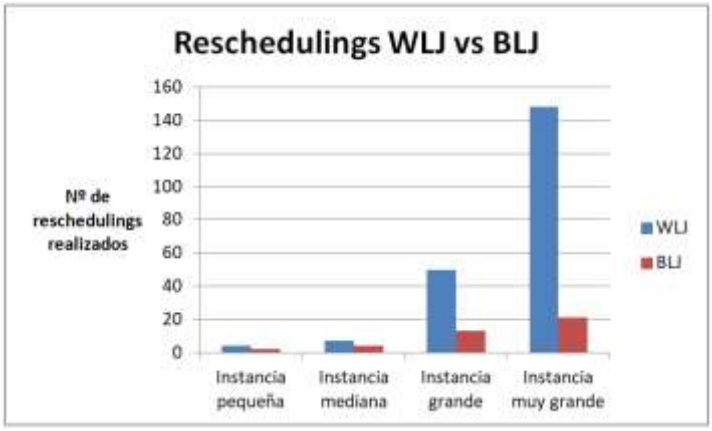

Figura 18: Comparativa nº de reschedulings

6. Rescheduling online para problemas de scheduling multiobjetivo.

6.1 Trabajo relacionado

La industria de producción está cada vez más centrada en afrontar los retos medioambientales actuales, por lo que sus procesos industriales deben estar optimizados tanto en términos de rentabilidad y productividad como en sostenibilidad. Teniendo en cuenta que muchos de estos procesos son dinámicos, muchas de las soluciones obtenidas pueden no ser válidas después de que algunas interrupciones o incidencias ocurran. En (Salido M.A, 2016) los autores se centran en la recuperabilidad ante incidencias en problemas de *job-shop scheduling* donde las máquinas pueden trabajar a diferentes velocidades. En ese contexto, dada una incidencia, el objetivo principal se centra en

recuperar la solución original mediante el *rescheduling* del mínimo número de tareas posible. Para ello, proponen una técnica de *punto de match-up,* para determinar la zona necesaria que reordenar y encontrar una solución factible. También proponen un algoritmo memético para encontrar schedules que minimicen el consumo de energía manteniendo la restricción del tiempo total empleado. En este artículo trabajan con una extensión del problema de job-shop scheduling donde cada máquina puede trabajar a distintas velocidades (Salido, Escamilla, Barber, Giret, Tang, & Dai, 2013), donde se asume que cuando un trabajo es procesado a una alta velocidad, su tiempo total de procesado disminuye mientras que su consumo de energía aumenta (Fang, Uhan, Zhao, & Sutherland, 2013).

En la literatura que rodea el tema, encontramos diversos métodos de scheduling dinámico para trabajar con entornos donde se requiere un scheduling online. (Arnaout, 2014) ataca el problema de scheduling en máquinas paralelas no relacionadas con tiempos de configuración dependientes de la secuencia y distintos ratios de error o llegadas de trabajos urgentes. Para hacer esto, desarrollaron una nueva regla de reparación, denominada *Minimum Weighted Cmax Difference (MWCD),* y fue comparada con el resto de algoritmos en términos de calidad y estabilidad.

(Hall & Potts, 2004) trabaja con problemas de scheduling donde un conjunto de trabajos ya han sido organizados para minimizar el coste de la función objetivo cuando una nueva tarea llega al entorno y crea una incidencia. El sistema que toma las decisiones necesitar insertar los trabajos nuevos en la organización sin perder excesiva calidad en las soluciones.

En (Qi, Bard, & Yu, 2006) se propone el problema de actualizar el schedule de una máquina cuando una incidencia, ya sea aleatoria o prevista, ocurre después de que un subconjunto de trabajos hayan sido ya organizados. La solución propuesta difiere de la mayoría de los análisis de rescheduling en que el coste asociado a la desviación existente entre la solución original y la nueva se incluye en el modelo.

(Vieira, Herrmann, & Lin, 2000) presenta nuevos modelos analíticos que pueden predecir el comportamiento de las estrategias de rescheduling y

cuantificar el equilibrio entre las diferentes medidas a tener en cuenta. Para hacer esto, se estudian tres estrategias de rescheduling: periódicas, híbridas y guiadas por eventos dependientes del tamaño de la cola.

(Vieira, Herrmann, & Lin, 2003) propone un sistema para entender las estrategias de rescheduling, las distintas políticas y métodos en sistemas de rescheduling de manufactura. El trabajo realizado expone métodos para generar schedules robustos y métodos para actualizar dichas soluciones.

En (Subramaniam & Raheja, 2003) se estudian las típicas incidencias de problemas de job-shop y sus procesos de reparación se descomponen en cuatro pasos de forma genérica, que son conseguidos mediante heurísticas aplicadas al rescheduling.

(Herroelen & Leus, 2004) se revisan diversas metodologías para scheduling proactivo y reactivo. Los autores también ofrecen un sistema que permite identificar la metodología de scheduling adecuada para diferentes entornos posibles.

En el trabajo que posteriormente exponemos, se presenta un sistema para conseguir la gestión de incidencias una vez la solución ha sido alcanzada, para lo que representaremos la incidencia como una tarea más a organizar e intentaremos introducirla en la solución existente tratando de minimizar la pérdida de calidad en el proceso. Compararemos la técnica de scheduling implementada con datos de referencia obtenidos propagando la incidencia una vez ocurre, sin realizar ningún tipo de rescheduling ni alterando la solución más allá de introducir la incidencia en el lugar que ha ocurrido.

6.2 Introducción

Llegados a este punto, ya tenemos soluciones factibles y viables para el problema de scheduling obtenidas mediante el sistema offline. Dadas estas soluciones, basta con seguir la planificación creada y poner las máquinas a trabajar sobre las tareas asignadas en el orden que especifique la planificación planteada por la solución.

Sin embargo, el problema no puede darse por resuelto, ya que suponer que podemos directamente ejecutar los planes creados por el sistema offline, separando tareas en máquinas y dejándolas trabajar en el orden preestablecido, es suponer que la naturaleza del contexto es estática, permanente, sin cambios, etc.

No obstante, la naturaleza de la realidad contextual al problema dista mucho de este planteamiento, donde pueden ocurrir cambios, errores imprevisibles o problemas de cualquier tipo, que impidan llevar a cabo el plan creado por el sistema offline.

Estos problemas que impiden ejecutar el scheduling confeccionado por el sistema se denominan problemas dinámicos donde ocurren incidencias. Las incidencias más comunes se deben a que una máquina o un conjunto de máquinas se estropeen o, por cualquier motivo, queden inutilizables durante un periodo de tiempo.

