Universidad Central “Marta Abreu” de Las Villas

Facultad de Matemática, Física y Computación

Trabajo para optar por el Título de

Máster en Ciencia de la Computación

Sistema para la optimización del proceso de secuenciación de reportes con múltiples restricciones

Autor:

Ing. Jessica Coto Palacio

Tutores:

Dra. Yailen Martínez Jiménez

MSc. Beatriz M. Méndez Hernández

Santa Clara, 2018

Hago constar que el presente Trabajo para optar por el Título de Master en Ciencia de la Computación ha sido realizado en la facultad de Matemática, Física y Computación de la Universidad Central “Marta Abreu” de Las Villas (UCLV) como parte de la culminación de los estudios de la Maestría en Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución para los fines que estime conveniente, tanto de forma total como parcial y que además no podrá ser presentado en eventos ni publicado sin la previa autorización de la UCLV.

_____

Firma del Autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y que el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____	_____
Firma del Tutor	Firma del Tutor
Dra. Yailen Martínez Jiménez	MSc. Beatriz M. Méndez Hernández

Resumen

Los problemas de secuenciación de tareas aparecen de forma frecuente en disímiles situaciones de la vida real, siempre que sea necesario asignar recursos a la realización de tareas en espacios de tiempo, optimizando una o más funciones objetivo. En dependencia del problema a resolver dichas tareas pueden tomar diferentes formas, y los objetivos también pueden variar. En esta investigación se aborda la secuenciación de tareas en ambientes de manufactura, donde es necesario secuenciar los reportes solicitados por los clientes en un conjunto de máquinas con restricciones de capacidad. Adicionalmente aparecen otra serie de limitaciones que es necesario tener en cuenta al construir una solución factible, lo cual hace que el sistema utilizado actualmente no resuelva de manera eficiente el problema de secuenciación de reportes, teniendo en cuenta todas las restricciones que presenta la empresa.

Para resolver el problema en cuestión se propone un algoritmo general, el cual inicialmente distribuye la capacidad total del sistema entre los recursos existentes, teniendo en cuenta las capacidades que presentan los mismos, siguiendo una de cuatro posibles alternativas de selección. Posteriormente cada recurso selecciona en qué orden procesará los reportes que tiene asignados.

El estudio experimental desarrollado muestra que las cuatro alternativas implementadas permiten obtener soluciones factibles para el problema de secuenciación de reportes planteado, siendo la alternativa basada en Q-Learning la que mejores resultados obtiene.

Abstract

Scheduling problems appear on a regular basis in many real life situations, whenever it is necessary to allocate resources to perform tasks, optimizing one or more objective functions. Depending on the problem being solved, these tasks can take different forms, and the objectives can also vary. This research addresses scheduling in manufacturing environments, where it is necessary to sequence the reports requested by customers in a set of machines with capacity constraints. Additionally, there is another set of limitations that must be taken into account when constructing a feasible solution, which means that the system currently used does not efficiently solve the report scheduling problem, taking into account all the constraints of the company.

To solve this problem, a general algorithm is proposed, which initially distributes the total capacity of the system among the existing resources, taking into account the capacity presented by teach hem, and following one of four possible selection alternatives. Subsequently, each resource decides in which order it will process the reports assigned to it.

The experimental study performed shows that the four alternatives implemented allow to obtain feasible solutions for the report scheduling problem, being the alternative based on Q-Learning the one that obtains the best results.

Tabla de contenidos

Introducción

La secuenciación de tareas o reportes (*scheduling*), de manera general, puede definirse como la asignación de recursos en un lapso de tiempo para realizar un conjunto de operaciones (Pinedo 1995), o bien como resolver el problema de encontrar la asignación óptima de ciertos recursos a determinadas tareas (Lawler et al. 1993). Por tanto, en un problema de secuenciación siempre existirán tres componentes importantes: los trabajos que se deben realizar, los recursos disponibles para su realización (máquinas), y las finalidades o función objetivo que se desea optimizar y que nos permite identificar, entre varias secuenciaciones posibles, aquellas que estén más cercanas al óptimo (Alberto et al. 2008).

Los problemas de secuenciación se pueden desarrollar en ambientes *offline* (determinísticos) o ambientes *online* (estocásticos). Según (Vredeveld 2012), en ambientes *offline* todos los datos relevantes sobre el problema se conocen de antemano, pero en un problema del mundo real no siempre ocurre así. Por lo tanto, en muchos escenarios se debe encontrar una buena secuencia cuando los datos están incompletos y se deben tomar decisiones basadas en la falta de conocimiento. Un enfoque que puede hacer frente a esta incertidumbre es la programación estocástica.

La mayoría de las investigaciones en el área de la secuenciación de tareas se han enfocado en el desarrollo de procedimientos exactos para la generación de una solución base, asumiendo que se cuenta con toda la información necesaria y, por tanto, se está en presencia de un ambiente determinístico (Herroelen & Leus 2005). Sin embargo, en problemas de secuenciación de la vida real el ambiente es tan dinámico que es imprescindible contar con métodos que puedan hacer frente a la ocurrencia de eventos inesperados como roturas de las máquinas, operaciones que demoran más de lo planificado, etc.

Otro de los problemas que ha sido identificado por los investigadores en este tema es el hecho de que muchas de las investigaciones se concentran en problemas de optimización que constituyen una versión simplificada de la realidad. Esto permite el uso de enfoques sofisticados y garantiza en muchos casos la obtención de soluciones óptimas, sin embargo, la exclusión de restricciones del mundo real daña la aplicabilidad de dichos métodos. Lo que la industria necesita son sistemas que se ajusten exactamente

a las condiciones de la planta de producción y que generen soluciones en poco tiempo (Urlings et al. 2010).

Existen tres tipos fundamentales de algoritmos o enfoques para resolver problemas de secuenciación de tareas, los exactos, los de aproximación y los heurísticos. Dentro de los primeros podemos encontrar, por ejemplo, la Programación Lineal (Medina P. & Cruz E. 2008; Ramezanian et al. 2010; Özgüven et al. 2012) y la técnica de Ramas y Cotas. El uso de heurísticas y metaheurísticas se vuelve más factible cuando la dimensión del problema aumenta. Dentro de estas últimas las más utilizadas han sido los Algoritmos Genéticos (Bertel & Billaut 2004; Wang et al. 2006; Najarro et al. 2017), los métodos de optimización basada en Colonias de Hormigas (Puris et al. 2007; Suarez & Martinez 2008) y Enjambre de Partículas (Tu et al. 2006; Ge et al. 2006), las Reglas de Despacho (Nhu Binh & Joc Cing 2005), el Recocido Simulado (Varadharajan & Rajendran 2005; Alvarez et al. 2008; Khan & Govindan 2011;), la búsqueda Tabú (Abiri et al. 2009; Bozejko et al. 2013), entre otras.

Las heurísticas pueden ser clasificadas en dos categorías, las heurísticas de mejora o las de construcción, las primeras comienzan con una secuencia y la van mejorando haciendo cambios en aras de encontrar una secuencia similar que sea mejor, las constructivas adicionan un trabajo a la vez hasta encontrar una solución factible. Las Reglas de Despacho son un ejemplo de heurística constructiva, las cuales establecen prioridades entre los trabajos y cada vez que una máquina o recurso se libera, inspecciona los trabajos en espera y selecciona aquel que tenga la mayor prioridad.

Para resolver problemas donde está implicado de una forma u otra la permutación y se cuenta con dos máquinas, el clásico algoritmo de Johnson (Johnson 1954) obtiene la solución óptima cuando se tiene como objetivo la minimización del tiempo que se necesita para procesar la totalidad de los trabajos. Cuando se tienen tres o más máquinas se han desarrollado métodos heurísticos constructivos que obtienen soluciones factibles, entre los que se destacan las conocidas heurísticas de Palmer, Gupta y NEH, diseñadas con el objetivo de minimizar el tiempo de completamiento de las tareas. En el caso de NEH se construye una secuencia agregando trabajos de una lista (ordenados teniendo en cuenta el tiempo de procesamiento) en forma sucesiva y evaluando múltiples inserciones de éstos en las secuencias parcialmente construidas, esta heurística ha obtenido buenos resultados para este problema, existiendo un algoritmo eficiente desarrollado por (Taillard 1990).

Otro ejemplo de enfoque constructivo es el aprendizaje reforzado, donde un "agente" debe aprender que "acciones" realizar cuando se encuentra en cierto "estado", de modo tal que se maximice una recompensa. El agente interactúa con un entorno definido, y las acciones que este realiza modifican dicho entorno (Sherstov & Stone 2005). La mayoría de los algoritmos de aprendizaje reforzado se basan en la aproximación de una función, la cual, dependiendo del enfoque, es un indicador de cuan bueno es estar en cierto estado o de cuan conveniente es estar en cierto estado y ejecutar cierta acción. La principal diferencia del aprendizaje reforzado con otros tipos de aprendizaje es que usa como información para el entrenamiento la evaluación de las acciones y el resultado de estas en el entorno, en vez de instruir al agente diciéndole que acción tomar (Recabal Guiraldes 2009).

El presente trabajo se centra en resolver un problema que se corresponde con un ambiente de máquinas paralelas idénticas, donde existen diferentes recursos (cada uno tiene una capacidad diferente) para procesar los reportes solicitados por los clientes y cada reporte puede ser ejecutado por una o dos máquinas simultáneamente. Es decir, que al construirse una solución para este problema, debe tenerse en cuenta que cada máquina tiene una cantidad de trabajos a procesar y el orden de los mismos debe determinarse teniendo en cuenta el objetivo a optimizar.

En esta investigación particularmente, en el proceso de asignación de reportes se llega a un problema de optimización con múltiples restricciones en el cual se debe tener en cuenta la cantidad de recursos, la disponibilidad de los mismos, las características de los reportes en cuanto a la cantidad de máquinas que necesitan para ser procesados, la prioridad que tienen asociada, los tiempos de completamiento de las operaciones, entre otras. A partir de esto, es posible construir algoritmos y métodos computacionales de una manera rápida y flexible con técnicas existentes en la literatura.

Debido a lo antes expuesto surge como **problema de investigación:** La no existencia de un algoritmo para optimizar el proceso de secuenciación de reportes teniendo en cuenta todas las restricciones del problema.

Para contribuir a la solución del problema científico antes plateado, se formuló la siguiente **hipótesis de investigación**:

El Aprendizaje Reforzado aplicado a problemas de secuenciación de reportes brinda mejores resultados en cuanto al tiempo final de procesamiento de los trabajos que enfoques basados en aleatoriedad, las reglas de despacho y la heurística NEH.

Para dar solución al problema de investigación se plantea como **objetivo general** de esta tesis: Desarrollar un sistema para la optimización del proceso de secuenciación de reportes con múltiples restricciones.

Este objetivo se desglosa en los siguientes **objetivos específicos**:

- Analizar los diferentes enfoques reportados en la literatura para resolver problemas de secuenciación de reportes.
- Desarrollar un algoritmo para la optimización del proceso de secuenciación de reportes utilizando como alternativas de solución para la distribución por máquinas el Aprendizaje Reforzado, la Pseudo- Aleatoriedad, la regla de despacho MWKR y la heurística NEH.
- Desarrollar un sistema de fácil interacción con los usuarios que incorpore todas las alternativas diseñadas.
- Evaluar mediante pruebas estadísticas y datos reales el sistema desarrollado.

La tesis está estructurada en tres capítulos. En el capítulo uno se realiza un estudio de los problemas de secuenciación de tareas enfocado a la industria, destacando sus distintas clasificaciones, tipos de soluciones y ambientes en que se desarrollan. También se presenta la notación más utilizada en estos tipos de problemas, así como los métodos de solución más empleados en la industria. En el segundo capítulo se propone un algoritmo para optimizar el proceso de secuenciación de reportes a través de un sistema donde el usuario puede ver diferentes soluciones basadas en cuatro técnicas existentes en la literatura (Pseudo-Aleatorio, Reglas de Despacho, NEH, Aprendizaje Reforzado). Más adelante, en el capítulo tres, se realizan pruebas estadísticas para evaluar el comportamiento de las alternativas implementadas a través de diferentes configuraciones en la simulación basada en un caso de estudio relacionado con un supermercado belga. Por último, se plantean las conclusiones y las recomendaciones.

Capítulo 1: Problemas de secuenciación de tareas en ambientes de manufactura

La secuenciación de tareas consiste en la asignación de entidades (personas, tareas, vehículos, conferencias, exámenes, reuniones, etc.) a un recurso en el espacio de tiempo, de manera tal que las restricciones impuestas sean satisfechas y ciertas metas sean logradas, es decir, un plan para ejecutar un trabajo o alcanzar un objetivo, especificando el orden y el tiempo asignado para cada parte. De manera general se ha definido la secuenciación de tareas (*scheduling*) como la asignación de recursos (máquinas) en un lapso de tiempo para realizar un conjunto de trabajos (Pinedo 1995), o cómo resolver el problema de encontrar la asignación óptima de ciertos recursos a determinadas tareas (Lawler et al. 1993).

Actualmente en la industria, la planificación de una secuencia de tareas para desarrollar actividades en un orden y tiempo determinados juega un papel primordial para conseguir el éxito, a través de la mejora de la eficiencia y la efectividad de las operaciones. Generalmente los problemas de secuenciación son clasificados como NP-Completos (Garey et al. 1976), y el tiempo de cómputo se incrementa exponencialmente de acuerdo al tamaño del problema, siendo los problemas de secuenciación en ambientes de manufactura uno de los más complejos de resolver (Shen 2002).

1.1 Problemas de Secuenciación de Tareas en la Industria

En los procesos de manufactura o industria, la secuenciación se define como un proceso de optimización que asigna recursos limitados en el tiempo a un conjunto de actividades que se pueden ejecutar en paralelo. Esta asignación tiene que cumplir un conjunto de restricciones que reflejan las relaciones temporales entre las actividades y las limitaciones de capacidad del conjunto de recursos compartidos (Wang & Shen 2007). A lo largo de los años, los problemas de la secuenciación en ambientes de producción han capturado el interés de muchos investigadores en numerosas comunidades de investigación, por ejemplo, Ingeniería Industrial, Investigación de Operaciones e Inteligencia Artificial, entre otras. La creciente complejidad del proceso de manufactura y la fuerte competencia en el mercado obliga a las empresas a optimizar sus operaciones tanto como les sea posible, pues de ello depende, en gran medida, la rentabilidad que puedan tener (Grossmann 2005).

En (Neto et al. 2015; Pinedo 2008) se expresa que los problemas de secuenciación de tareas constituyen un tipo de problemas de decisión que juega un rol crucial en fábricas manufactureras y en industrias de prestación de servicios, donde determinadas tareas deben ser asignadas y ejecutadas por algún recurso teniendo como objetivo la optimización de uno o varios criterios.

Según (Vukovic et al. 2017), la planificación es anterior a cualquier otra función directiva. La eficiencia del plan de secuenciación refleja el nivel de logro del propósito y de los objetivos del negocio. Simultáneamente, la eficiencia del plan implica una relación costo-efectividad con respecto al logro de los objetivos, frente a los gastos y otros factores necesarios para su realización. La secuenciación de tareas en los procesos de la industria está relacionada con la utilización adecuada de los recursos de un sistema de fabricación con el objetivo de cumplir razonablemente con las demandas de productos destinadas a los clientes finales (Hmida et al. 2014).

Un ejemplo de secuenciación en procesos de manufactura se ve reflejado en (Jungwattanakit et al. 2008), donde se le da solución a un problema de fabricación textil, en el cual se elaboran productos que contienen fibras, como ropa, neumáticos e hilo. Esta fábrica se caracteriza por instalaciones de flujo de producción de etapas múltiples, donde una etapa de producción puede estar compuesta por líneas de producción paralelas. En algunas etapas las instalaciones (máquinas, líneas, etc.) son duplicadas y funcionan en paralelo con el fin de aumentar las capacidades generales del taller o para equilibrar las capacidades de las etapas. Esto posibilita además eliminar o reducir el impacto de etapas de congestión en la capacidad general de la planta.

Otro problema que se puede encontrar en la industria es el planteado en (Yang et al. 2016). En el mismo se analiza el problema de secuenciación robótico flexible en celdas de manufactura con múltiples robots, una celda de manufactura no es más que un conjunto de máquinas que están agrupadas por el tipo de producto que pueden producir en un entorno de fabricación. En este ambiente se analiza el movimiento de los brazos robóticos entre las estaciones de carga, las máquinas y las estaciones de descarga, y se propone una estrategia de programación multi-robot para minimizar la operación de desplazamiento al insertar las tareas de transportación.

Por otra parte, (Sivakumar 1999) propone un sistema de optimización y planificación dinámica basado en simulación de eventos discretos en ambientes *online*, para optimizar el tiempo de ciclo y la utilización de recursos en el complejo entorno de fabricación de

semiconductores de prueba. El impacto del sistema incluye el logro de un ciclo de tiempo eficiente, una mejor utilización de las máquinas, reducción en el tiempo de planificación y un rendimiento de fabricación más predecible. Además, permite a los gerentes y planificadores sin experiencia llevar a cabo un análisis de las posibles interrupciones o inconvenientes que puedan ocurrir en un futuro.

En (Staeblein & Aoki 2014) se ofrecen ideas para investigadores y gerentes sobre el diseño y la administración de sistemas de cumplimiento de pedidos automotrices. El objetivo de las mismas es mejorar su capacidad de hacer planes y cronogramas relacionados con las demandas y pedidos detallados de clientes a través de la utilización de técnicas basadas en la secuenciación de tareas. También en la industria químico-farmacéutica se han realizado investigaciones en este sentido, con el fin de proponer soluciones que permitan obtener mejores resultados productivos. En (Moniz et al. 2014) se define una metodología a través de un acercamiento integrado que considera simultáneamente la representación del problema de planificación, el modelo de optimización y el proceso de decisión-fabricación.

La Figura 1.1 muestra algunos de los factores que influyen en un problema de secuenciación, estos factores no necesariamente aparecen al mismo tiempo.

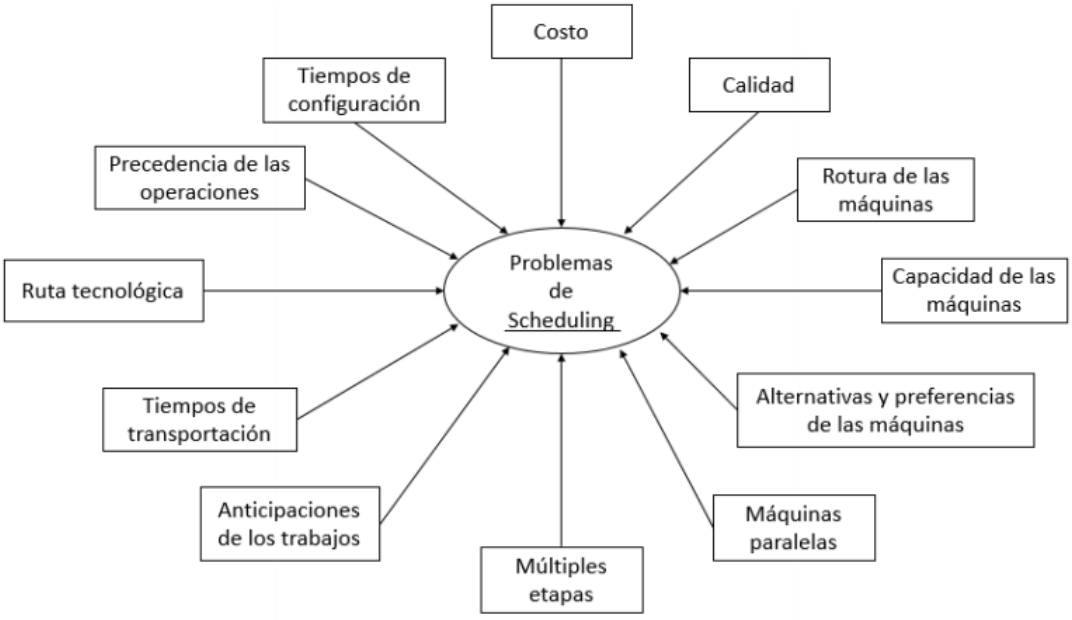

Fig. 1.1 Factores que pueden influir en un problema de secuenciación de tareas.

1.2 Clasificación de los problemas de secuenciación de tareas

Los problemas de secuenciación pueden clasificarse teniendo en cuenta el ambiente de las máquinas, las características de los trabajos y la función objetivo. Esta clasificación se conoce comúnmente como $\alpha|\beta|\gamma$ (Graham et al. 1979), donde α representa el ambiente de las máquinas, β las características de los trabajos y γ el criterio a optimizar.

1.2.1 Ambiente de las máquinas (α)

Según el ambiente de las máquinas el escenario más simple es el que tiene solo un recurso, pues los trabajos tienen una sola operación a ser procesada y solo existe una máquina que pueda ejecutarla. Cuando existen múltiples máquinas el entorno se torna más complicado, ya que pueden ser idénticas o pueden diferir en velocidad. Los posibles ambientes con máquinas paralelas se resumen de la siguiente forma (Martínez Jiménez 2012):

- Máquinas Paralelas Idénticas: Procesan los trabajos a la misma velocidad.
- Máquinas Paralelas Diferentes: El tiempo de procesamiento depende de la máquina.
- Máquinas Paralelas no relacionadas: El tiempo de procesamiento depende de la máquina y del trabajo.

Cuando un trabajo tiene un número fijo de operaciones que requieren diferentes máquinas se dice que se está en presencia de un problema tipo 'shop', y de acuerdo a las restricciones que el mismo presente se puede clasificar en:

- *Open Shop*: Existen 'm' máquinas y cada trabajo tiene que ser procesado en cada una de ellas. No existen restricciones de orden, lo que implica que cada trabajo puede pasar por las máquinas siguiendo rutas distintas (Yang et al. 2016), (Brasel et al. 2009).

- *Job Shop*: Los trabajos pasan por cada máquina sólo una vez, existen restricciones de orden, sin embargo, el orden no es el mismo para todos los trabajos (Fonseca Reyna et al. 2015), (El-Bouri et al. 2007), (Márquez Delgado et al. 2012), (Puris et al. 2007), (Mendez-Hernandez et al. 2017).

- *Flow Shop*: Todos los trabajos pasan por todas las máquinas en el mismo orden, es decir, la primera tarea de todos los trabajos se debe ejecutar en la máquina 1, la segunda tarea en la máquina 2, y así sucesivamente (Fonseca Reyna & Márquez

Delgado 2012), (Fonseca Reyna et al. 2015), (Yenisey & Yagmahan 2014), (Agarwal et al. 2006), (Zobolas et al. 2009), (Wang et al. 2006), (Rajendran & Ziegler 2004).

- *Job Shop_Flexible*: Es una combinación del *Job Shop* y el ambiente de máquinas no relacionadas. Cada operación puede ser procesada por cualquier máquina de un conjunto determinado, y dichas máquinas pueden diferir en velocidad (Ho et al. 2007; Martínez et al. 2011; Alberto et al. 2008; Nhu Binh & Joc Cing 2005; Nhu Binh HO 2005).
- *Flow Shop* Híbrido: Extensión del *Flow Shop* donde las máquinas tienen tiempos de preparación y se produce en lotes (más de un trabajo a la vez) (Martínez Jiménez 2012).

1.2.2 Características de los trabajos (β)

El campo β puede reflejar múltiples características. Estas pueden ser:

- Tiempo de liberación (*release date*): todos los trabajos tienen asociado un tiempo, el cual indica el momento en que deben empezar a ejecutarse (Ruiz & Vázquez-Rodríguez 2010).
- Tiempo de terminación (*Due date*): todos los trabajos tienen asociado un tiempo, el cual indica el momento en que debe terminar su ejecución.
- Tiempos de configuración: significa la existencia de un tiempo de configuración dependiente de la secuencia para un trabajo cuando viene precedido por otro (Nagano et al. 2012), (Ruiz et al. 2008), (Martinez-Jimenez & Fonseca Reyna 2017), (Allahverdi 2015).
- Restricciones de precedencia: estas pueden estar presentes en un ambiente de máquina única o máquinas paralelas, teniendo en cuenta las restricciones de precedencia entre las operaciones de los trabajos (Ruiz et al. 2008).
- *Preemption*: indica que existen trabajos que pueden ser interrumpidos (una o varias veces) antes de ser terminados. Los mismos deben completarse posteriormente en la misma máquina o en otra distinta. Se especifica con la entrada *preempt* (Djellab & Djellab 2002).
- *Breakdowns*: las máquinas no están disponibles de forma continua. Se indica con la entrada brk (Adressi et al. 2016), (Alisantoso et al. 2003).

- Recirculación: ocurre cuando un trabajo puede visitar una máquina en más de una ocasión, en problemas de tipo *Job Shop* (Bertel & Billaut 2004).

1.2.3 Criterio a optimizar (γ)

El criterio de optimización viene dado por una función objetivo de maximización o minimización en correspondencia con el objetivo que se desea alcanzar con la secuenciación final. Ejemplo de algunos de esos diversos objetivos que pueden ser perseguidos en los problemas de secuenciación son:

- *Flowtime,* suma de los tiempos de completamiento de los trabajos.
- *Lateness,* diferencia entre la fecha de terminación planificada y la de terminación real. Esta medida puede ser positiva (tardanza, *tardiness*) o negativa (anticipación, *earliness*).
- *Makespan,* máximo tiempo de completamiento de los trabajos, equivalente al tiempo en que el último trabajo abandona el sistema.

1.3 Tipos de soluciones y tipos de ambientes

Según lo reportado en la literatura, cualquier solución factible puede clasificarse en:

1. Sin-retardo: ningún recurso se mantiene inactivo si hay al menos una operación que pueda ser procesada.
2. Semi-activa: la única opción para que una operación comience antes es cambiar el orden de procesamiento en un recurso.
3. Activa: si no es posible construir otra solución a través de cambios en el orden de procesamiento en los recursos, con al menos una operación terminando más temprano y ninguna terminando más tarde.

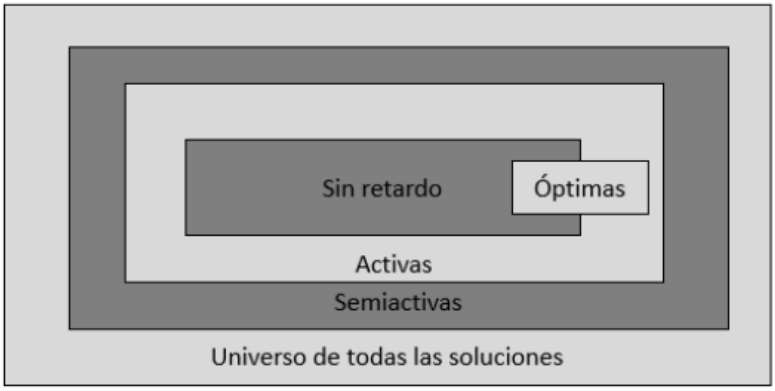

Fig. 1.2. Clasificación de los diferentes tipos de soluciones

El siguiente ejemplo ayuda a entender la diferencia entre estas tres categorías, en la figura se muestran tres posibles soluciones para el mismo problema de secuenciación, y se analiza a cuál de las categorías pertenece cada una.

Ejemplo: Asumamos que se está resolviendo un problema de secuenciación con dos trabajos y dos máquinas. Cada trabajo tiene dos operaciones y la información sobre qué máquina puede ejecutarlas se brinda en la siguiente imagen:

Job	Op_1	Op_2
1	M_2,2	M_1,1
2	M_1,2	M_2,1

Fig. 1.3. Problema de secuenciación con 2 máquinas y 2 trabajos

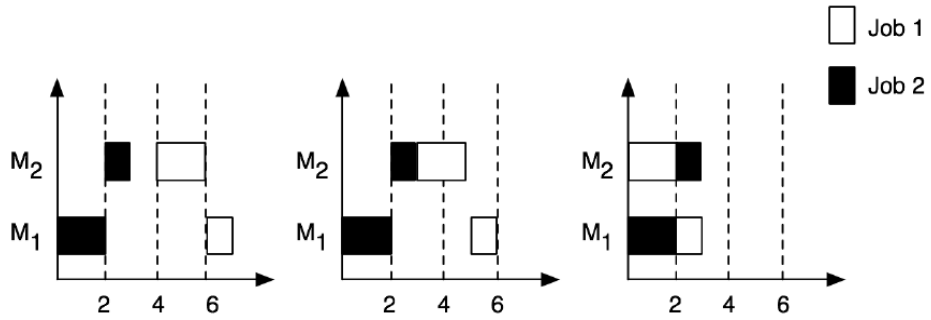

Fig. 1.4. Tres ejemplos de soluciones para el problema presentado en la Figura 1.3.

Analizando las soluciones presentadas en la Figura 1.4 es posible ver que la secuencia 1 (a la izquierda) es una secuencia factible, porque las restricciones de orden del problema son satisfechas, pero no pertenece a ninguna de las categorías descritas anteriormente. Si miramos la máquina M_2 es posible ver que permanece inactiva mientras hay operaciones que puede ejecutar (por ejemplo, entre el tiempo 3 y 4), lo que quiere decir que no es una secuencia sin retardo.

Tampoco es una secuencia semi-activa, pues se puede efectuar un corrimiento a la izquierda en la primera operación del trabajo J_1, la cual se está ejecutando en M_2 en el minuto 4 cuando pudiera ejecutarse en el 3. Si ejecutamos corrimientos a la izquierda en la secuencia 1 entonces obtendremos la secuencia 2 (al centro), lo cual significa que esta es semi-activa. Pero ni ésta ni la secuencia 1 son activas, pues pueden efectuarse modificaciones en ellas que hagan que disminuya el tiempo total de completamiento, por ejemplo, la primera operación del trabajo J_1 puede moverse al tiempo inactivo que tiene al inicio la maquina M_2, sin modificar el tiempo de inicio de la segunda operación

de J_2 que se encuentra allí secuenciada. Si realizamos estos cambios entonces la secuencia resultante sería la 3 (a la derecha), la cual podemos clasificar en activa y sin retardo.

La mayoría de las investigaciones en el área de la secuenciación de tareas se han enfocado en el desarrollo de procedimientos exactos para la generación de una solución base asumiendo que se tiene toda la información necesaria y un ambiente determinístico (Herroelen & Leus 2005). Este enfoque es conocido en la literatura como problemas de secuenciación de tareas en ambientes de tipo offline o determinísticos.

Las secuenciaciones a menudo se generan por adelantado para dirigir las operaciones de producción y para apoyar otras actividades de planificación (Taylor et al. 2013). Desafortunadamente, el procesamiento de los trabajos puede estar expuesto a interrupciones constantes que llevan a una reprogramación para lograr un rendimiento óptimo. Este proceso debe ser capaz de manejar eventos imprevistos o urgentes como la llegada de nuevos trabajos al sistema que no se tenían en plan con antelación, es decir, que deben ser procesados estos trabajos de igual manera que los que se ya tenían en plan, estos acontecimientos se desarrollan en ambientes online o estocásticos.

1.3.1 Ambientes offline o determinísticos

Un ambiente se denomina determinístico cuando todos los datos del problema de planificación o secuenciación son conocidos a priori. Estos modelos son estudiados por la optimización combinatoria, una característica común a la mayoría de los problemas estudiados por la optimización combinatoria es que son relativamente “fáciles” de plantear, pero difíciles de modelar y consecuentemente, mucho más difíciles de resolver (Alcaide López de Pablo 1995).

Una extensión natural de los modelos de planificación determinístico, consiste en asumir que ciertos datos del problema varían aleatoriamente y, de esa forma, aparecen los problemas de Planificación Estocástica (Alcaide López de Pablo 1995).

1.3.2 Ambientes *online* o estocásticos

En problemas de secuenciación *online* una secuencia de trabajos $\sigma = J_1, J_2, \dots, J_n$ tiene que procesarse en un número determinado de máquinas. Los trabajos arriban al sistema uno a uno, y cuando un nuevo trabajo llega tiene que ser inmediatamente despachado

hacia una de las máquinas, sin tener conocimiento sobre trabajos futuros. El objetivo es optimizar una función objetivo dada (Coto Palacio et al. 2018).