Cuando esto sucede, el scheduling creado no puede ejecutarse, dado que las tareas que deberían ejecutarse en las máquinas averiadas no pueden llevarse a cabo. Un posible solución a esto podría ser, simplemente, esperar a que las máquinas estén reparadas y continuar con las tareas asignadas desde que se interrumpió el trabajo. Sin embargo, esta opción retrasaría mucho el fin de las tareas y perjudicaría a la función de evaluación, lo que conllevaría un uso de recursos alejado de la optimalidad con los costes asociados.

Por lo tanto, dado este contexto incierto donde los planes creados corren el riesgo de no poder llevarse a cabo, trataremos de gestionar las incidencias de la forma más eficiente posible, tratando de optimizar de nuevo la función objetivo descrita en la ecuación (4). Sin embargo, en este caso no basta con optimizar la función objetivo, y por lo tanto los recursos necesarios, sino que además, hay que optimizar el tiempo en el que se resuelven las incidencias. Tengamos en cuenta que las incidencias ocurren durante el tiempo de producción, y que una máquina se estropee no debería perjudicar ni interferir en el correcto funcionamiento del resto de máquinas. Por lo tanto, dada una incidencia, trataremos de resolverla, además de optimizando la ecuación (4), en

el menor tiempo posible, utilizando estrategias voraces que rápidamente puedan subsanar el error ocurrido.

Para ello, crearemos un nuevo módulo que se integrará en el sistema offline escrito en el punto anterior, que será el encargado de tratar las incidencias cuando estas ocurran y de intentar que el resto de máquinas sigan trabajando como estaba previsto.

6.3 Estructura del entorno de rescheduling

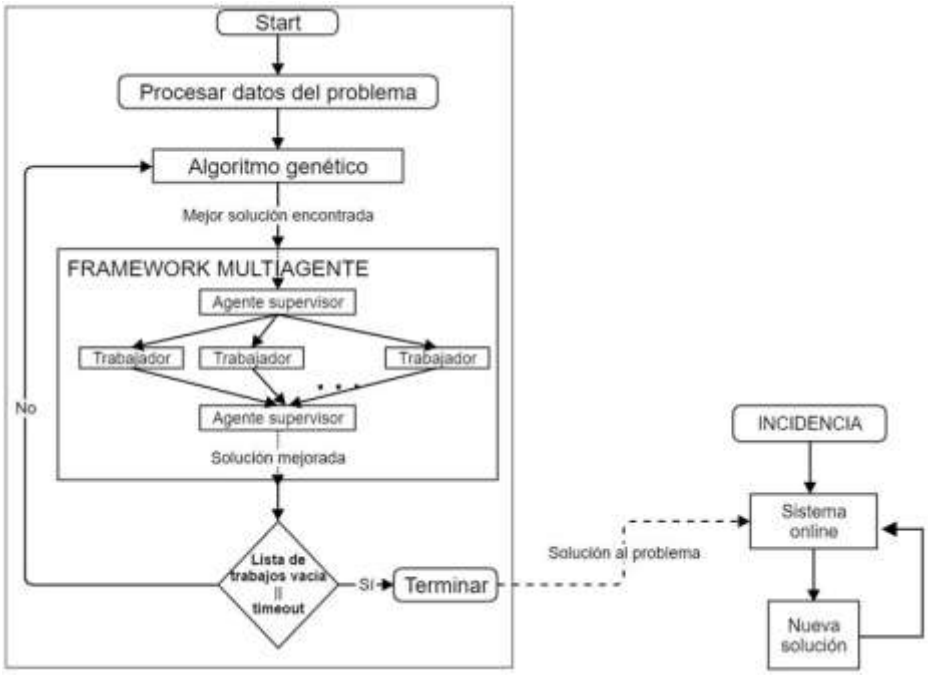

Figura 19: Sistemas offline y online integrados

A continuación, exponemos las modificaciones hechas sobre el sistema de la figura 2 para integrar el sistema online de control de incidencias. La figura 19 muestra el resultado de la integración de ambos sistemas. Como se puede observar, el sistema offline, también denominado *solver*, se sitúa tras la ejecución del sistema online.

El solver recibe como entrada la solución que ha generado el sistema offline y su labor es monitorizar esta solución, manteniéndose a la espera y comprobando que todas las máquinas y tareas se están ejecutando como estaba previsto en la solución recibida. Durante la espera, es posible que una incidencia ocurra.

Cuando la incidencia ocurre, el sistema la recibe como entrada, modifica la solución obtenida intentado optimizar los recursos en un tiempo razonable para introducir la incidencia en la planificación original y la nueva salida se devuelve para su puesta en marcha. Posteriormente, esta nueva solución pasa a ser monitorizada, por lo que el sistema es capaz de gestionar de nuevo más incidencias entrantes sobre la misma solución. De esta forma, el solver puede ir modificando la solución de forma iterativa de tal forma que siempre se pueda gestionar una incidencia entrante.

6.4 Definición formal de incidencia

Las incidencias son sucesos que ocurren de forma imprevista que provocan que el plan original no pueda llevarse a cabo, por ejemplo: averías de máquinas, trabajadores ausentes, cortes de suministros eléctricos, accidentes, etc. Independientemente de qué provoque la incidencia o de cuál sea su naturaleza, su representación es siempre la misma. Es decir, al sistema no le interesa qué o quién ha causado la incidencia, si no las características que tiene esta incidencia, y todas las incidencias, independientemente de su origen, tienen exactamente los mismos parámetros.

Estos parámetros que definen a una incidencia son: el momento en el que ocurre la incidencia, qué máquina se ve perjudicada y cuánto tiempo es necesario para arreglar el daño causado y que la máquina vuelva a estar operativa. De manera más formal, una incidencia i es una tupla de tres elementos:

$$i = [\text{st}, \text{m}, \text{tts}]:$$

donde:

- **st (starting time):** Es el momento en el que la incidencia ocurre.
- **m:** Es la máquina perjudicada por la incidencia.
- **tts (time to solve):** Es el tiempo necesario para resolver la incidencia, por ejemplo: el tiempo necesario para arreglar la máquina estropeada, el tiempo necesario para que el operario vuelva a su puesto de trabajo, etc. La máquina m estará apagada o inoperativa durante tts unidades de tiempo a partir de st.

Nótese que una incidencia sólo puede ocurrir sobre una máquina. Sin embargo, pueden haber sucesos que afecten a varias máquinas simultáneamente, por ejemplo: un corte de suministro eléctrico durante algunas horas. Este tipo de sucesos se formalizan añadiendo una incidencia distinta por cada máquina afectada, ya que, como dijimos, el origen de las mismas es indiferente para el sistema. Por lo tanto, no es necesario establecer relaciones de ningún tipo entre incidencias que tengan el mismo origen, ya que para el sistema, serán incidencias distintas e independientes si ocurren en máquinas distintas.

Además, cabe notar que una incidencia, a nivel conceptual, puede ser vista como una tarea más a incluir en el schedule. Se trataría, bajo esta visión, de una tarea no compartida (que solo puede ser ejecutada en una máquina) y que de forma imperativa debe empezar en el instante en que se produce.