Cuando un trabajo nuevo J_t, $1 \leq t \leq n$ arriba al sistema, su tiempo de procesamiento P_t es conocido con anterioridad. Cada trabajo tiene que asignarse inmediatamente a una de las máquinas, sin conocimiento de trabajos futuros, con el objetivo de optimizar la función objetivo definida según el problema a resolver. Los tiempos de ejecución de los trabajos son conocidos de manera aproximada, debido a que los usuarios están comúnmente forzados a estimar los requisitos de CPU de sus trabajos. Ocurre en diversas ocasiones en estos ambientes la prohibición de las preferencias (*preemption)*, pues las altas exigencias de memoria de los trabajos hacen que la preferencia sea demasiado costosa. El objetivo de obtener el menor *makespan* posible se traduce en conseguir una elevada utilización de las máquinas. Además de su relevancia práctica, el problema de Graham (Graham et al. 1979) es importante porque es la raíz de muchas variantes problémicas dónde, por ejemplo, la preferencia es permitida, existen restricciones de precedencia entre los trabajos, o las máquinas funcionan a velocidades diferentes.

Un estudio experimental de algoritmos para uno de los principales problemas de secuenciación *online* se expuso en (Albers & Schröder 2001). Este problema es conocido como el problema de Graham (Graham et al. 1979) y se ha investigado solamente desde un punto de vista teórico (Albers 1997; Bartal et al. 1995; Karger et al. 1996). En el problema de Graham, una secuencia de trabajos tiene que ser secuenciada en *m* máquinas paralelas idénticas.

Snyman y Bekker plantean que la secuenciación *online* es un proceso de toma de decisiones en tiempo real, que intenta abordar las deficiencias de los enfoques tradicionales al realizar la secuenciación al mismo tiempo que el proceso de producción. Plantean además que la simulación por computadora se usa a menudo para ayudar con la programación, especialmente de procesos estocásticos complejos, discretos, dinámicos (Snyman & Bekker 2017).

En (Mao et al. 1995) se estudia el problema de secuenciación en una sola máquina, donde se definen dos algoritmos *online*: *First-Come-First-Served* (FCFS) y *Shortest-Available-Job-First* (SAJF). En los dos algoritmos se conserva una cola que contiene todos los trabajos que arriban, pero no se han ejecutado. En SAJF, los trabajos en cola están ordenados teniendo en cuenta solo el P_j (tiempo de procesamiento), mientras que

en FCFS, los trabajos en cola están listados de forma creciente según las fechas de liberación r_j (los trabajos con el mismo r_j son ordenados teniendo en cuenta el P_j). Cuando la máquina se vacía después del procesamiento de un trabajo, el primero en la cola es asignado a la máquina para su ejecución. Cuando un nuevo trabajo llega, se coloca en su posición correspondiente en la cola. FCFS y SAJF son algoritmos *online* ya que se toman decisiones solo basadas en la información sobre la cola disponible. Además, son conservadores, lo cual significa que la máquina no está nunca desocupada cuándo hay trabajos esperando. Aunque estos dos algoritmos son comúnmente utilizados, pocos resultados analíticos están disponibles.

Existen diversos estudios recientes desarrollados en ambientes *online,* uno de ellos es expuesto en (Palacio et al. 2017), donde se propone una solución al problema de secuenciación de tareas en ambientes *online* expuesto en (Peeters 2008), el cual se basa en la planta química de producción de (Castillo & Roberts 2001). En esta planta se producen dos tipos de productos y se cuenta con una serie de una o más etapas procesadoras con unidades paralelas de procesamiento (máquinas) en cada etapa. El orden de llegada de cada producto y el tiempo de permanencia en cada máquina es generado con una distribución exponencial con determinada media. La solución propuesta utiliza dos algoritmos basados en Aprendizaje Reforzado (*Q-Learning y Learning Automata*).

1.4 Notación empleada en los problemas de secuenciación

Los problemas de secuenciación de tareas presentan su propia notación proveniente de la literatura, a continuación, se muestra una lista de algunas de las que serán usadas en esta tesis:

m	Número de máquinas
n	Número de trabajos
M_i	Máquina *i*, donde $i = \{1, \dots, m\}$
J_j	Trabajo *j*, donde $j = \{1, \dots, n\}$
C_j	Tiempo de terminación del trabajo *j*
P_j	Tiempo de procesamiento del trabajo *j*
o_{ij}	La *i*-ésima operación del trabajo *j*

s_{ij}	Tiempo de inicio de la operación *i*, trabajo *j*
c_{ij}	Tiempo de terminación de la operación *i*, trabajo *j*
p_{ij}	Tiempo de procesamiento de la operación *i*, del trabajo *j*
r_j	Fecha de inicio del trabajo *j*
d_j	Fecha de vencimiento del trabajo *j*
C_{max}	El *makespan*, máximo C_j entre todos los trabajos, $max\{C_1, \dots, C_n\}$

En la Figura 1.5 muestra un diagrama de tiempo de la operación o_{ij} donde se resume la notación antes mencionada.

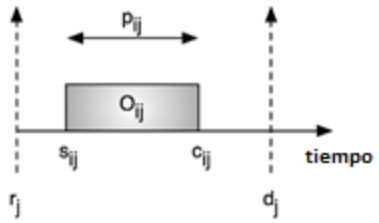

Fig. 1.5 Diagrama de tiempos de una operación

1.5 Métodos de solución de problemas de secuenciación de tareas

Los métodos para resolver problemas de secuenciación provienen de las técnicas de la optimización combinatoria, tanto de los métodos aproximados (algoritmos voraces, búsqueda local, algoritmos genéticos, etc.) como de los métodos exactos (programación dinámica, ramas y cotas, satisfacción de restricciones, etc.). El enfoque clásico de solución consiste en construir un modelo matemático, generalmente basado en Programación Mixta en Enteros, y luego aplicar un algoritmo de búsqueda como el algoritmo de Ramificación y Acotamiento para encontrar la solución óptima (Mendez et al. 2006). Sin embargo, a medida que el número de recursos disponibles en el problema se incrementa, o el número de operaciones a planificar crece, este enfoque no será capaz de encontrar la solución óptima en un tiempo computacional razonable (Ruiz et al. 2008). Por consiguiente, los métodos heurísticos se convierten en el centro de atención, pues son capaces de encontrar buenas soluciones de manera eficiente en un tiempo computacional adecuado (Zhang 1996).

Los problemas de secuenciación de tareas en la industria o manufactura no son una excepción, es posible encontrar en la literatura métodos para su solución, tanto exactos como heurísticos. La programación entera mixta es uno de los métodos exactos utilizados, un ejemplo de esto se ve reflejado en (Moniz et al. 2014), en el cual se desarrolla un modelo de tiempo discreto basado en uno propuesto en (Moniz et al. 2013). Este modelo es aplicado en la solución de un problema real de la industria químico-farmacéutica y su principal ventaja es la inclusión de una representación general del proceso que permite utilizarse por varios departamentos de la empresa.

La propuesta planteada en (Stefansson & Shah 2005) da solución a un problema del mundo real que se origina en una planta de producción farmacéutica compleja, donde los clientes exigen constantemente precios bajos, así como niveles de servicio altos y un nivel de flexibilidad en sus demandas. El acercamiento que proponen se basa en un *framework* con un horizonte jerárquicamente estructurado y cambiante, el cual recibe el fragmento de datos de entrada y tiene que tomar las decisiones con sólo un conocimiento parcial de la entrada requerida.

En problemas de secuenciación de tareas, la diferenciación de los trabajos y el proceso de secuenciarlos tienen un impacto directo en el desempeño de la heurística. Son varias las reglas y los algoritmos de secuenciación que se han desarrollado basados en estos criterios, sin embargo, el cómo dar prioridad a los trabajos sigue siendo un desafío en problemas de secuenciación de tareas en la industria.

Para resolver estos problemas también se incluyen metaheurísticas y métodos de aprendizaje. Por ejemplo, en (Najarro et al. 2017) se analiza el efecto de la inclusión de varias restricciones que influyen negativamente en la programación de la producción en un ambiente de manufactura real, y se introduce un eficiente Algoritmo Genético combinado con una búsqueda local de vecindad para problemas de n tareas y m máquinas. El objetivo perseguido es minimizar el tiempo de completamiento total de las tareas o *makespan*. Los experimentos computacionales realizados sobre un conjunto de ejemplos de problemas de diferentes tamaños demuestran que la metaheurística híbrida propuesta alcanza soluciones de alta calidad comparables con los óptimos reportados.

Otra tipo de modelación es la utilizada en (Snyman & Bekker 2017), donde se desarrolla un prototipo de un sistema de simulación en tiempo real como herramienta de apoyo a la decisión para la reprogramación en tiempo real de los pasos que ejecutan las máquinas en un ambiente de tipo *Job Shop*. El sistema utiliza una red sensorial,

dispositivos móviles y computación en la nube de conjunto con métodos de simulación y secuenciación. También se crea un modelo de simulación para describir el entorno y las operaciones de dicho ambiente.

Una solución a un problema de secuenciación ampliamente reconocido en la industria fue expuesta en (Moreira da Silva et al. 2014), donde se aplican métodos matemáticos para resolverlo a través de un algoritmo genético y la heurística *greedy*. El problema abordado se presenta en un entorno industrial real, específicamente en la industria del pan, donde las tasas de producción mostraban un incremento anual y la necesidad de una planificación optimizada cobraba importancia para satisfacer las demandas. Aquí se consideraban de forma simultánea la producción paralela, los tiempos de configuración de las máquinas, la producción por lotes y el cumplimiento de las fechas de entrega.

En (Glass et al. 1994) los autores aplican metaheurísticas en la solución de un problema de secuenciación de tareas para minimizar el *makespan* o tiempo de completamiento, reportando que el algoritmo genético no obtiene buenos resultados por sí solo, pero cuando se le incorpora un método de descenso entonces los resultados son comparables con los del recocido simulado. Otros resultados se presentan en (Moonsri et al. 2015), (Najarro et al. 2017) y (Rodriguez et al. 2013). Esta última se basa en una heurística *greedy* iterada para generar una secuencia de soluciones al iterar sobre una heurística constructiva utilizando las fases de destrucción y construcción. El modelo *greedy* iterado es propuesto para instancias de gran tamaño.

Por otra parte, en (Fonseca Reyna & Márquez Delgado 2012) se propone un algoritmo genético para el caso de *m* máquinas. En esta investigación los autores realizan un estudio de los parámetros principales de esta metaheurística con el objetivo de obtener la mejor combinación de ellos en la solución del *Flow Shop*. Para comprobar el rendimiento de este algoritmo, los autores, una vez determinada la mejor combinación de parámetros, realizan una comparación con los resultados óptimos reportados en la literatura por (Taillard 1993), demostrando que el algoritmo alcanza en la mayoría de los problemas estos valores.

1.5.1 Heurística NEH

La heurística NEH (propuesta por Nawaz, Enscore y Ham) (Nawaz et al. 1983) ha sido identificada como la mejor heurística constructiva para resolver problemas de tipo *Flow Shop*. El criterio seguido por la misma es ordenar los trabajos en orden decreciente y

secuenciarlos de forma tal que se vaya minimizando la solución parcialmente construida hasta ese momento. Esta heurística ha servido como base para el desarrollo de otras propuestas (Kalczynski & Kamburowski 2007; Singhal et al. 2012; Fernandez-viagas et al. 2015; G & Rajaram 2016; Liu et al. 2017).

La idea básica del algoritmo NEH (Nawaz Enscore Ham) es secuenciar inicialmente de forma óptima los dos trabajos de mayor tiempo de procesamiento usando la regla de Johnson (Johnson 1954). Luego se introducen los trabajos restantes en orden decreciente tomando en cuenta sus tiempos totales de procesamiento en uno de los espacios dentro de los trabajos ya secuenciados de forma tal que el *makespan* total se minimice en cada caso.

La heurística NEH es una heurística constructiva que sigue los siguientes pasos (Nawaz et al. 1983):

Paso 1: Ordenar los *n* trabajos en orden decreciente de acuerdo a su tiempo de procesamiento.

Paso 2: Tomar los primeros dos trabajos y secuenciarlos de forma tal que se minimice el *makespan* parcial, como si fuesen los únicos dos trabajos existentes.

Paso 3: Para $k= 3$ hasta *n* ejecutar el Paso 4

Paso 4: Insertar el k-ésimo trabajo en el lugar que minimice el *makespan* parcial, entre los k posibles lugares de inserción.

Número total de secuencias a ser analizadas: $n(n+1)/2 - 1$

1.5.2 Reglas de Despacho

Como se mencionó anteriormente, al ser los problemas de secuenciación de tareas de tipo NP-completos y su tamaño generalmente grande, los métodos de optimización exactos fallan en proveer soluciones óptimas en un tiempo razonable y comienzan a utilizarse en mayor medida los métodos heurísticos. Entre las heurísticas utilizadas uno de los enfoques más sencillos son las llamadas reglas de despacho. Una regla de despacho es una estrategia de secuenciación en la cual se le asigna una prioridad a cada trabajo que espera ejecutarse en una máquina específica. Siempre que una máquina está disponible, una regla de despacho basada en prioridad examina los trabajos en espera y selecciona el de prioridad más alta para procesarse a continuación (Nhu Binh & Joc Cing 2005).

Se ha probado que las reglas de despacho obtienen buenas soluciones por si solas, pero no siempre óptimas (Shahzad & Mebarki 2016; Al-Turki et al. 2014; Zahmani et al. 2015). Una variante sería, por ejemplo, combinarlas con algoritmos de aprendizaje, utilizándolas en la función de recompensa para indicar la calidad de la acción escogida por un agente. Existen diversos criterios a seguir para obtener prioridades utilizando reglas de despacho, por lo que en este epígrafe se abordarán con mayor profundidad algunos de ellos.

Algunas de las reglas de despacho más utilizadas son (Pinedo 2008):

- Menor tiempo de Procesamiento (*Shortest Processing Time*, SPT): La prioridad más alta se le asigna a la operación en espera con el menor tiempo de procesamiento.
- Primero en entrar primero en salir (*First In First Out*, FIFO): La operación que arriba primero a la cola recibe la prioridad más alta.
- Mayor tiempo de procesamiento restante (*Most Work Remaining*, MWKR): La prioridad más alta se le da a la operación que pertenece al trabajo con el mayor tiempo de procesamiento restante.
- Fecha de terminación más próxima (*Earliest Due Date*, EDD): El trabajo con la fecha de terminación más próxima es procesado primero.

En algunos trabajos se introduce el uso de reglas de despacho en ambientes *online*, por ejemplo, en (Al-Turki et al. 2014) los autores enfocan el problema en presencia de trabajos con tiempo de procesamiento estocástico y cuya entrada al sistema ocurre de forma aleatoria siguiendo distribuciones específicas.

La regla *SPT* es un ejemplo de regla basada en tiempo de procesamiento, estas reglas ignoran la información sobre la fecha de vencimiento de los trabajos. Se ha demostrado que la regla *SPT* minimiza la media del *flowtime* y también se ha observado que tiene un buen desempeño cuando el objetivo a optimizar es la tardanza bajo condiciones de alta carga a procesar (Conway 1965; Rochette & Sadowski 1976; Blackstone et al. 1982; Haupt 1989)

Un ejemplo de una regla basada en fecha de vencimiento es *EDD*. En general, las reglas basadas en fechas de vencimiento dan buenos resultados bajo condiciones ligeras de carga, y su desempeño se deteriora en caso contrario (Ramasesh 1990). Las reglas combinadas hacen uso de las reglas de despacho mencionadas anteriormente, y asignan

las prioridades utilizando ambas informaciones, las fechas de vencimiento y el tiempo de procesamiento. Ejemplos de este tipo son: la regla *Least Slack*, *Critical Ratio*, etc. (Blackstone et al. 1982). Las reglas que no caen en ninguna de estas categorías asignan los trabajos en dependencia de las condiciones del ambiente de trabajo en vez de en las características de los mismos, un ejemplo de este tipo de regla es la denominada WINQ (Haupt 1989).

Cuando se utiliza esta regla los trabajos se ordenan de acuerdo a una estimación del tiempo que deben esperar antes de que el procesamiento en la próxima máquina pueda comenzar. Esta estimación incluye el tiempo necesitado por una máquina M_i para terminar su trabajo actual más la suma de los tiempos de procesamiento de todos los trabajos esperando actualmente para ser procesados por M_i. El trabajo que tenga la menor de estas sumas tiene la prioridad más alta.

Según lo revisado en la literatura, no existe un buen algoritmo de solución de problemas de secuenciación de tareas *online*, estocásticos y multi-etapa al mismo tiempo. En el caso de una etapa existe una muy buena heurística: *Weighted Shortest Expected Processing Time* (WSEPT) (Megow et al. 2006).

Esta heurística trabaja de la siguiente manera: las órdenes o trabajos llegan con el paso del tiempo y deben procesarse por una de las máquinas paralelas existentes que se encuentre desocupada. El objetivo es reducir el tiempo ponderado total de terminación ($\sum_j w_j C_j$, para todo trabajo j). Cada vez que una orden llega, la regla WSEPT envía el trabajo a la máquina que se espera que la termine de procesar primero. Con este fin, la regla encuesta cada máquina para saber su makespan actual esperado (incluyendo el trabajo nuevo). Si, por ejemplo, todos los trabajos tienen igual tiempo esperado de procesamiento, y cada máquina tiene la misma velocidad promedio, el makespan esperado es el largo de la cola (incluyendo el trabajo actualmente procesado, si existe).

1.5.3 Aprendizaje Reforzado

El Aprendizaje Reforzado o RL (terminología en inglés) se asocia con aprender qué hacer (cómo asociar situaciones con acciones) de forma tal que se maximice una señal numérica de recompensa. Es el problema encarado por un agente que debe aprender comportamientos a través de interacciones a ‘prueba y error’ con un ambiente dinámico. Un agente es un sistema computacional situado en un ambiente, que es capaz de actuar

de manera autónoma y flexible en dicho ambiente en aras de cumplir con su objetivo (Jennings et al. 1998).

Con cada acción realizada por un agente, el ambiente provee una recompensa o una penalización resultante de la acción tomada. Por ejemplo, al entrenar a un agente para jugar un juego, el ambiente podría proveer una recompensa positiva cuando el juego es ganado, una negativa cuando está perdido y cero en otros casos.

Un aprendiz o agente debe tomar sus propias decisiones sobre qué acciones escoger con el fin de obtener la mayor recompensa posible. Las acciones en cada momento pueden afectar no solo a la recompensa inmediata sino también a las situaciones siguientes y por tanto las recompensas futuras. Estas dos características, búsqueda a 'prueba y error' y recompensa retardada son dos de las características más significativas que distinguen el aprendizaje reforzado.

En el paradigma estándar del RL, un agente está conectado con su ambiente vía percepción y acción, como se muestra en la Figura 1.6. En cada interacción el agente percibe el estado actual *'s'* del ambiente y selecciona una acción '*a'* para cambiar el estado en que se encuentra. Esta transformación genera una señal '*r'* que es recibida por el agente, cuya tarea es aprender una política de selección de acciones en cada estado para recibir la mayor recompensa acumulada a largo plazo (Zhang 1996).

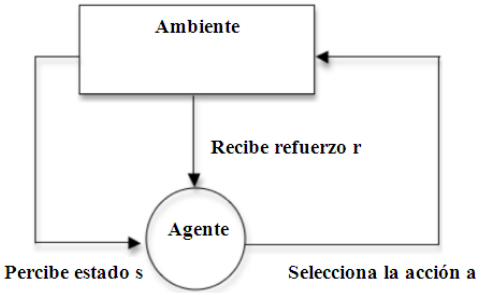

Fig. 1.6 Modelo estándar del aprendizaje reforzado

El modelo básico de RL consiste en:

- Un conjunto de estados del ambiente *S*
- Un conjunto de acciones *A*
- Un conjunto de "recompensas" en *R*
- Una función de transición *T*

Uno de los desafíos del RL es el intercambio entre la exploración y la explotación. Para obtener una recompensa alta, un agente RL debe preferir acciones que ha probado en el

pasado y ha encontrado que son efectivas a la hora de producir recompensa. Pero para descubrir tales acciones, tiene que probar otras que no ha seleccionado antes. El agente tiene que explotar lo que sabe para obtener gratificaciones o recompensas, pero también tiene que explorar para hacer mejor selección de acciones en el futuro. El dilema es que ni la exploración ni la explotación pueden perseguirse exclusivamente sin fallar en la tarea. Por eso, el agente debe analizar las acciones disponibles y progresivamente favorecer aquellas que parecen ser mejores.

Más allá del agente y del ambiente, se pueden identificar cuatro subelementos de un sistema de RL: la política, la función de recompensa, la función de valor y opcionalmente el modelo del ambiente.

La política define la forma del comportamiento del agente en un tiempo dado (Sutton & Barto 1998a), es decir, es la probabilidad de seleccionar una acción a en el estado s en un tiempo t. En cada instante t, el agente percibe su estado $s_t \in S$ y el conjunto de posibles acciones (s_t). Se elige una acción $a \in (s_t)$ y recibe del entorno el nuevo estado s_{t+1} y una recompensa r_{t+1}, esto significa que el agente implementa un mapeo de los estados a las probabilidades de selección de cada acción posible. La política del agente se denota π, donde (π_t) es la probabilidad de que $a_t = a$ si $s_t = s$.

La función de recompensa define la meta en un problema de RL, es decir, se asigna a cada par estado-acción del ambiente, una recompensa, lo que indica la conveniencia intrínseca de ese estado.

Mientras que la función de recompensa indica qué es bueno en sentido "inmediato", la función de valor representa qué es bueno a largo plazo. En términos generales, el valor de un estado es la cantidad total de recompensa que un agente puede acumular en el futuro, a partir de ese estado (Sutton & Barto 1998b).

El cuarto y último elemento de algunos sistemas de aprendizaje reforzado es el modelo del ambiente, el cual imita el comportamiento del ambiente. Por ejemplo, dado una condición y una acción, el modelo podría predecir el siguiente estado y la siguiente recompensa resultante. Los modelos son usados para la planificación, lo cual quiere decir cualquier forma de decidir el curso de la acción considerando las posibles situaciones futuras antes de que se presenten (Sutton & Barto 1998b).

El Aprendizaje Reforzado en la solución de problemas de secuenciación de tareas se ha utilizado como un enfoque capaz de proporcionar buenos resultados. Uno de los

primeros trabajos en los que se aplicó Aprendizaje Reforzado en problemas de secuenciación, específicamente de tipo *Job Shop*, fue presentado en 1995 (Zhang & Dietterich 1995) y posteriormente extendido en (Zhang 1996). El objetivo era secuenciar las diversas tareas que se deben realizar para instalar y probar las cargas útiles que se colocan en la bahía de carga de un transbordador espacial (para la NASA). Este estudio se refiere a una especificación típica de la tarea, que toma la forma de un problema de tipo *Job Shop.* Cada misión del transbordador es un trabajo que está vinculado a una fecha de lanzamiento fija. Cada trabajo consiste en un conjunto de tareas parcialmente ordenadas con requisitos de recursos. El objetivo es programar el conjunto de tareas para satisfacer un conjunto de restricciones temporales y de recursos, al mismo tiempo que se busca minimizar el margen de tiempo del cronograma.

Posteriormente, en (Gabel & Riedmiller 2007) y (Gabel 2009), los autores analizan y sugieren la aplicación de técnicas de aprendizaje reforzado para resolver problemas de secuenciación de tareas, ya que demuestran que modelar y resolver estos escenarios como problemas de aprendizaje con múltiples agentes es beneficioso para obtener soluciones cercanas a las óptimas y puede competir con enfoques alternativos de solución. Basándose en esta propuesta, en (Martínez Jiménez 2012) se propone un enfoque genérico Multi-Agente con Aprendizaje Reforzado para problemas de secuenciación, el cual modela y resuelve diferentes escenarios, enfocándose en encontrar soluciones robustas para los mismos usando enfoques basados en tiempos de inactividad.

Uno de los adelantos más importantes en el aprendizaje reforzado fue el de (Watkins & Dayan 1992) con el desarrollo del algoritmo Q-*Learning* (QL). Dicho algoritmo se basa en aprender una función acción-valor que brinda la utilidad esperada de tomar una acción determinada en un estado específico. El centro del algoritmo es una simple actualización de valores, cada par *(s, a)* tiene un Q-valor asociado, cuando la acción '*a'* es seleccionada mientras el agente está en el estado '*s,* el Q-valor para ese par estado-acción se actualiza basado en la recompensa recibida por el agente al tomar la acción. También se tiene en cuenta el mejor Q-valor para el próximo estado '*s*'. La regla de actualización completa es la siguiente:

$$Q\ (s,a) \leftarrow Q\ (s,a) + \alpha\,[r + \gamma\ maxQ\,(s',a') - Q\ (s,a)] \qquad (1.1)$$

En la expresión anterior, $\alpha \in [0; 1]$ representa la velocidad del aprendizaje y *r* la recompensa o penalización resultante de ejecutar la acción *a* en el estado *s*. La

velocidad de aprendizaje α determina el grado por el cual el valor anterior es actualizado. Normalmente se utiliza un valor pequeño para la velocidad de aprendizaje, por ejemplo, $\alpha = 0.1$. El factor de descuento (parámetro γ) toma un valor entre 0 y 1, si está cercano a 0 entonces el agente tiende a considerar sólo la recompensa inmediata, si está cercano a 1 el agente considerará la recompensa futura como más importante.

El seudocódigo del algoritmo se muestra en la Figura 1.7.

Inicializar $Q(s, a)$ arbitrariamente

Repetir (por cada episodio)

  Inicializar s

  Repetir (para cada paso del episodio)

    Escoger a de s usando una política derivada de Q (e.g., e-*greedy*)

    Tomar acción a, recompensa r, estado futuro s'

    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma\, maxQ(s', a') - Q(s,a)]$

    $s \leftarrow s'$;

  Hasta que s sea terminal

Fig. 1.7 Algoritmo Q-*Learning*

El algoritmo Q-*Learning* es usado por los agentes para aprender de la experiencia, donde cada episodio es equivalente a una sesión de entrenamiento. En cada iteración el agente explora el ambiente y obtiene señales numéricas hasta alcanzar el estado objetivo. El propósito del entrenamiento es incrementar el conocimiento del agente, representado a través de los Q-valores. A mayor entrenamiento, mejores serán los valores que el agente puede utilizar para comportarse de una forma más óptima.

1.6 Conclusiones parciales

- Los problemas de secuenciación de tareas pueden verse como un problema de optimización del tipo asignación recursos limitados en el tiempo entre actividades que se pueden ejecutar en paralelo. Estos problemas son muy utilizados en los procesos de manufacturas tanto en ambientes *offline* como *online*, sin embargo, en la bibliografía consultada aparecen más resultados en ambientes *offline.*

- Los métodos heurísticos son capaces de encontrar buenas soluciones en un tiempo computacional adecuado donde los recursos disponibles y el número de operaciones aumentan y resulta costoso utilizar métodos exactos para su solución. La heurística NEH es un ejemplo de solución factible a problemas en los procesos de la industria manufacturera.
- Las Reglas de Despacho constituyen una variante de solución al problema de secuenciación de tareas, las cuales asignan prioridades en base al objetivo al optimizar, siendo MWKR la más conveniente para esta investigación debido a sus buenos resultados en términos del *makespan* que es el criterio a optimizar en este trabajo.
- El aprendizaje reforzado constituye una técnica factible para resolver problemas de secuenciación de tareas, ya que permite modelar y resolver estos escenarios como problemas de aprendizaje con múltiples agentes lo cual resulta beneficioso para obtener soluciones cercanas a las óptimas y puede competir con enfoques alternativos de solución.

Capítulo 2: Algoritmo para optimizar el proceso de secuenciación de reportes con múltiples restricciones

En este capítulo se introduce un problema de secuenciación de tareas que se presenta en la industria, el cual constituye un ejemplo de problema de secuenciación de tareas en ambientes tanto *online* como *offline*. Se proponen cuatro enfoques para la selección de lotes de reportes a ser procesados por cada máquina. Se tienen en cuenta para cada caso las múltiples restricciones que presenta el problema. El primer enfoque está basado en la selección aleatoria, el segundo en las ya mencionadas Reglas de Despacho (*Dispatching Rules*), el tercero en la heurística de Nawaz, Enscore y Ham (NEH) y por último una alternativa basada en Aprendizaje Reforzado. Conjuntamente con lo mencionado anteriormente, se propone además un sistema de fácil interacción con los usuarios para resolver la problemática planteada.

2.1 Descripción del problema

En los problemas de secuenciación de tareas de la vida cotidiana se procesan trabajos siguiendo un orden determinado, de forma tal que se logren optimizar los tiempos en los que transitan por el sistema. La ejecución de los reportes generados por los clientes de los supermercados a través de compras por diferentes vías, es un problema de secuenciación de tareas. Muchos de los reportes deben cumplir una serie de condiciones y el sistema debe ser capaz de priorizar su ejecución en caso de ser necesario (Coto Palacio et al. 2018).

En el problema a resolver existen *m* tipos de máquinas, donde cada tipo de máquina cuenta con una capacidad determinada, es decir, cada máquina dispone de una cantidad máxima de reportes que puede procesar al mismo tiempo. También se tienen *n* reportes en lotes, agrupados por tipo debido a la presencia de características iguales, y que deben ser procesados por las mismas máquinas. Estos reportes cuentan además con un identificador o instancia del reporte, conformado por el nombre y un número identificativo del idioma. También se tiene como dato el tiempo medio de procesamiento de un reporte, una cantidad determinada de pedidos, un tiempo total de procesamiento que está dado por la cantidad de pedidos y un nivel de prioridad. Los reportes son independientes, lo que quiere decir que pueden ser procesados al mismo tiempo siempre que haya capacidad disponible en las máquinas y en el sistema. Si las máquinas están ocupadas o la capacidad del sistema está en el límite, los reportes deben

esperar a que exista disponibilidad. Existe un recurso que no presenta límite de capacidad, solo el límite que impone el sistema para no sobrecargarse, en este recurso se ejecutan los reportes que no necesitan ser ejecutados por una máquina específica. Los recursos no están interrelacionados.

Algunos reportes necesitan ejecutarse en dos máquinas simultáneamente, y el procesamiento de los mismos está limitado a la existencia de disponibilidad en ambas máquinas en el mismo instante de tiempo. La Tabla 2.1 muestra un pequeño ejemplo donde se puede apreciar el formato de los datos que necesitan procesarse.

Tabla 2.1 Formato de los datos

Trabajo	Instancia del Trabajo	Tiempo Total (Segundos)	Número de ejec.	Tiempo por Reporte (Segundos)	Máq.	Prioridad
RAP1952	RAP1952-02	229455.709	103	2227.71562	7	1
RAP27279	RAP27279-01	90256.683	51	1769.71927	1-6	1
RAP3100	RAP3100-01	210295.147	119	1767.18611		3
RAP1952	RAP1952-03	233587.705	136	1717.54934	7-9	1
RAP26466	RAP26466-01	24852.869	17	1461.87476	1	4

En un instante de tiempo *t* se contará con un registro de todos los reportes generados, los cuales son almacenados en un fichero Excel (.xlsx), y nuevos reportes llegan al sistema diariamente. A las 12 de la noche se actualizan las colas de las máquinas, adicionando a los reportes que no han sido aún procesados los nuevos incorporados.

El sistema es capaz de asimilar el ambiente *offline*, al tener conocimiento de los trabajos almacenados hasta el día 1 (momento en que se ejecuta por primera vez) y el ambiente *online* en el que se desarrollarán los demás días (cada día a las 12 de la noche se adicionan nuevos pedidos).

Las múltiples restricciones que presenta el problema están enfocadas en los siguientes aspectos:

- Capacidad máxima del sistema
- Capacidad de las máquinas
- Los reportes solo deben ser procesados por una o dos máquinas simultáneamente como máximo
- Las máquinas trabajan en paralelo mientras tengan reportes que ejecutar

De acuerdo a las características resumidas en el capítulo 1 estamos en presencia de un problema con máquinas paralelas idénticas, que no presenta fechas de liberación o de terminación para los trabajos, estos no pueden ser interrumpidos una vez comenzada su ejecución y existen relaciones de precedencia, pues la prioridad que tienen asociada indica cuáles deben procesarse primero.

2.2 Propuesta de solución

El problema de secuenciar *n* trabajos independientes compuestos por una sola operación en *m* máquinas paralelas no relacionadas con el objetivo de minimizar el tiempo de completamiento de todos los trabajos es un problema combinatorio. Por lo tanto, el tiempo necesario para obtener la solución óptima utilizando ya sea un modelo matemático, una técnica de enumeración completa o una técnica de ramas y cotas crecerá exponencialmente con respecto al aumento del tamaño del problema. Es por esto que el uso de una heurística para superar esta situación es altamente inevitable (Coto Palacio et al. 2018).