6.5 Funcionamiento del sistema

A continuación expondremos la estrategia seguida por el solver para incluir las incidencias en la solución previamente encontrada. En la figura 20 se muestra una posible solución sobre una máquina que se ve afectada por una incidencia mientras ejecuta una tarea.

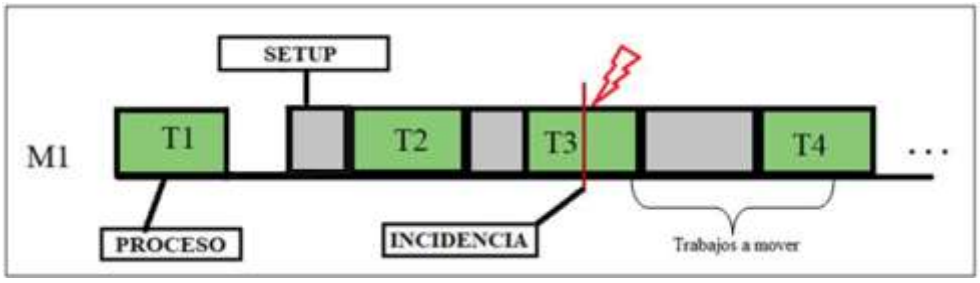

Figura 20: Esquema ejemplo de asignación

Mientras una máquina se ejecuta, las incidencias pueden ocurrir en cualquier momento. Siempre que una incidencia ocurre, hay que calcular cuales son los trabajos que potencialmente se ven afectados por la incidencia y cuáles no. Sin pérdida de generalidad, todos los trabajos afectados serán aquellos que se encuentren durante o después de la ocurrencia de la incidencia.

Todos los trabajos afectados no podrán ser movidos a otras máquinas, ya que algunos de ellos no serán trabajos compartidos y por lo tanto sólo podrán ejecutarse en la máquina afectada. Para todos los demás, el sistema intentará reubicarlos en otra máquina, siempre y cuando moverlos sea más beneficioso que mantenerlos a la espera de que la máquina se repare.

Por lo tanto, una vez ocurre la incidencia, el sistema recorre todos los trabajos afectados y de ellos selecciona únicamente los compartidos. Para todos los trabajos compartidos seleccionados, se calcula su BAM (apartado 4.5) y si su BAM es distinta a la máquina en la que está contenida, entonces se mueve a su BAM. De lo contrario, el trabajo se mantiene en la máquina afectada.

En la tabla 6 se muestra el pseudocódigo del algoritmo que realiza el tratamiento de las incidencias.

Entrada: Una solución S y una incidencia I
Salida: Una solución S' con la incidencia I incluida
Algoritmo:

```
1  //I[0] es el st de la incidencia (punto 6.4)
2  //I[1] es la maquina afectada (punto 6.4)
3  trabajosAfectados = seleccionaTrabajosPosterioresA(I[0], I[1])
4  lista = [x for x in trabajosAfectados if x in trabajosCompartidos]
5
6  incluirIncidencia(I, S)
7
8  for t in lista:
9    bam = selectBAM(t)
10    if bam =! maquinaRota:
11      moverTrabajo(t, bam)
```

Tabla 6: Algoritmo de tratamiento de incidencias

El comportamiento del algoritmo es el siguiente: cuando una incidencia ocurre en una máquina, todos los trabajos que deberían ejecutarse en la máquina afectada, si se pueden mover a otra máquina, se mueven, siempre y cuando el cambio sea beneficioso para la función objetivo, ya que todos los movimientos pueden no ser productivos, en función de la duración de la

incidencia y de la ganancia o pérdida que tenga cada una de las máquinas implicadas en el movimiento del trabajo.

Cuando el algoritmo termina, la nueva solución incluye la incidencia como una tarea más en la máquina en la que ha ocurrido y todas las tareas compartidas de esa máquina se han realojado en una máquina distinta en caso de que esto mejorara la solución frente a la opción de mantenerlas en espera hasta que la incidencia se solucione.

Posteriormente, la nueva solución se entrega de nuevo al solver que monitoriza el estado de avance de la ejecución del plan proporcionado por esta nueva solución. Si el solver detecta otra incidencia o incidencias, el procedimiento comienza desde el principio hasta que todas las incidencias se han incluido en las máquinas afectadas.

6.6 Resultados del sistema online

El solver se ha ejecutado sobre las soluciones obtenidas en el punto 5. Las incidencias que se necesitan como entrada se han generado de forma aleatoria con las siguientes restricciones:

- Las incidencias deben aparecer en el primer 80% de tiempo total de trabajo de la máquina. Esto es así porque se ha observado de forma experimental que cuando las incidencias aparecen muy retrasadas en el tiempo, no se pueden reubicar de forma satisfactoria, ya que el sistema no tiene apenas tiempo de margen para solventar el problema debido a que al final del plan el número de trabajos afectados se reduce drásticamente, por lo que no se pueden obtener mediciones realistas del funcionamiento del solver y tan solo se genera una extensión del makespan.

- La duración de las incidencias se estima de forma aleatoria, eligiendo al azar una duración entre uno y el doble de la duración del trabajo más largo.

- La máquina a la que se le asigna la incidencia se calcula de forma completamente aleatoria sin prioridades ni ponderaciones de ningún tipo.

A la hora de evaluar el comportamiento del sistema necesitamos establecer unos datos de referencia a partir de los cuales podamos determinar si la utilización del sistema resulta efectiva o no. Estos datos de referencia se han calculado introduciendo la incidencia sin realizar ningún tratamiento, es decir, cuando ocurre una incidencia, simplemente la introducimos en el momento en que se ha producido dentro del schedule y el resto de tareas esperan hasta que la incidencia se resuelve. Cuando la incidencia queda solventada, la máquina continúa las tareas por donde se había detenido. De esta forma, la referencia se establece en el comportamiento que tiene el sistema cuando, al ocurrir una incidencia, no se aplica ningún tipo de inteligencia en cómo gestionarla.

En contraposición a este *baseline,* se muestra el comportamiento de nuestro solver, que en lugar de esperar a que la incidencia termine, trata de reubicar las tareas para no retrasarlas más en el tiempo como se expone en el punto 6.5.

Para ejemplificar ambos comportamientos presentamos el siguiente ejemplo. Supongamos que tenemos la siguiente lista de trabajos y sus posibles máquinas en las que ejecutarse:

Trabajo: j	**M_j**		**Trabajo: j**	**M_j**
1	{1,3}		7	{2}
2	{1,2}		8	{3}
3	{1,2,3}		9	{1,3}
4	{1}		10	{1,2,3}
5	{2,3}		11	{2,3}
6	{1,2,3}		12	{1,2,3}

Tabla 7: Ejemplo. Posibles máquinas asignables a cada trabajo

Además, supongamos también que tenemos la solución que muestra la figura 21, y que mientras la estamos ejecutando se produce un fallo en la máquina 1 mientras se ejecuta el trabajo 2, tal y como se muestra en la figura.

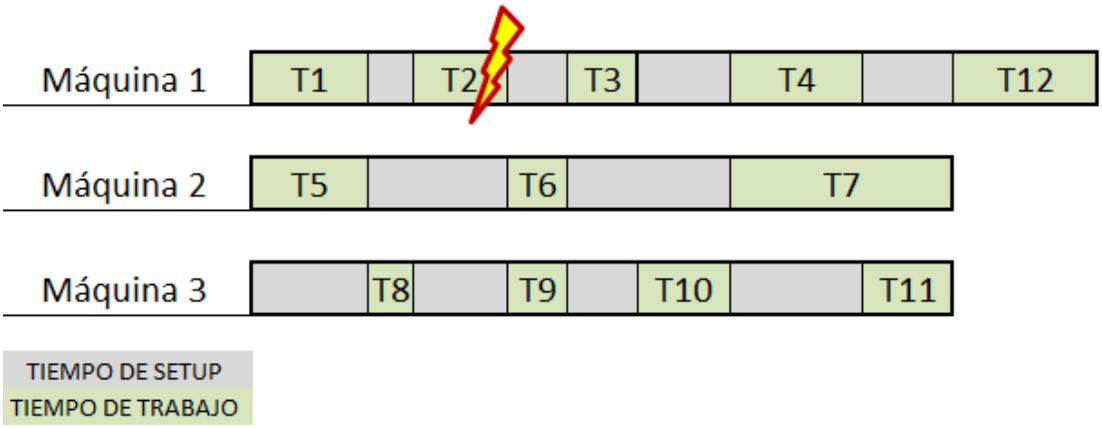

Figura 21: Ejemplo. Scheduling original que sufre una incidencia durante su ejecución

Ante esta situación, el *baseline* que se propone consiste en propagar la incidencia y retrasar el resto de trabajos lo que provoca que todos los trabajos asociados a la máquina retrasen su ejecución. La figura 22 representa el estado en que quedaría la solución al aplicar esta técnica, donde todos los trabajos asignados originalmente a la primera máquina han tenido que esperar a que la incidencia se resuelva, retrasando así el tiempo de finalización y perjudicando a la función de evaluación global.

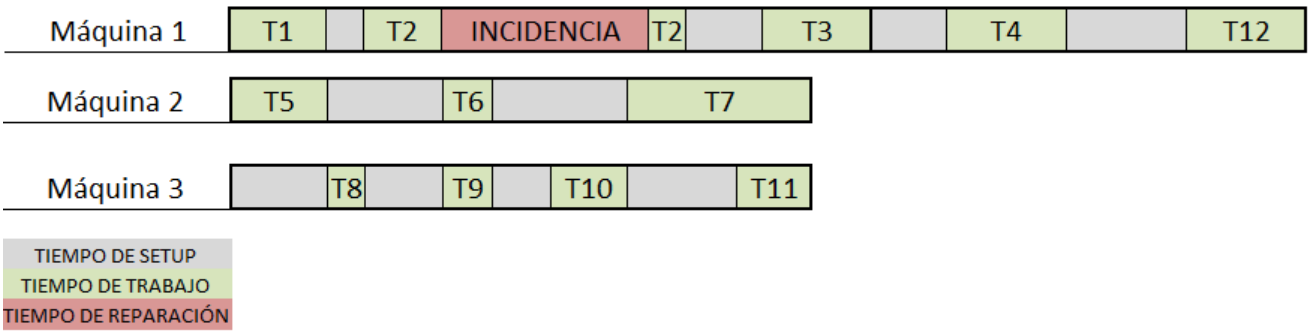

Figura 22: Ejemplo. Técnica de resolución propuesta como baseline: propagación

Nótese además que el trabajo T2 no ha tenido que comenzar desde el principio, sino que en el momento en que se produce la incidencia, todo el trabajo hasta entonces realizado queda almacenado y por lo tanto, cuando la máquina es reparada, solo debe terminar la parte de trabajo que le quedaba pendiente. Lo que sí es interesante es destacar el tiempo total que ha necesitado

la máquina 1 para terminar sus tareas asignadas, que se ha visto notablemente incrementado y perjudicado.

En contraposición a este comportamiento proponemos el comportamiento llevado a cabo por nuestro solver. La figura 23 muestra un posible estado en el que podría acabar la solución tras aplicarle el solver, donde movemos los trabajos compartidos pendientes alojados en la máquina accidentada (3 y 12) a otras posibles máquinas, siempre y cuando la ganancia en la función objetivo sea superior a la perdida que supone esperar en la máquina averiada.

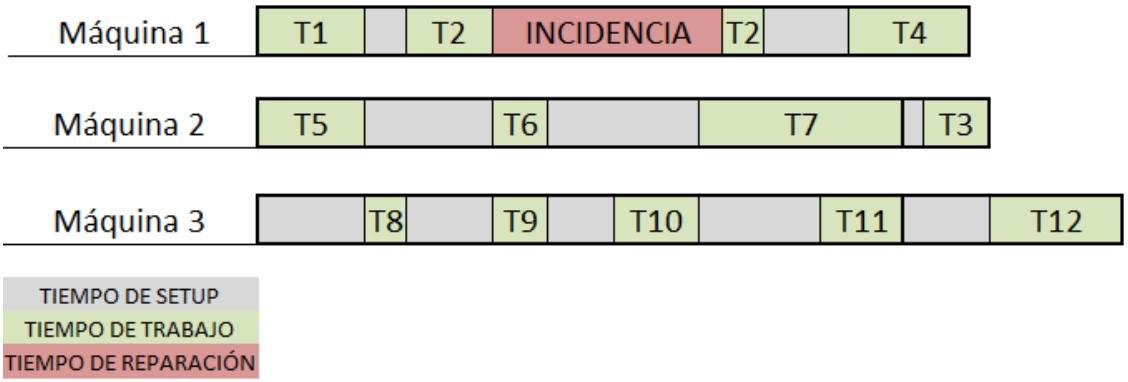

Figura 23: Ejemplo. Resolución de la incidencia mediante el solver propuesto

Como se puede observar, las tareas T3 y T12 se han realojado en otras máquinas, logrando que el tiempo total del sistema no se extienda tanto como si se propagara la incidencia. Nótese que todas las tareas que pueden realojarse, no tienen por qué hacerlo. Por ejemplo, la tarea T2 ha permanecido en la máquina averiada, a pesar de que podría haberse movido. Esto es porque, a pesar de que una tarea pueda moverse, puede no interesar hacerlo ya que al moverla habría que añadir, además del tiempo de trabajo de la tarea, un nuevo tiempo de setup que puede ser peor incluso que la penalización sufrida al esperar la resolución de la incidencia para terminar la tarea interrumpida. En cualquier caso, el solver siempre busca la mejor opción: si mejora mover la tarea, ésta se mueve, de lo contrario, permanece en su lugar. Como último apunte, téngase en cuenta que hay tareas que no pueden realojarse y no queda otra alternativa que esperar a que la incidencia se resuelva. Este es el caso de la tarea cuatro, que solo puede ejecutarse en la máquina averiada y por lo tanto la única posibilidad es esperar a que ésta se repare.