En la Figura 2.1 se muestra el pseudocódigo general de la solución propuesta en esta investigación (método “execute”). Como se ha mencionado, la entrada del sistema es un documento como el presentado en la Tabla 2.1. Estos datos son almacenados en colas de prioridad en dependencia del recurso que se necesite usar, por lo que existe una cola por cada tipo de recurso. Cada lote de reportes es organizado según su prioridad y cuando existen múltiples reportes con la misma preferencia, el primer reporte en ejecutarse es el de mayor tiempo de procesamiento total, debido a que se desea minimizar el *makespan* final. En el mejor de los casos los trabajos tendrán un tiempo de procesamiento final igual al tiempo por reportes, esto ocurre solo si los conjuntos de reportes pertenecientes a un mismo lote son ejecutados al mismo tiempo. Por el contrario, en el peor de los casos el tiempo de procesamiento final será igual a su tiempo total. Este último caso sucede casi siempre al final de la ejecución del algoritmo, donde los reportes van finalizando su procesamiento y el sistema solo puede admitir pequeñas cantidades de reportes.

```
Input: PriorityQueue reports, int[] machines
Output: makespan
makespan = 0;
repeat
    int [] a = distribution(cant);
    for i = 1 to machines.size() do
        for j = 1 to a[i] do
            machines[i].pool[j] = reports[i].pop();
        end for
    end for
    Report report_exit = find_min();
    machines[report_exit].remove();
    makespan = max(makespan, report_exit.getFinalTime())
until reports.isEmpty()
return makespan
```

Fig. 2.1: Pseudocódigo del algoritmo propuesto.

A medida que se recorren las máquinas, siempre y cuando tengan asignada una cantidad mayor que cero (“distribution(cant)”), se extrae de su cola de prioridad el reporte con mayor preferencia y este pasa a ser procesado (“reports[i].pop()”). En cada momento se chequea si existe una segunda máquina por la que deba pasar el reporte y se verifica, según la disponibilidad de las dos, cual fue la cantidad real asignada para ese trabajo dentro del recurso.

Una vez concluidas las asignaciones, se extraen de la lista de ejecución los reportes de menor tiempo final (“find_min()”) y se mueve el reloj de la simulación a dicho tiempo. Los trabajos finalizarán su procesamiento en lote o individual en dependencia de su tiempo final. El sistema nunca se detiene, cada 24 horas se lee un nuevo fichero y se continúan procesando los trabajos, actualizando la lista de reportes y a medida que se conocen las prioridades y sus tiempos de procesamiento se van insertando en las colas de prioridad. Los nuevos trabajos pueden procesarse incluso antes que otros de días anteriores. Puede darse el caso también de que no existan trabajos procesándose debido a que se agotaron todos los que se encontraban en el sistema, en esta situación pasan a ejecutarse directamente los nuevos reportes siguiendo el orden plasmado por el algoritmo utilizado. Los nuevos ficheros presentan exactamente el mismo formato que el del día cero o primer día del sistema.

Un ejemplo de posible solución del problema sería la presentada en la Figura 2.2, donde mediante un diagrama de Gantt se muestra como quedarían secuenciados los reportes en las diferentes máquinas.

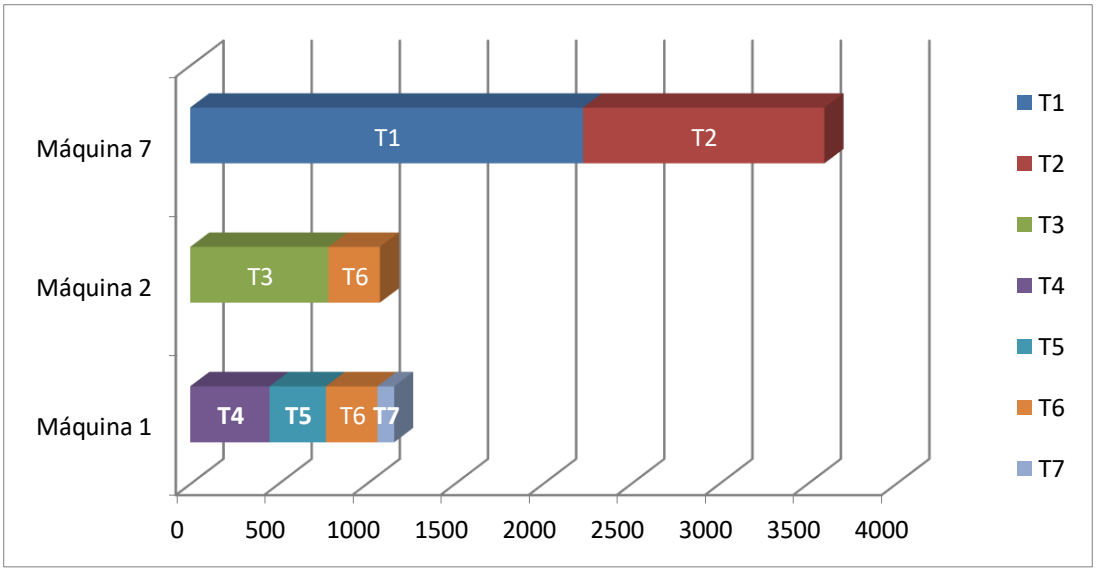

Fig. 2.2: Diagrama de Gantt

Las distribuciones de los reportes independientemente de la alternativa utilizada deben cumplir con las siguientes restricciones:

- La suma de los números la primera vez que se generan es el límite que impone el sistema.
- Cada número se corresponde con un tipo de máquina y no es mayor que la cantidad de máquinas existentes del tipo correspondiente ni de la cantidad de reportes que se encuentran en la cola de espera.
- Tiene en cuenta para las generaciones diferentes de la primera la cantidad de reportes que se encuentran dentro del sistema en el momento de generar los números.
- Los reportes que necesiten ser procesados por dos máquinas deben ser enviados a ambas al mismo tiempo y con la misma cantidad de lotes.

2.3 Alternativa basada en Pseudo-Aleatoriedad

Históricamente, la primera acepción de aleatoriedad está vinculada a la causalidad, por lo que el azar se consideró como la causa que produce un suceso aleatorio. Dichos sucesos se suponía que no podían modificarse por ninguna mediación o influencia, tanto humana como sobrenatural; es decir el fenómeno aleatorio no tiene causa o el “azar” es la propia causa (Máñez 2015).

En el presente trabajo la aleatoriedad se utiliza para determinar la prioridad con la que cada recurso escoge la cantidad de reportes que procesará. Los pasos a seguir para realizar la distribución siguiendo esta alternativa está plasmada en el pseudocódigo siguiente (Figura 2.3):

```
Input: int capacity, PriorityQueue reports, int[] machines
Output: int [] assignment
int [] random = random(num_machines);
for i = 1 to random.length do
    repeat
        assignment[i] += reports[random[i]].pop().numRequest();
    until capacity == 0 || reports[random[i]].isEmpty() ||
            machines[random[i]].size()==0
end for
return assignment
```

Fig. 2.3: Pseudocódigo de la alternativa Pseudo-Aleatoria

Para la ejecución del método es necesario conocer la cantidad que se desea asignar ("capacity"), esto se traduce en lo que le falta al sistema para estar ocupado en su totalidad. Luego se genera un orden para las máquinas con la ayuda del método *random* perteneciente a la clase *Math* de la API de Java. Una vez obtenida la prioridad se recorre cada máquina en ese orden, y se asignan la mayor cantidad de reportes posibles. En cada momento se tiene en cuenta la posibilidad de que un reporte necesite procesarse en más de una máquina a la vez, por lo que la cantidad asignada viene dada por el mínimo entre la capacidad disponible en cada máquina, la cantidad de trabajos proveniente del lote de reportes que se analiza y la capacidad disponible del sistema. En caso contrario, si solo debe procesarse por una máquina, se analiza la cantidad de operaciones del reporte contra la disponibilidad de la máquina y del sistema. Como resultado se devuelve la asignación final de los próximos reportes en el sistema en orden de prioridad.

A continuación, se explicará mediante un ejemplo la ejecución de este método donde se utilizan los datos de la Tabla 2.1. En un primer instante se genera la cantidad de reportes que van a procesarse por cada máquina de forma aleatoria y teniendo en cuenta no sobrepasar la cantidad admitida por las máquinas ni tampoco las del sistema. Las capacidades de las máquinas que se utilizarán son: máquina 0 (130), máquina 1 (20), máquina 2 (10), máquina 3 (2), máquina 4 (60), máquina 5 (10), máquina 6 (15),

máquina 7 (6), máquina 8 (4), máquina 9 (1). El sistema admitirá un total de 130 reportes.

En una primera iteración, la posible combinación de máquinas siguiendo la aleatoriedad podría ser: {M7, M9, M0, M1, M6}. Luego se asignan las cantidades de la siguiente manera:

- Máquina 7: Tiene una capacidad de 6 y dos reportes con prioridad 1, por lo que se selecciona el reporte RAP1952-03 por tener el mayor tiempo total de procesamiento, luego se analiza si existe capacidad en la máquina 9 ya que dicho reporte debe procesarse en ambas máquinas al mismo tiempo. Finalmente, se le asigna 1 reporte a la máquina, pues el mínimo entre las capacidades de ambas máquinas, del sistema y el número de ejecuciones del reporte es igual a 1.

  Al no cubrir la totalidad de la capacidad de la máquina se pasa a analizar el segundo reporte en cola (RAP1952-02), y de este se asigna la capacidad máxima posible (5).
- Máquina 9: No se le vuelve a asignar reportes, pues no tiene disponibilidad, ya que fue cubierta con la asignación anterior del reporte RAP1952-03 que necesitaba las máquinas 7 y 9.
- Máquina 0: Se le asignan 119, por tener un solo reporte (RAP3100-01) que solo debe procesarse en esa máquina y presenta un número de ejecuciones menor que la capacidad admitida por dicha máquina y por el sistema en general.
- Máquina 1: El primer reporte en cola es RAP27279-01 por ser el de mayor prioridad, el mismo debe procesarse además en la máquina 6. El sistema solo admite 4 reportes más, por lo que la cantidad debe ser distribuida de forma tal que se aproveche la mayor cantidad a asignar posible, esto equivale a 2 reportes para la máquina 1 y 2 reportes para la máquina 6.

La distribución final de las cantidades por máquinas quedó de la manera siguiente: {M7-6, M9-1, M0-119, M1-2, M6-2}. En la Figura 2.4, se muestra gráficamente la distribución de los reportes realizada en la primera iteración.

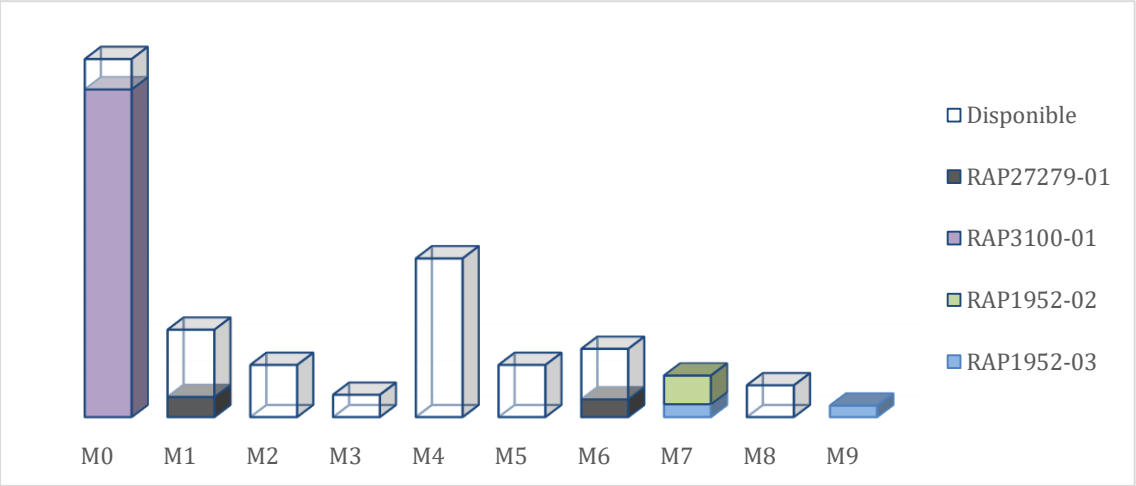

Fig. 2.4 Ejemplo de asignación en las máquinas con el método Pseudo-Aleatorio.

Luego de realizar esta asignación inicial, los trabajos que permanecen en cola esperando por la disponibilidad de las máquinas son representados en la Tabla 2.2.

Tabla 2.2 Trabajos que permanecen en cola

Trabajo	Instancia del Trabajo	Tiempo Total (Segundos)	Número de ejec.	Tiempo por Reporte (Segundos)	Máq	Prioridad
RAP1952	RAP1952-02	218316.131	98	2227,71562	7	1
RAP27279	RAP27279-01	86716.244	49	1769.71927	1-6	1
RAP3100	RAP3100-01	0	0	1767.18611		3
RAP1952	RAP1952-03	231869.16	135	1717.54934	7-9	1
RAP26466	RAP26466-01	24851.869	17	1461.87476	1	4

Una vez concluida la asignación, se procede a extraer del sistema el o los reportes con el menor tiempo de ejecución, que en este caso sería el RAP1952-03 con un tiempo de 1717.54934. La cantidad de reportes a distribuirse en la próxima iteración es de 2. El proceso se repite varias veces hasta que no existan más reportes en cola ni procesándose.

2.4 Alternativa basada en Reglas de Despacho (MWKR)

La alternativa basada en Reglas de Despacho presenta un funcionamiento similar al Pseudo-Aleatorio, con la variación de la técnica utilizada para dar el orden de prioridad a las máquinas (Figura 2.5). La técnica escogida fue la de Mayor tiempo de procesamiento restante (*Most Work Remaining*, MWKR) (Ecuación 2.1).

$$\max \quad Z_j = \sum_{t=q}^{O(j)} p'_{tj} \tag{2.1}$$

Donde:

$O(j)$ = Número total de operaciones del trabajo j

p'_{tj} = Tiempo de procesamiento restante de cada trabajo

Como el objetivo es la minimización del *makespan* final, se escogió esta técnica por ser la que más se ajusta al problema en cuestión.

El pseudocódigo que se muestra en la Figura 2.5 presenta el funcionamiento del enfoque basado en MWKR. Al igual que el Pseudo-aleatorio, esta propuesta cuenta como entrada principal con la cantidad de reportes que se desean asignar o la capacidad disponible en el sistema, así como los reportes por máquinas, con el objetivo de conocer la cantidad de trabajos que faltan por procesarse. La prioridad de las máquinas viene dada por una proporción de la suma total de reportes que faltan por ser procesados en cada una y la disponibilidad de las mismas en cada momento. Por consiguiente, a medida que se procesen los trabajos se actualizan las prioridades de las máquinas. La forma en que se calculan las prioridades propicia un conocimiento más certero sobre el posible tiempo de procesamiento total de los trabajos en cada máquina.

A medida que se realizan las asignaciones se va teniendo en cuenta la posible existencia de otra máquina por la que deba pasar el trabajo. La cola de prioridad permite en cada momento escoger la máquina con mayor tiempo de procesamiento restante, así como el reporte con mayor tiempo total.