En las figuras 24, 25, 26 y 27 se muestra el resultado de aplicar el solver a los distintos tamaños de instancia. El número de incidencias se ha fijado en 32, lo que representa un número más que suficiente para simular el estrés que puede suponer un entorno real. Tengamos en cuenta, que cada incidencia simula una avería de la maquinaria, por lo que 32 máquinas averiadas trata de de ser un margen superior de lo que pudiera pasar en la realidad. Aún así, en el anexo 1 se muestra la evolución del sistema con 128 y 256 incidencias.

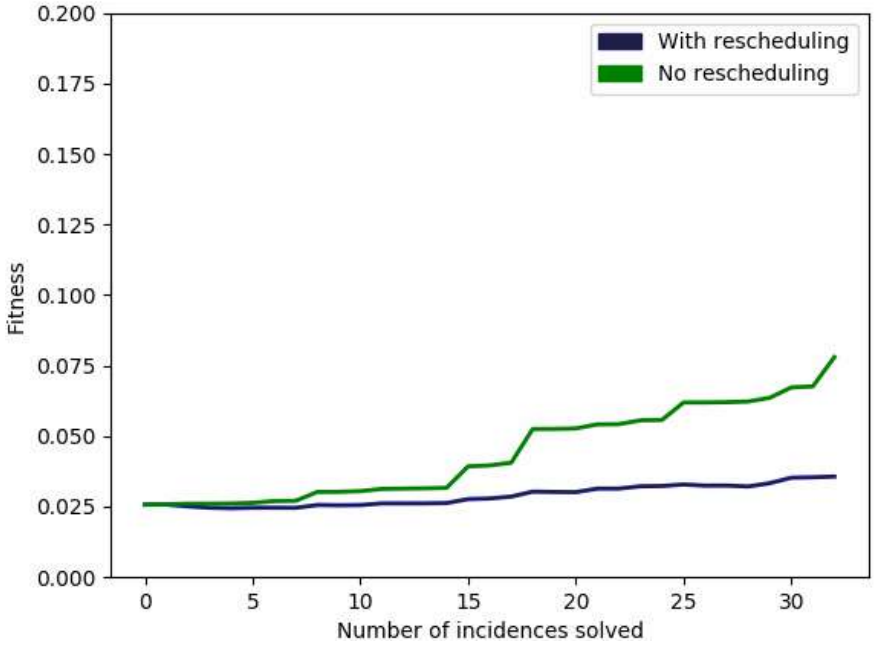

Figura 24: Evaluación del solver, instancia muy grande

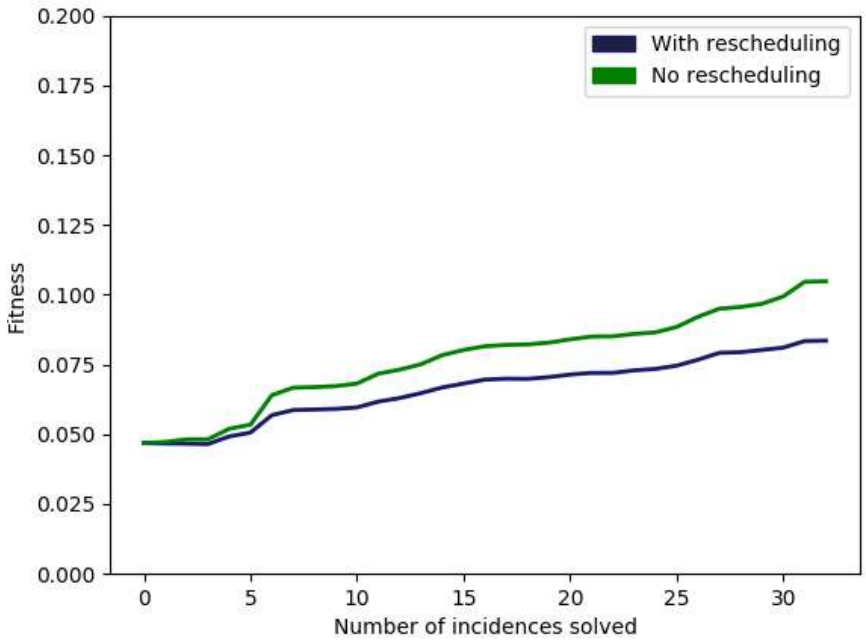

Figura 25: Evaluación del solver, instancia grande

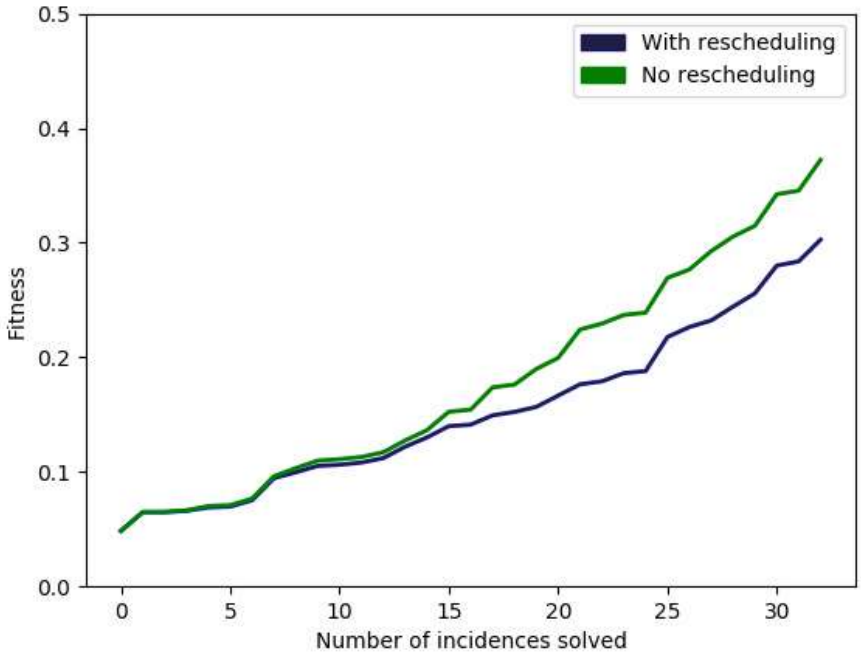

Figura 26: Evaluación del solver, instancia mediana

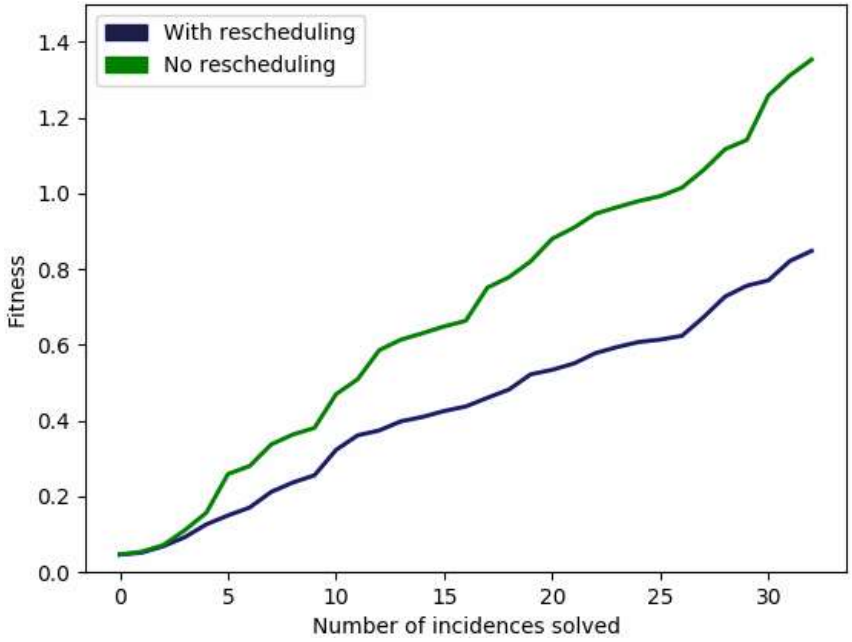

Figura 27: Evaluación del solver: instancia pequeña

Los tiempos medios de repuesta para resolver una incidencia se muestra en la tabla 8.

Instancia	Tiempo (segundos)
Muy grande	0.31
Grande	0.19
Mediana	0.15
Pequeña	0.04

Tabla 8: Tiempo de respuesta del solver

Como se pude observar, a medida que crece el número de incidencias que se van introduciendo al sistema, el fitness de la solución empeora de forma progresiva a cada nueva incidencia. Además, también se evidencia cómo nuestro sistema solver es capaz de gestionar las incidencias de una forma efectiva, ya que se mejora la solución utilizándolo frente a la opción de no utilizarlo.

Incluso en algunas instancias, se observa cómo al no utilizar el solver la solución es hasta dos veces peor que sí usando el solver. Esto nos deja ver que la estrategia implementada por el sistema online es efectiva a la hora de tratar las incidencias y que los resultados son satisfactorios.

Además, los tiempos de respuesta son muy ajustados en el tiempo, todos inferiores a un segundo, lo que encaja con lo que se esperaba de un sistema online que debe resolver los problemas en el menor tiempo posible para tratar de afectar lo menos posible a la evaluación de las soluciones.

7. Conclusiones

Con la realización de este proyecto se ha presentado un sistema de resolución de problemas de "Green scheduling" (scheduling que contempla la eficiencia energética) y se le ha integrado un sistema online de resolución de incidencias en tiempo real. Nótese que a lo largo del trabajo se ha entendido como "eficiencia" la capacidad de lograr el objetivo deseado con el mínimo consumo de recursos posible, o en nuestro caso, de energía total consumida. Es por ello que la función de evaluación de nuestro sistema incluye el consumo de energía y, por lo tanto, al minimizar nuestra función objetivo estamos minimizando también, entre otros recursos, la energía total que consume el sistema de producción, logrando así integrar el proyecto en un contexto de "Green scheduling" donde se prioriza aquellas soluciones que reducen el consumo energético necesario para ser ejecutadas.

Por lo que respecta a los objetivos que nos planteábamos al inicio del proyecto (punto 1.4), podemos afirmar que han sido conseguidos de forma satisfactoria, mejorando incluso las metas propuestas en algunos casos.

Por un lado, los objetivos del sistema offline eran dos: conseguir soluciones competentes en tiempos inferiores a 30 minutos y conseguir mejorar las soluciones del algoritmo genético. Tal y como se muestra en los resultados del apartado 5, ambos han sido conseguidos. Recordemos que a cada uno de los experimentos se le ha dejado un timeout de 10 minutos, lo que supera con creces los 30 minutos que nos habíamos propuesto. Además, la política WLJ, descrita en el punto 4, ha mostrado proporcionar mejores resultados en todas las instancias y en cualquier número de iteraciones que el algoritmo genético, por lo que podemos afirmar que el segundo objetivo de mejorar el algoritmo genético se ha cumplido.

Por otro lado, los objetivos del sistema online consistían en integrarlo con el sistema offline, conseguir gestionar las incidencias en tiempo del orden de segundos y lograr la mínima degradación posible al gestionar una incidencia. En este caso también podemos afirmar que los 3 objetivos han sido conseguidos

satisfactoriamente, ya que ambos sistemas se integran como muestra la figura 19, el tiempo de respuesta del solver es siempre inferior a un segundo, como muestra la tabla 8 y la degradación entre incidencias supone mínimos cambios en la función de evaluación, apenas perceptibles entre una incidencia y otra, como muestran las figuras 24, 25, 26 y 27.

Por lo tanto, y habiendo aclarado que todos los objetivos propuestos al inicio del proyecto se han conseguido alcanzar o incluso superar, podemos afirmar que la realización del proyecto ha sido exitosa y que el sistema resultante es una herramienta efectiva para trabajar sobre problemas de "Green Scheduling" de tamaños grandes y que es capaz de gestionar la naturaleza dinámica y cambiante de la realidad sin sacrificar excesiva calidad en las soluciones.

Publicaciones relacionadas con este trabajo

Con el desarrollo de este trabajo, en concreto, con el desarrollo de la primera parte del mismo, que hace referencia al sistema offline de búsqueda de soluciones mediante la poda sucesiva del espacio de búsqueda, se ha conseguido una publicación en el *workshop* COPLAS-2017 de la 27ª Conferencia internacional de planning automático y scheduling (ICAPS 2017). La publicación conseguida es la siguiente:

G. Nicolo, M. A. Salido, S. Ferrer, A. Giret, F.Barber. (2017). A Multi-Agent Approach using dynamic constraints to solve energy-aware. 27th International Conference on Automated Planning and Scheduling, ICAPS2017-COPLAS , 31-37

Bibliografía

Agnetis, A., Billaut, J., Gawiejnowicz, S., Pacciarelli, D., & and Soukhal, A. (2014). Multiagent Scheduling: Models and Algorithms. *Springer* .

Arnaout, J.-P. (s.f.). Rescheduling of parallel machines with stochastic processing and setup times. *J. Manuf. Syst.* , 376-384.