Para ilustrar mejor el funcionamiento de esta propuesta se hará uso de la Tabla 2.1 para explicar una iteración, de forma similar a como se hizo en el epígrafe anterior. Las capacidades de las máquinas que se tendrán en cuenta son: máquina 0 (130), máquina 1 (20), máquina 2 (10), máquina 3 (2), máquina 4 (60), máquina 5 (10), máquina 6 (15), máquina 7 (6), máquina 8 (4), máquina 9 (1). El sistema admitirá un total de 130 reportes.

```
Input: int capacity, PriorityQueue reports, int[] machines
Output: int [] assignment
PriorityQueue dispatching_rule;
for j = 1 to machines.length do
    dispatching_rule[i] = totalTime[i] / machines[i].size();
end for
for i = 1 to dispatching_rule.length do
    repeat
        assignment[i] += reports[dispatching_rule[i]].pop().numRequest();
    until capacity == 0 || reports[dispatching_rule[i]].isEmpty() ||
          machines[random[i]].size()==0
end for
return assignment
```

Fig. 2.5: Pseudocódigo de la alternativa basada en MWKR

En cada una de las iteraciones se ordenan las máquinas siguiendo el procedimiento explicado con anterioridad, para este caso el par máquina-tiempo aproximado sería {M0-1617.655, M1-5755.3776, M2-0, M3-0, M4-0, M5-0, M6-6017.046, M7-77173.569, M8-0, M9-233586.705}. El orden de prioridad de los recursos en la primera distribución es: {M9, M7, M6, M1, M0}. Las cantidades se asignan de la siguiente manera:

- Máquina 9: Tiene una capacidad de 1 y un único reporte que debe ser procesado además por la máquina 7. Se analizan las capacidades de ambas máquinas (M9-1, M7-6) y el número de ejecuciones del trabajo (RAP1952-03, 136) y finalmente se le asigna 1 reporte.
- Máquina 7: El primer reporte analizado es el RAP1952-03 por tener prioridad 1 y el mayor tiempo total. Como este reporte debe ser procesado además por M9 y esta no tiene espacio disponible, entonces se descarta y se selecciona el siguiente trabajo en cola (RAP1952-02) y de este se asigna una cantidad de 5.
- Máquina 6: El único reporte de la cola de esta máquina es RAP27279-01, el cual se debe ejecutar también en la máquina 1. El mínimo entre las capacidades de ambas máquinas, la del sistema y el número de ejecuciones del reporte es de 15, por consiguiente, este es el número asignado a la máquina 6.
- Máquina 1: Tiene disponibles 5 de las 20 capacidades con las que cuenta. El primer reporte en cola es el RAP27279-01 por ser el de mayor prioridad, el mismo debe procesarse además en la máquina 6, en la cual no existe capacidad

ya que fue ocupada totalmente con anterioridad. El segundo reporte en cola es el RAP26466-01 con un número de ejecuciones de 17, del que se toman solo 5.

- Máquina 0: En este punto de la asignación al sistema solo le queda disponibilidad para 88 reportes. El reporte RAP3100-01 es el único perteneciente a esta máquina con 119 ejecuciones de las cuales solo se toman las 88 admitidas por el sistema.

La distribución final de las cantidades por máquina quedó de la manera siguiente: {M9-1, M7-6, M6-15, M1-20, M0-88}. En la Figura 2.6, se muestra gráficamente la distribución de los reportes realizada en la primera iteración.

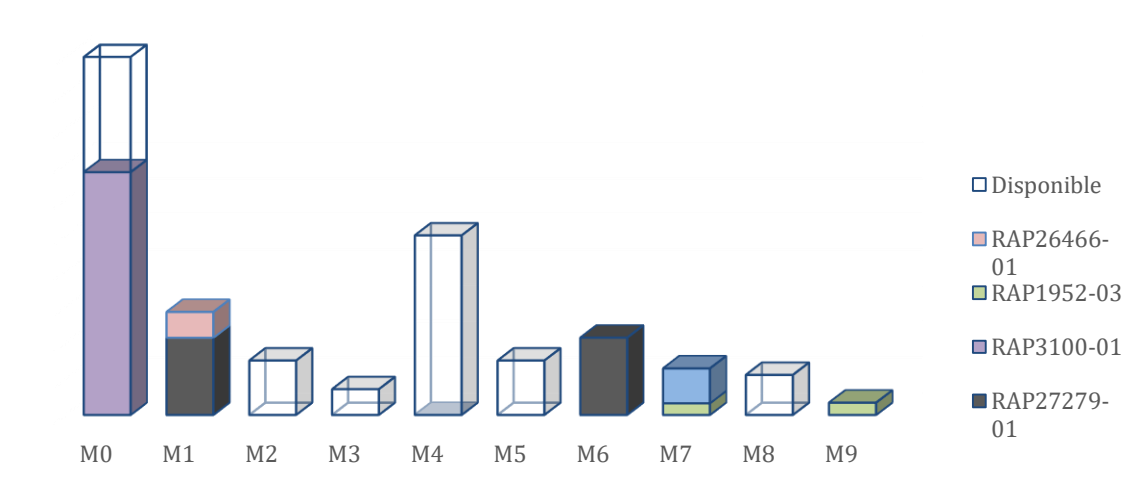

Fig. 2.6 Ejemplo de asignación en las máquinas MWKR.

Luego de realizar esta asignación inicial, los trabajos que permanecen en cola esperando por la disponibilidad de las máquinas son representados en la Tabla 2.3.

Tabla 2.3 Trabajos que permanecen en cola

Trabajo	Instancia del Trabajo	Tiempo Total (Segundos)	Núm. de ejec.	Tiempo por Reporte (Segundos)	Máq.	Prioridad
RAP1952	RAP1952-02	218316.131	98	2227,71562	7	1
RAP27279	RAP27279-01	63709,894	36	1769.71927	1-6	1
RAP3100	RAP3100-01	54782,769	31	1767.18611		3
RAP1952	RAP1952-03	231869.16	135	1717.54934	7-9	1
RAP26466	RAP26466-01	17542,497	12	1461.87476	1	4

Una vez concluida la asignación se procede a extraer del sistema el o los reportes con el menor tiempo de ejecución que en este caso sería el RAP26466-01 con un tiempo de 1461.87476. La cantidad de reportes a ser distribuida en la próxima iteración es de 5. El

proceso se repite varias veces hasta que no existan más reportes en cola ni siendo procesados.

2.5 Alternativa basada en NEH

La heurística constructiva NEH es aplicada en esta investigación como se muestra en el pseudocódigo de la Figura 2.7. La entrada y la salida presentan el mismo formato que las alternativas explicadas en epígrafes anteriores. La diferencia de esta propuesta de solución con respecto a las demás es el criterio utilizado para decidir el orden de las máquinas.

Primeramente, se cuenta con una lista de tiempos totales por máquina basada en la cantidad de reportes que debe procesar, organizada de mayor a menor ("tt"). Luego se coloca en "order" la primera máquina resultante del ordenamiento de "tt", para luego insertar la segunda máquina en orden tal que se minimice el *makespan* parcial. El método que devuelve la mejor posición según el criterio de minimizar el *makespan* parcial va calculando dicho valor en cada una de las posibles posiciones según el tamaño de "order" y por tanto se obtiene la posición con el menor tiempo encontrado en cada instante.

Por último, después de obtenida la secuencia de las máquinas se va asignando la mayor cantidad posible de reportes por máquina en dicho orden hasta completar el máximo admitido por el sistema o hasta agotar la cantidad de reportes existente. La asignación es recibida por el método "execute()" explicado con anterioridad, y este se encarga de asignar los reportes a cada máquina en el orden y con las cantidades fijadas por el enfoque NEH.

Al igual que se realizó en epígrafes anteriores, a continuación, nos basaremos en la Tabla 2.1 para ejemplificar el funcionamiento de la heurística NEH. Las capacidades de las máquinas que se tendrán en cuenta son las mismas utilizadas con anterioridad, pero vale la pena recordarlas: máquina 0 (130), máquina 1 (20), máquina 2 (10), máquina 3 (2), máquina 4 (60), máquina 5 (10), máquina 6 (15), máquina 7 (6), máquina 8 (4), máquina 9 (1). El sistema admitirá un total de 130 reportes.

```
Input: int capacity, PriorityQueue reports, int[] machines
Output: int [] assignment
ArrayList order, tt;
for j = 1 to machines.length do
    tt.add(i, totalTime[i]);
end for
Collections.sort(tt, Collections.reverseOrder());
order.add(tt.get(0);
for j = 1 to tt.size() do
    int index = position(order, tt.get(i).getMachine());
    order.add(index, tt.get(j));
end for
for i = 1 to order.size() do
    repeat
        assignment[i] += reports[order[i]].pop().numRequest();
    until capacity == 0 || reports[order[i]].isEmpty() ||
            machines[order[i]].size()==0
end for
return assignment
```

Fig. 2.7 Pseudocódigo de la alternativa basada en NEH.

El orden de las máquinas se irá construyendo poco a poco, a diferencia del MWKR, una vez organizadas las mismas de forma descendente se irá calculando el *makespan* parcial. El orden de las máquinas en un primer momento quedaría {M7, M9, M0, M1, M6}, de ese orden se fija la primera (M7) y se van seleccionando una a una las demás.

- Máquina 9: Si se coloca esta máquina en la primera posición el primer reporte que se tiene en cuenta es el RAP1952-03, del cual se toma solo 1 ejecución debido a la capacidad de este recurso, luego se toman 5 del RAP1952-02 perteneciente a la máquina 7. El *makespan* parcial sería 2227.71562.

  Si se ubica en la segunda posición, primero se analiza la máquina 7 y luego la 9, de la cual resulta un *makespan* parcial igual al anterior de 2227.71562.

  Por lo que se coloca esta máquina de forma aleatoria, una posible combinación quedaría {M7, M9}.
- Máquina 0: Al agregar esta máquina las posiciones posibles de colocación aumentan.

  Primera posición (M0, M9, M7): el único reporte en la cola de la máquina 0 es el RAP3100-01, del que se toman las 119 ejecuciones para un tiempo de ejecución de 1767.18611, luego se pasa a analizar la máquina 9, escogiéndose el RAP1952-03 con un tiempo de 1717.54934 y por último la máquina 7 con el

único trabajo que se puede tener en cuenta en este momento que es el RAP1952-02, con tiempo 2227.71562. El *makespan* parcial final sería 2227.71562.

Segunda (M9, M0, M7) y tercera (M9, M7, M0) posición: el *makespan* parcial es igual al anterior, ya que entrarían los mismos trabajos en otro orden, pero con igual *makespan* parcial final de 2227.71562.

Como el tiempo es igual en los tres casos, la máquina será colocada siguiendo el azar, {M9, M0, M7}.

- Máquina 1: En este punto las posibles soluciones son cuatro.

  Primera posición (M1, M9, M0, M7): la primera máquina se ocupa en su totalidad con los reportes RAP27279-01 y RAP26466-01 respectivamente para un tiempo de terminación de 1769.71927. En M9 se ejecutaría el RAP1952-03 con finalización a los 1717.54934 segundos, mientras que en M0 se completaría la capacidad del sistema con 99 reportes del RAP3100-01 para un tiempo de 1767.18611 segundos. El *makespan* parcial de esta combinación es de 1769.71927.

  Segunda (M9, M1, M0, M7), y tercera (M9, M0, M1, M7) posición: diferentes distribuciones, pero con igual tiempo de procesamiento.

  Cuarta posición (M9, M0, M7, M1): en esta distribución se presentan cambios significativos que difieren de las anteriores. La máquina 9 con capacidad de 1 procesará el único reporte que presenta, igualmente ocurre con la 0 donde se procesará las 119 ejecuciones del RAP3100-01 con tiempo de 1767.18611. Hasta el momento el sistema está ocupado al 93%, por lo que se pasa a analizar la máquina 7 de la cual la disponibilidad es de 5, el reporte con mayor prioridad debe ser procesado en la máquina 9 de la cual no existe capacidad, por lo que se toma el RAP1952-02 para un tiempo final de procesamiento de 2227.71562. Finalmente se tiene en cuenta la máquina 1 de la cual, el reporte que encabeza la cola debe ejecutarse en la máquina 6 y con un tiempo de 1769.71927 segundos. El *makespan* parcial de esta secuencia es de 2227.71562.

  Los tiempos según la posición del recurso varía. Se escoge el menor *makespan* parcial que sería de 1769.71927 y tres de las cuatro posiciones presentan ese tiempo, por lo que se escoge aleatoriamente la solución {M9, M0, M1, M7}.

- Máquina 6: No es necesario analizarla ya que el sistema completó su capacidad con la distribución anterior y además no presenta capacidad disponible.

La distribución final de las cantidades por máquinas siguiendo el pseudocódigo de la Figura 2.5 queda de la manera siguiente: {M9-1, M0-119, M1-5, M7-1, M6-4}. En la Figura 2.8, se muestra gráficamente la distribución de los reportes realizada en la primera iteración.

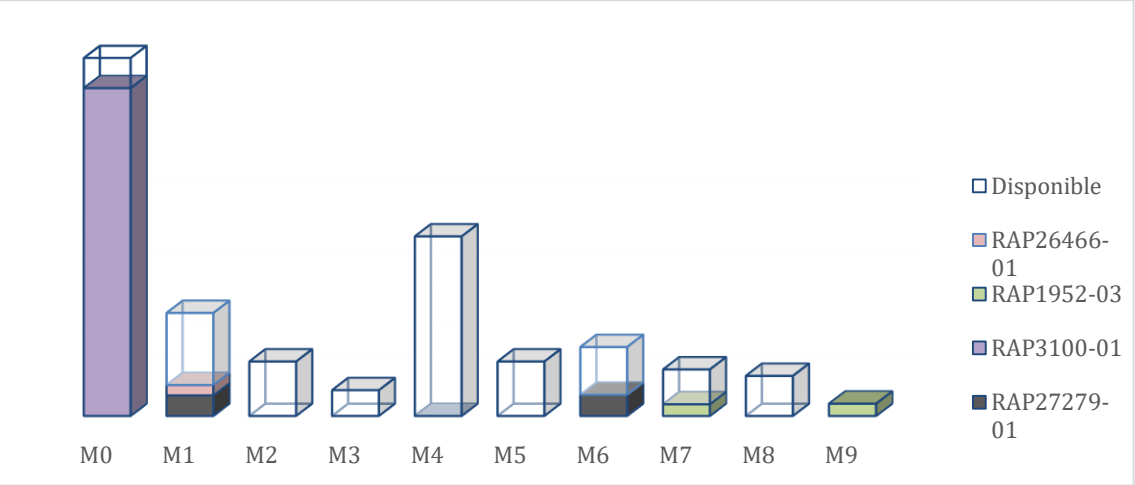

Fig. 2.8 Ejemplo de asignación en las máquinas NEH.

Luego de realizar esta asignación inicial, los trabajos que permanecen en cola esperando por la disponibilidad de las máquinas son representados en la Tabla 2.4.

Tabla 2.4 Trabajos que permanecen en cola, NEH.

Trabajo	Instancia del Trabajo	Tiempo Total (Segundos)	Número de ejec.	Tiempo por Reporte (Segundos)	Máq	Prioridad
RAP1952	RAP1952-02	229454.709	103	2227,71562	7	1
RAP27279	RAP27279-01	83176.802	47	1769.71927	1-6	1
RAP3100	RAP3100-01	0	0	1767.18611		3
RAP1952	RAP1952-03	231869.16	135	1717.54934	7-9	1
RAP26466	RAP26466-01	23389.996	16	1461.87476	1	4

Una vez concluida la asignación se procede a extraer del sistema el o los reportes con el menor tiempo de ejecución que en este caso sería el RAP26466-01 con un tiempo de 1461.87476. La cantidad de reportes a distribuir en la próxima iteración es de 5. El proceso se repite varias veces hasta que no existan más reportes en cola ni procesándose.

2.6 Alternativa basada en Q-*Learning*

En todo problema de secuenciación de tareas donde se involucra el QL se deben definir algunos elementos importantes como los estados, las acciones, los Q-valores, la

recompensa y la estrategia de selección de acciones. A continuación, se explican estos elementos con mayor profundidad.

Estados: en este caso existirá un estado en lo que se llama punto de decisión, que no es más que cada lugar donde haya que decidir la próxima máquina que ejecutará cada trabajo a procesar por el sistema, es decir, el estado actual está compuesto por un conjunto de máquinas elegidas con anterioridad, y de donde se debe elegir la próxima máquina o próximo estado a visitar.

Acciones: conjunto de posibles elementos a escoger desde el estado actual, en el problema de secuenciación de tareas que se aborda en esta tesis, la acción equivale a seleccionar la posición de una máquina para procesar los reportes.

Q-Valores: arreglo de pares estado–acción que refleja, para cada estado, cuan buena resulta cada una de las posibles acciones que se pueden tomar. Este valor se actualiza para cada uno de los estados involucrados en la solución parcial encontrada a cada momento de acuerdo a la ecuación 1.1.

Recompensa: valor recibido por el agente como señal de retroalimentación una vez que ha seleccionado una acción desde un estado determinado y que indica la calidad de la misma. La recompensa en este caso está estrechamente relacionada con el *makespan*.