Fang, K., Uhan, N., Zhao, F., & Sutherland, J. (2013). Flow shop scheduling with peak power consumption constraints. *Ann. Op. Res. 206* , 115-145.

Hall, N., & Potts, C. (2004). Rescheduling for new orders. *Op. Res. 52* , 440-453.

Herroelen, W., & Leus, R. (2004). Robust and reactive project scheduling: a review and classification of procedures. *Int. J. Prod. Res. 42* , 1599-1620.

Liu, N., Abdelrahman, M. A., & Ramaswamy, S. (2007). A complete multiagent framework for robust and adaptable dynamic job shop scheduling. *IEEE Transactions on Systems Man, and Cybernetics-Part C* .

Newman, S., Nassehi, A., Imani-Asrai, R., & Dhokia, V. (2012). Energy efficient process planning for CNC machining. *CIRP J Manuf Sci Technol* , 127-136.

Nicoló, G., Salido, M. A., Ferrer, S., Giret, A., & Barber, F. (2017). A Multi-Agent Approach using dynamic constraints to solve energy-aware. *27th International Conference on Automated Planning and Scheduling, -COPLAS-* , 31-37.

Paolucci, M., Anghinolfi, D., & Tonelli, F. (2016). Facing energy-aware scheduling: a multi-objective extension of a scheduling support system for improving energy efficiencyin a moulding industry. *Soft-computing* , 1-12.

Qi, X., Bard, J., & Yu, G. (2006). Disruption management for machine scheduling: the case of spt schedules. *Int. J. Prod. Econ. 103* , 166-184.

Salido, M., Escamilla, J., Barber, F., Giret, A., Tang, D., & Dai, M. (2013). Energy-aware parameters in job-shop scheduling problems. *GREEN-COPLAS 2013: 2013 Workshop on Constraint Reasoning, Planning and Scheduling Problems for a Sustainable Future* , 44-53.

Subramaniam, V., & Raheja, A. (2003). Maor: a heuristic-based reactive repair mechanism for job shop schedules. *Int. J. Adv. Manuf. Technol. 22* , 669-680.

Verfaillie, G., & Schiex, T. (1994). Solution reuse in dynamic constraint satisfaction problems. *AAAI-94* .

Vieira, G., Herrmann, J., & Lin, E. (2000). Predicting the performance of rescheduling strategies for parallel machine systems. *J. Manuf. Syst. 19* , 256-266.

Vieira, G., Herrmann, J., & Lin, E. (2003). Rescheduling manufacturing systems: a framework of strategies, policies, and methods. *J. Sched. 6* , 39-62.

Wallace, R. J., Grimes, D., & Freuder, E. C. (2009). Solving Dynamic Constraint Satisfaction Problems by Identify Stable Features. *IJCAI* .

Anexo

A continuación, se muestra el comportamiento del sistema online de reschedule con un número de incidencias más grande al mostrado en la memoria del proyecto, por si se desea consultar el comportamiento del solver a medida que las incidencias van creciendo hasta límites muy altos.

128 incidencias:

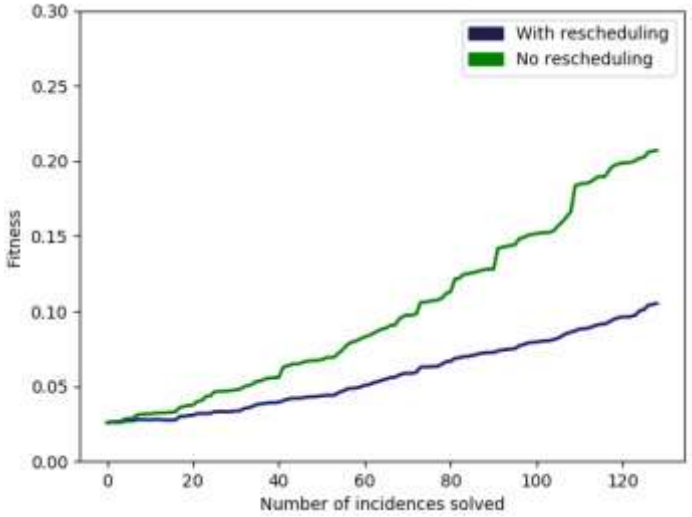

Figura 28: Comportamiento del solver, instancia muy grande

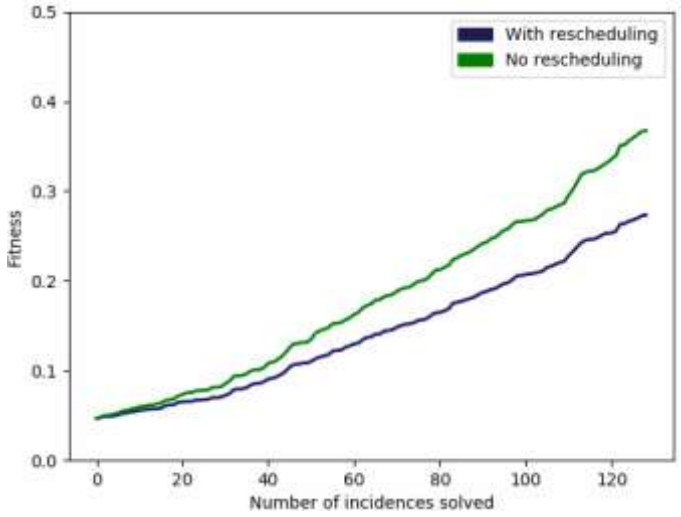

Figura 29: Comportamiento del solver, instancia grande

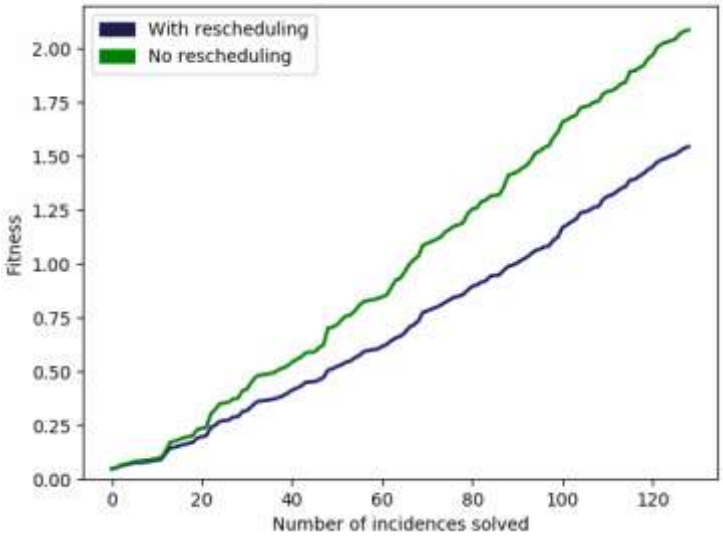

Figura 30: Comportamiento del solver, instancia mediana

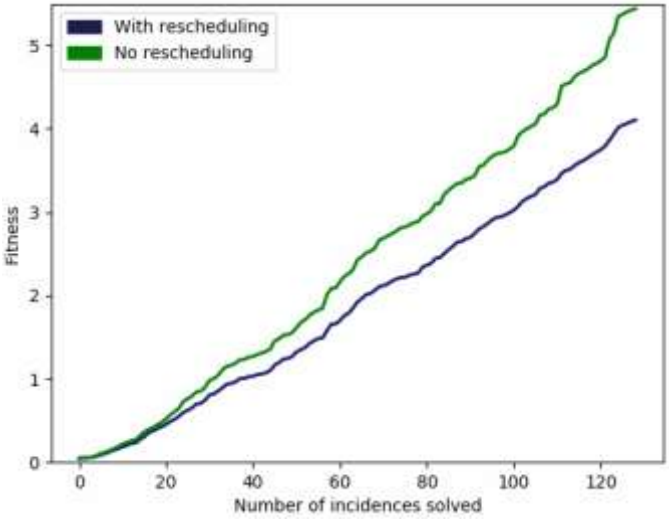

Figura 31: Comportamiento del solver, instancia pequeña

256 incidencias:

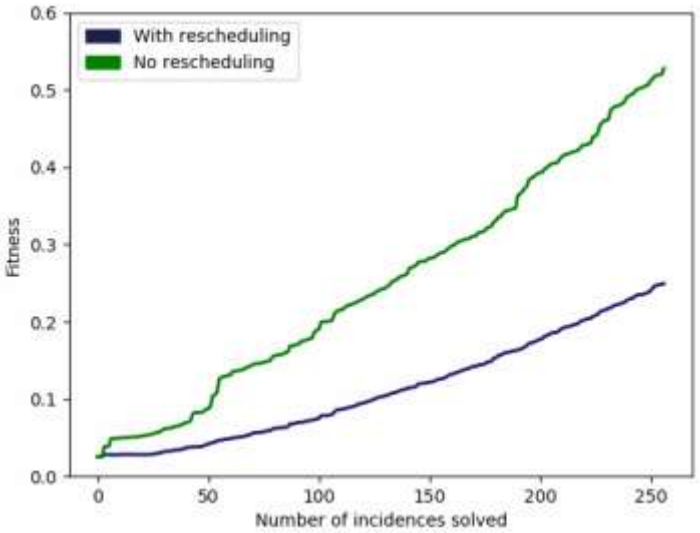

Figura 32: Comportamiento del solver, instancia muy grande.

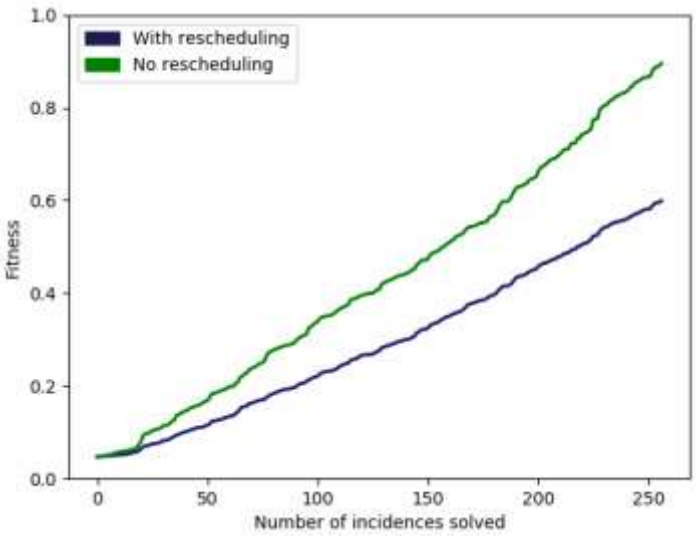

Figura 33: Comportamiento del solver, instancia grande

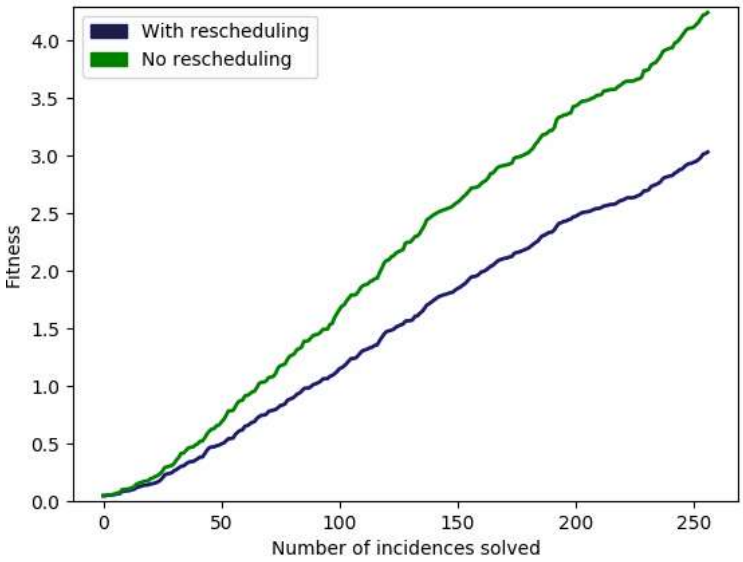

Figura 34: Comportamiento del solver, instancia mediana

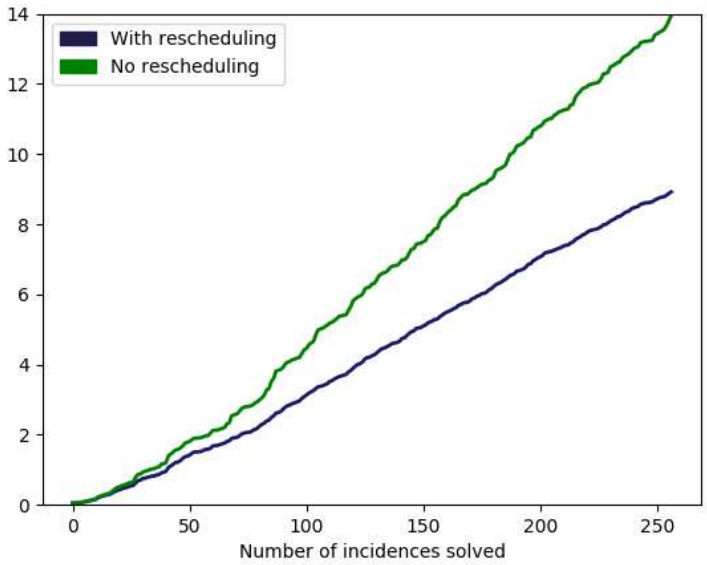

Figura 35: Comportamiento del solver, instancia pequeña