Estrategia de selección de acciones: Para seleccionar las acciones la estrategia que se utiliza es la ε -greedy, pues el uso de la misma proporciona una mayor exploración del espacio de soluciones. El parámetro ε, como se indica en la literatura, debe ser un valor pequeño, idealmente 0.1, lo que indica que el 10% de las veces se explora y el 90% restante se explota, es decir, se escoge la mejor de las opciones existentes en el momento de la decisión.

La Figura 2.9 muestra el pseudocódigo perteneciente a la alternativa basada en QL enfocado en el problema actual. Solo existe un agente, el cual tiene la tarea de escoger en qué posición será colocada la próxima máquina. La selección de la próxima máquina es según el orden establecido, basado en el mayor tiempo de procesamiento total de los trabajos de cada máquina.

```
Input: int capacity, PriorityQueue reports, int[] machines
Output: int [] assignment
for k = 1 to 1000 do
    ArrayList order, tt;
    Double makespan_final = DOUBLE.MAX;
    for i = 1 to machines.length do
        tt.add(i, totalTime[i]);
    end for
    Collections.sort(tt, Collections.reverseOrder());
    int first_machine = rand.nextInt(machines.size());
    order.add(tt.get(first_machine).getMachine());
    for j = 1 to tt.size() do
        int index = QL(order, tt.get(j).getMachine());
        order.add(index, tt.get(j).getMachine());
    end for
    for i = 1 to order.size() do
        repeat
            assignment[i] += reports[order[i]].pop().numRequest();
            makespan = Math.max(makespan, assignment[i].getFinalTime());
        until capacity == 0 || reports[order[i]].isEmpty() ||
            machines[order[i]].size()==0
    end for
    updateQV(makespan, assignment);
    makespan_final = Math.min (makespan_final, makespan);
end for
return assignment
```

Fig. 2.9: Pseudocódigo de la alternativa basada en QL.

Para una mayor eficiencia en el aprendizaje del agente es recomendable realizar varias iteraciones para lograr que se visite la mayor cantidad posible de estados y así se obtenga una mejor recompensa o penalización en cada uno.

La primera máquina que formará parte de la secuencia en cada iteración es escogida al azar, es decir, de forma aleatoria mediante el método “nextInt” de la clase “Random”. Las demás son escogidas en orden y colocadas en una posición tal que se explote las soluciones encontradas con anterioridad o que se exploren nuevas soluciones. El pseudocódigo de selección de acciones se muestra en la Figura 2.10. En un primer instante se genera un valor entre 0 y 1, y si es menor que el valor definido como épsilon (ε) se selecciona de forma arbitraria la posición de esa máquina. En caso de ser mayor que ε se busca entre las acciones posibles desde el estado actual, cuál presenta mayor Q-valor y se toma esa acción como la mejor encontrada hasta el momento.

```
Input: ArrayList order, int machine, QL ql
Output: int index
if rand.nextDouble() < ϵ then
    index = rand.nextInt(orden.size()+1);
else
    for j = 1 to ql.size() do
        previous = Math.max(previous, ql.get(j).getQValue());
        index = previous.getAction();
    end for
end if
return index
```

Fig. 2.10 Pseudocódigo de selección de acciones del QL.

Esta alternativa presenta algunas variaciones respecto a los explicados con anterioridad. La principal modificación consiste en conocer el *makespan* parcial en cada asignación de reportes en las máquinas, con el fin de compararlo con el encontrado en iteraciones anteriores para hallar el menor tiempo de ejecución posible (“makespan_final”). En la línea donde se utiliza el “Math.max” se busca saber el tiempo de salida del último trabajo que se está ejecutando, por tanto, en la variable *makespan* se guarda el tiempo parcial encontrado en ese momento. Luego, dicho valor es utilizado para comparar la mejor solución encontrada hasta el momento con la perteneciente a la iteración actual y guardar en ella la mejor solución de las dos.

En cada iteración se actualizan los Q-valores de los pares estado acción involucrados en el proceso de construir la solución parcial. En el pseudocódigo de la Figura 2.11 se muestra el procedimiento de la actualización de los Q-Valores adaptado a la solución del problema en cuestión.

```
Input: ArrayList state, int action, int machine, ArrayList < QQ > qq,
       double makespan
for j = 1 to qq.size() do
    if (state.equals(qq.get(j).getState())) &&
      (machine == qq.get(j).getMachine()) &&
      (action == qq.get(j).getAction())  then
       UpdateQV();
    end if
end for
```

Fig. 2.11 Pseudocódigo de actualización de los Q-valores.

Un ejemplo de ejecución de este enfoque donde se utilizan los datos de la Tabla 2.1 se explicará a continuación, para una mejor claridad del proceso del QL. Las capacidades

de las máquinas y del sistema serán conciliadas de igual forma que los ejemplos explicados en los epígrafes anteriores. Los datos para calcular los Q-valores serán $\alpha = 0.1$, $\gamma = 0.8$, $\epsilon = 0.1$. Para la primera iteración los Q-valores son iguales, por lo que se partirá de una iteración n, donde se hayan visitado previamente varios pares estado-acción.

Primeramente, se selecciona una máquina aleatoriamente, en este caso será la máquina 1, luego se selecciona la que tenga mayor tiempo de procesamiento restante y así sucesivamente. El orden de las máquinas a seleccionar sería {M7, M9, M0, M6}, sin tener en cuenta M1 porque ya se seleccionó al azar como la primera.

- Máquina 7: se genera un número aleatorio entre 0 y 1, 0.25 > 0.1(épsilon), se selecciona la posición donde el Q-valor es mayor para ese par estado-acción (Q (M1, M7-posición 0) = 0.23; Q (M1, M7-posición 1) = 0.10), en este caso sería en la posición 0. El orden queda {M7, M1}.
- Máquina 9: el número generado para comparar con épsilon es de 0.05, por lo que se escoge la posición aleatoria que podría ser la 2 {M7, M1, M9}.
- Máquina 0: el número aleatorio esta vez podría ser mayor que épsilon y se selecciona el mejor de los Q-valores (Q (M7M1, M0-posición 0) = 0.03; Q (M7M1, M0-posición 1) = 0.05; Q (M7M1, M0-posición 2) = 0.102; Q (M7M1, M0-posición 3) = 0.65). La mejor posición es la 3 y la secuencia hasta el momento sería {M7, M1, M9, M0}.
- Máquina 6: siguiendo el procedimiento anterior, un posible número aleatorio sería 0.89 > 0.1. Los Q-valores podrán ser (Q (M7M1M9M0, M6-posición 0) = 0.23; Q (M7M1M9M0, M6-posición 1) = 0.63; Q (M7M1M9M0, M6-posición 2) = 0.228; Q (M7M1M9M0, M6-posición 3) = 0.005; Q (M7M1M9M0, M6-posición 4) = 0.38). La secuencia sería {M7, M6, M1, M9, M0}.

La distribución final de las cantidades por máquinas quedaría de la manera siguiente: {M7-6, M6-15, M1-20, M9-1, M0-88}. En la Figura 2.12, se muestra gráficamente la distribución de los reportes realizada en esta iteración.

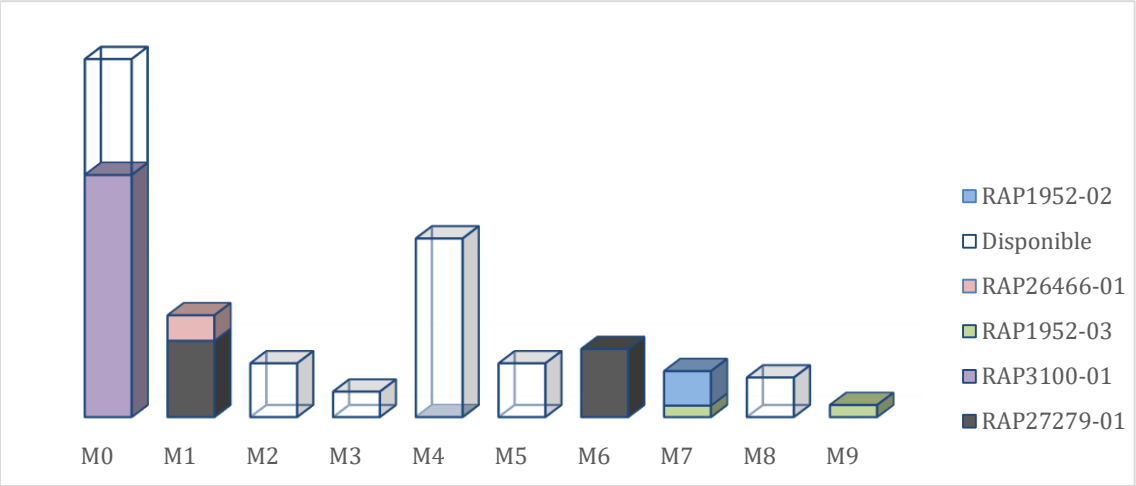

Fig. 2.12 Ejemplo de asignación en las máquinas QL.

Luego de realizar esta asignación inicial, los trabajos que permanecen en cola esperando por la disponibilidad de las máquinas son representados en la Tabla 2.5.

Tabla 2.5 Trabajos que permanecen en cola

Trabajo	Instancia del Trabajo	Tiempo Total (Segundos)	Número de ejec.	Tiempo por Reporte (Segundos)	Máq	Prioridad
RAP1952	RAP1952-02	218316.131	98	2227,71562	7	1
RAP27279	RAP27279-01	63709.894	36	1769.71927	1-6	1
RAP3100	RAP3100-01	54782.766	31	1767.18611		3
RAP1952	RAP1952-03	231869.16	135	1717.54934	7-9	1
RAP26466	RAP26466-01	17542.496	12	1461.87476	1	4

Una vez concluida la asignación se procede a extraer del sistema el o los reportes con el menor tiempo de ejecución que en este caso sería el RAP26466-01 con un tiempo de 1461.87476. La cantidad de reportes a distribuir en la próxima iteración, es de 5. El proceso se repite varias veces hasta que no existan más reportes en cola ni siendo procesados.

Cuando el último reporte sale del sistema se actualizan los Q-valores siguiendo los pasos presentados en la Figura 2.11, por ejemplo, si el *makespan* de esta iteración fue 15800 segundos los Q-valores se calcularían de la siguiente forma:

- Q (M1, M7-posición 0) = Q (M1, M7-posición 0) + 0.1 * [1.0/15800 + 0.8 * max_a, Q((M7M1, a')) - Q (M1, M7-posición 0)]
- Q (M7M1, M9-posición 2) = Q (M7M1, M9-posición 2) + 0.1 * [1.0/15800 + 0.8 * max_a, Q((M7M1M9, a')) - Q (M7M1, M9-posición 2)]

- Q (M7M1M9, M0-posición 3) = Q (M7M1M9, M0-posición 3) + 0.1 * [1.0/15800 + 0.8 * max_a, Q((M7M1M9M0, a')) - Q (M7M1M9, M0-posición 3)]
- Q (M7M1M9M0, M6-posición 1) = Q (M7M1M9M0, M6-posición 1) + 0.1 * [1.0/15800 + 0.8 * max_a, Q((M7M6M1M9M0, a')) - Q (M7M1M9M0, M6-posición 1)]

En todas las iteraciones se realiza el mismo procedimiento y al final se toma la mejor combinación encontrada para así distribuir todos los reportes siguiendo el mejor orden encontrado.

2.7 Diagrama de clases del Sistema

La aplicación se implementó en el lenguaje de programación orientado a objetos Java. La Figura 2.13 muestra un diagrama de clases en el cual se recogen las principales clases utilizadas para resolver el problema en cuestión y las relaciones entre ellas. El software cuenta con cinco clases encargadas del procesamiento del algoritmo y con varias clases visuales encargadas de recoger y mostrar la información necesaria.

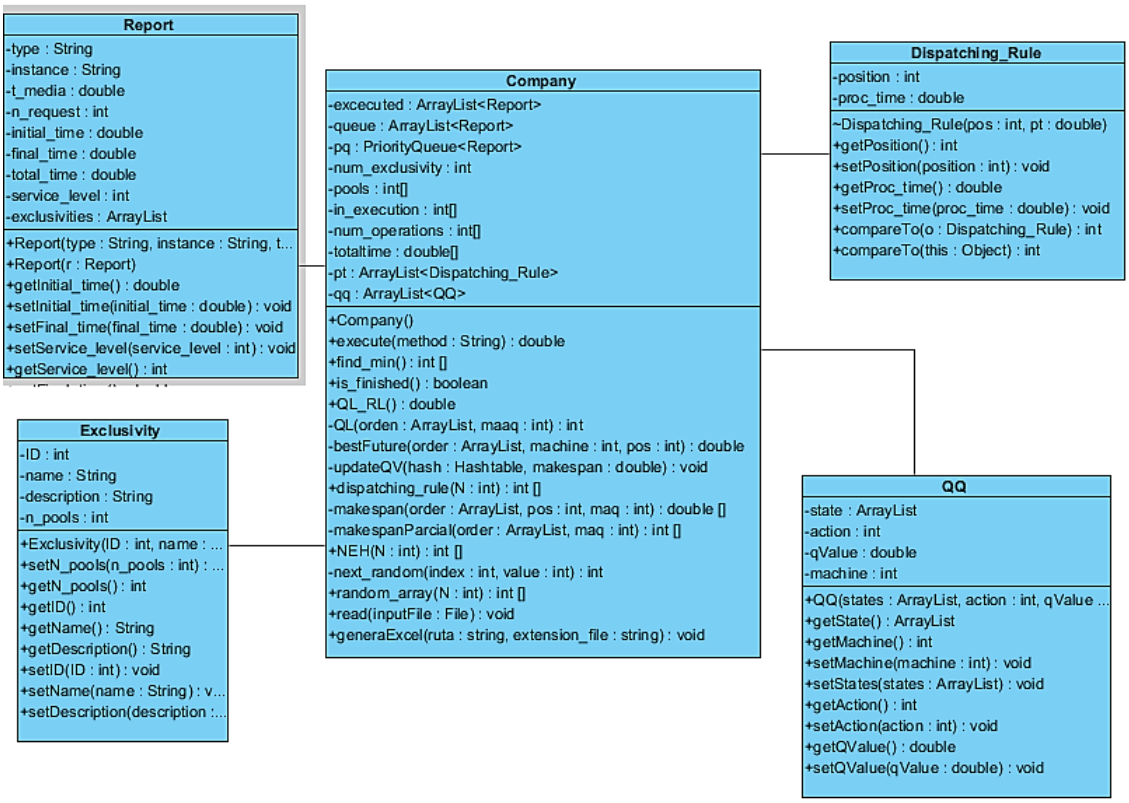

Fig. 2.13 Diagrama de clases

Como se indica en la Figura 2.8, las clases “QQ”, “Exclusivity”, “Report” y “Dispatching_Rule” están estrechamente relacionadas con la clase “Company”. La clase “Exclusivity” contiene las características que distinguen a las máquinas, como el nombre, una pequeña descripción y la capacidad máxima de reportes que pueden procesar al mismo tiempo. También se cuenta con la clase “Report”, la misma contiene la estructura de los reportes como su tipo, la instancia que se traduce en su idioma, el tiempo por reporte, el tiempo total del lote de reportes, el nivel de prioridad, cantidad de reportes, así como el tiempo inicial y final en que comenzó y terminó su ejecución respectivamente. De esta última clase se conoce, además de los atributos mencionados, si existe otra máquina por la que deben ser ejecutados los reportes, esta es una característica importante pues el sistema debe controlar constantemente la posibilidad de que un reporte vaya a dos máquinas en el mismo instante de tiempo.

Otra clase perteneciente al sistema es la llamada “Dispatching_Rule”. La misma contiene, por cada máquina, la suma de los tiempos totales de procesamiento de los reportes dividido por la capacidad de la máquina, menos el espacio que tenga ocupado en cada momento que se realice una acción que involucre la Regla de Despacho implementada en este trabajo, por lo que la información almacenada en el atributo “proc_time” va a sufrir cambios constantemente.

La clase “QQ” también forma parte del sistema implementado, esta solo será utilizada en caso de aplicarse la alternativa basada en Aprendizaje Reforzado. “QQ” contiene la relación de cada par estado-acción involucrado en el proceso de construcción de las soluciones (estado, acción, máquina, Q-valor).

“Company” es la clase principal del sistema por ser la que realiza todos los procesamientos de los reportes. Por un lado, cuenta con atributos que son instancias de las otras clases, y por otro lado con atributos propios de la clase como la suma de los tiempos totales por máquina (totalTime), el número de operaciones agrupados por máquinas (num_operations), la cantidad de reportes que están en ejecución (in_execution), entre otros. El método “execute” es el encargado de realizar la simulación del algoritmo, en este se realizan las asignaciones de los reportes a las máquinas de acuerdo a lo generado por las distintas alternativas. Los métodos “QL_RL”, “QL”, “bestFuture” y “updateQV” están relacionados con el procedimiento del Q-*Learning* para las asignaciones. En el caso NEH se utilizan los métodos de “NEH”, “makespanParcial” y “makespan”. Para la técnica basada en la Regla de

Despacho MWKR se utiliza el método “dispatching_rule”. Para leer y generar los ficheros Excel seleccionados por el usuario se recurre a los métodos “read” y “generaExcel” respectivamente, los cuales utilizan la librería Apache POI.

2.8 Interfaz visual del sistema

El sistema implementado con el objetivo de llevar a cabo la optimización del proceso de secuenciación de reportes con múltiples restricciones, se compone de una ventana principal y varias ventanas complementarias. En este epígrafe se muestran dichas ventanas, y se detallan los pasos a seguir por el usuario para procesar los reportes basado en alguna de las cuatro alternativas que se explicaron anteriormente.

En la Figura 2.14 se muestra la ventana principal del sistema, la cual se encuentra estructurada de la manera siguiente: una barra de menú con las opciones de Archivo y Ayuda; y cuatro cuadros de diálogo, donde cada uno representa una alternativa de solución y el correspondiente *makespan* final de cada corrida se muestra en el área de texto.

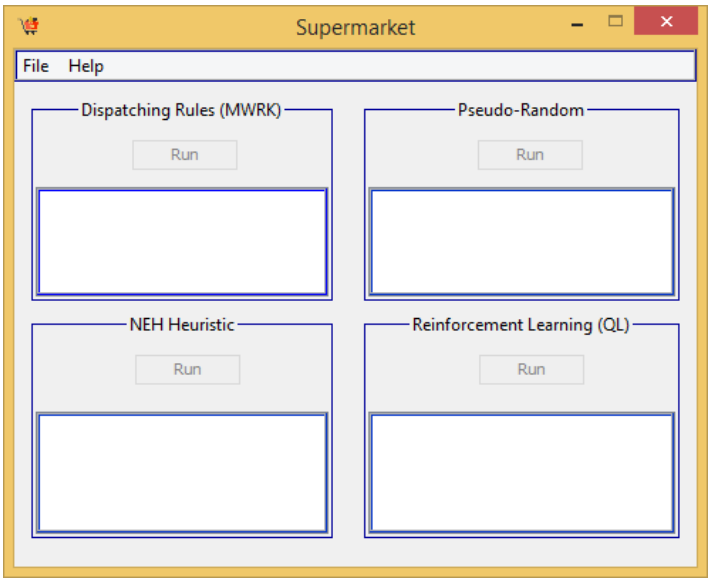

Fig. 2.14 Ventana principal del sistema.

El primer paso es cargar el fichero con extensión XLSX perteneciente a los datos que se desean procesar (Tabla 2.1). Para realizar dicha operación se debe acceder al menú *File* y en “*Import data*” escoger un fichero (Figura 2.15). Si la acción es llevada a cabo con éxito muestra un cartel como el presentado en la Figura 2.16, en caso de que el fichero escogido no tenga la estructura establecida por el sistema se mostrará un cartel informando del error ocurrido (Figura 2.17). Además de la opción “*Import data*”, en *File*, existe la opción “*Clear*”, la cual elimina los datos presentes en todas las áreas de

texto de la ventana principal. Solo cuando existen datos cargados correctamente en la aplicación se activarán los botones "*Run"*, los cuales están destinados a la ejecución de la alternativa en correspondencia con el botón presionado.

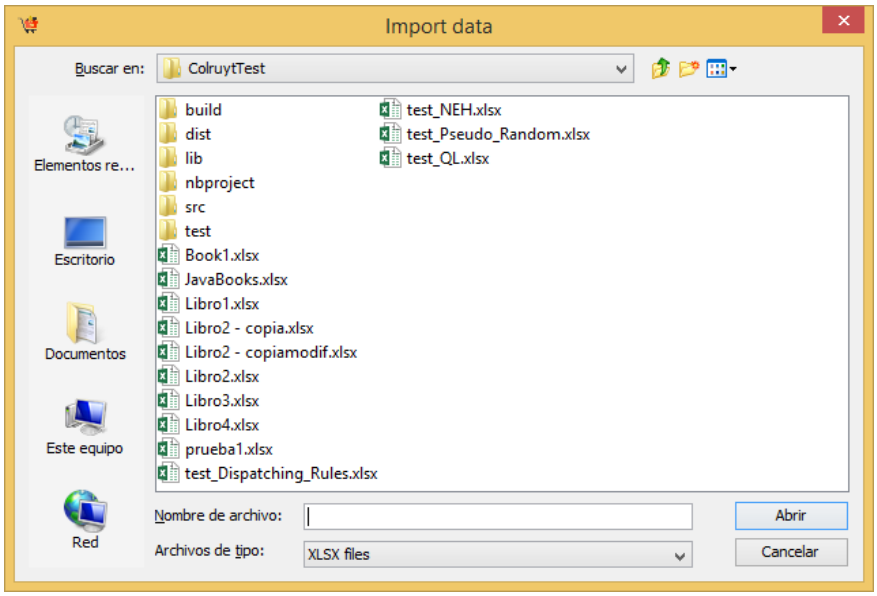

Fig. 2.15 Importar datos

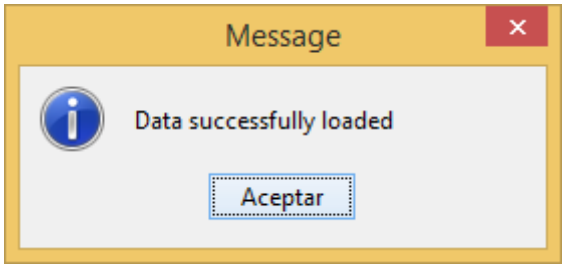

Fig. 2.16 Mensaje de carga exitosa de datos

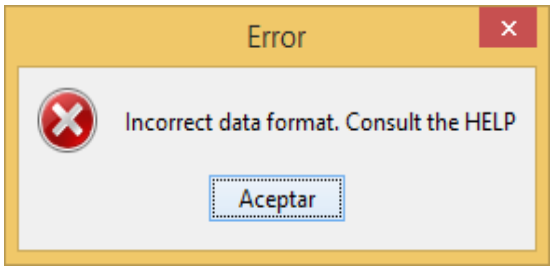

Fig. 2.17 Mensaje de error de carga de datos

La otra opción perteneciente al menú de la aplicación corresponde a la ayuda ("*Help*"), donde se explica con detalles el funcionamiento de cada uno de las alternativas implementadas. Por cada ejecución realizada, el sistema permite guardar la solución encontrada, es decir, el orden en que las máquinas procesan los reportes. En el menú *File* se activará la opción "*Export data*" que llevará a cabo tal encomienda. Estos datos pueden ser guardados en formato XLSX y XLS. El sistema brinda como información el éxito o no de las distintas operaciones realizadas a cada momento.

2.9 Conclusiones parciales

Se implementó un algoritmo para la secuenciación de reportes el cual utiliza cuatro alternativas para la distribución de las cantidades de reportes por máquina: Regla de Despacho MWKR, la heurística NEH, Q-*Learning* y se desarrolló además una

perspectiva guiada a la aleatoriedad, con el propósito de encontrar la mejor solución, es decir, aquella que minimice el tiempo de completamiento de todas las tareas a realizar.

El sistema diseñado facilita una mejor interacción con el usuario a partir de interfaces visuales de fácil acceso y comprensión, en la cual a través de la lectura de los reportes se procesan los datos y se brinda un resultado final que puede exportarse a formato Excel para luego ponerlo en práctica dentro de la empresa.

Capítulo 3: Resultados experimentales

Comparar los resultados obtenidos por dos o más algoritmos para un conjunto de problemas determinado es un paso esencial en áreas como el aprendizaje automático o la optimización. El uso de herramientas estadísticas como las pruebas de hipótesis es clásicamente el enfoque utilizado para sacar conclusiones de estas comparaciones, para esto existen distintos paquetes de software estadístico propietario como SPSS, Minitab, Statistica, SAS, etc, que cubren todas las necesidades de cualquier usuario de técnicas estadísticas básicas o avanzadas. Por otro lado, en los últimos años el uso del lenguaje R ha surgido con fuerza como alternativa de software libre en ambientes docentes y de investigación.

En este capítulo se detallan las características de la simulación realizada al algoritmo con las diferentes alternativas, especificándose los parámetros que se tuvieron en cuenta para cada una. Los resultados obtenidos, se comparan entre ellos teniendo en cuenta el *makespan* final en cada una de las instancias. Para la validación se aplicaron pruebas estadísticas de *Friedman* y *Wilcoxon* en el SPSS, mientras que en R se utilizó la extensión de *Friedman*, Iman y Davemport.

3.1 Especificaciones del experimento

Se midió el funcionamiento del algoritmo implementado teniendo en cuenta las características del ambiente tanto *offline* como *online*. Para ambos casos la capacidad de las máquinas no varía ni los parámetros básicos del QL y el 10% de los reportes deben ejecutarse en dos de las diez posibles máquinas. En este epígrafe se explicará con profundidad la conformación de los experimentos para cada uno de los ambientes, utilizando como caso de estudio un supermercado belga.

Parámetros de la simulación:

- Capacidad de las máquinas: {M_0, M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8, M_9}

  Para todos los ficheros: {130, 15, 60, 10, 4, 2, 10, 6, 1, 20}

  (La máquina M0 es la encargada de procesar aquellos reportes que no necesiten un tipo de máquina en específico).

Capacidad del sistema: 130 reportes simultáneamente.

Parámetros del QL (recomendados en la literatura (Martínez Jiménez 2012)):

- Velocidad de aprendizaje: $\alpha = 0.1$

- Factor de descuento: $\gamma = 0.8$
- Épsilon: $\epsilon = 0.1$

3.1.1 Ambiente *offline*

En un primer instante se decidió tomar como punto de partida una simulación sin la existencia de cambios en el sistema, es decir, se escogió un fichero para cada experimento y durante la simulación no se incorporó ningún lote de reportes, pues no se tiene en cuenta el ambiente *online* del sistema. Este análisis nos permite conocer con certeza el rendimiento de las distintas alternativas, para luego explicar en epígrafes posteriores la capacidad que tienen los mismos de asimilar cambios en el sistema.

Para dicho ambiente se escogieron seis ficheros, los cuales presentan diferencias como la cantidad de lotes de reportes, y los tipos de reportes en sí. Uno de estos ficheros se muestra en la Tabla 3.1. Las cantidades son mostradas a continuación:

- Fichero 1: 1416
- Fichero 2: 19 (Tabla 3.1)
- Fichero 3: 75
- Fichero 4: 100
- Fichero 5: 300
- Fichero 6: 500

Tabla 3.1 Fichero de 19 lotes de reportes

Trabajo	Instancia del Trabajo	Tiempo Total (Seg)	Número de ejec.	Tiempo por Reporte (Seg)	Máquinas	Prioridad
RAP1952	RAP1952-02	229455	103	2227,715621	7	1
RAP27279	RAP27279-01	90256	51	1769,719275	0	3
RAP3100	RAP3100-01	15905	9	1767,186111	0	2
RAP1952	RAP1952-03	233587	136	1717,549338	7	1
RAP26466	RAP26466-01	24852	17	1461,874765	0	1
RAP1951	RAP1951-02	151902	111	1368,487595	7-1	3
RAP1903	RAP1903-04	134127	102	1314,971363	7	1
RAP6095	RAP6095-01	451647	351	1286,744875	0-3	4
RAP1951	RAP1951-03	125216	102	1227,607529	7	4
RAP3099	RAP3099-01	12962	11	1178,324091	0	2
RAP3101	RAP3101-01	15069	13	1159,134077	0	1
RAP24586	RAP24586-02	81479	73	1116,143932	0	2
RAP1919	RAP1919-04	126054	114	1105,740026	7	3
RAP3102	RAP3102-01	10986	10	1098,5992	0	4
RAP25805	RAP25805-01	81323	79	1029,408329	0	2
RAP3146	RAP3146-01	67122	70	958,8887714	0	4
RAP1926	RAP1926-01	275672	289	953,8818201	7	2
RAP25894	RAP25894-01	233049	248	939,7119556	0	1
RAP4559	RAP4559-01	47596	52	915,3156346	0-9	1

La siguiente Tabla muestra cómo se distribuyen en cada fichero la cantidad de reportes a ser procesados por cada máquina, por ejemplo, en el fichero 1 existen 27 lotes de reportes que deben ser procesados en la máquina 1.

Tabla 3.2 Distribución por fichero de la cantidad de reportes a procesar por cada máquina.

Fichero / Máquina	1	2	3	4	5	6
0	1149	12	60	63	238	403
1	27	1	1	6	1	7
2	210	0	5	21	47	95
3	32	1	2	2	6	11
4	33	0	2	3	8	13
5	17	0	1	2	5	6
6	9	0	0	1	4	1
7	49	7	10	10	14	3
8	25	0	2	1	5	11
9	8	1	0	1	1	0

3.1.2 Ambiente *online*

En ambientes *online* la simulación está dirigida al tiempo en que demora cada uno en procesar los trabajos teniendo en cuenta el ambiente variable del sistema. Cada 24 horas serán incluidos en el sistema nuevos lotes de reportes que deben ser incorporados a la ejecución.

El tiempo de la simulación que se tendrá en cuenta será de 2, 3, 4, 5 y 6 días para cada uno de los enfoques, es decir, si la simulación es de 2 días, se añadirá un fichero más a las 12a.m. del día siguiente, luego de esto, no existirán más variaciones en las colas de trabajo. Una vez realizada todas las inserciones de reportes en el sistema, se deja concluir el procesamiento de los trabajos, aunque transcurran más días, pues solo se necesita medir el nivel de adaptación del algoritmo a diversos cambios en un tiempo determinado y conocer su *makespan* final.

Al igual que en el ambiente *offline*, las Tablas 3.3, 3.4, 3.5, 3.6 y 3.7 muestran cómo se distribuye en cada fichero la cantidad de reportes a ser procesados por cada máquina.

Todas las simulaciones parten del mismo archivo, la diferencia radica en que se adiciona uno más que el añadido a la simulación anterior.

Tabla 3.3 Simulación de 2 días.

Simulación	Día / Máquina	1	2
2 días	Ninguna	12	60
	1	1	1
	2	0	5
	3	1	2
	4	0	2
	5	0	1
	6	0	0
	7	7	10
	8	0	2
	9	1	0

Tabla 3.4 Simulación de 3 días.

Simulación	Día / Máquina	1	2	3
3 días	Ninguna	12	60	238
	1	1	1	1
	2	0	5	47
	3	1	2	6
	4	0	2	8
	5	0	1	5
	6	0	0	4
	7	7	10	14
	8	0	2	5
	9	1	0	1

Tabla 3.5 Simulación de 4 días.

Simulación	Día / Máquina	1	2	3	4
4 días	Ninguna	12	60	238	403
	1	1	1	1	7
	2	0	5	47	94
	3	1	2	6	11
	4	0	2	8	13
	5	0	1	5	7
	6	0	0	4	1
	7	7	10	14	3
	8	0	2	5	11
	9	1	0	1	0

Tabla 3.6 Simulación de 5 días.

Simulación	Día / Máquina	1	2	3	4	5
5 días	Ninguna	12	60	238	403	500
	1	1	1	1	7	10
	2	0	5	47	94	32
	3	1	2	6	11	74
	4	0	2	8	13	1
	5	0	1	5	7	0
	6	0	0	4	1	1
	7	7	10	14	3	0
	8	0	2	5	11	10
	9	1	0	1	0	20

Tabla 3.7 Simulación de 6 días.

Simulación	Día / Máquina	1	2	3	4	5	6
6 días	Ninguna	12	60	238	403	500	558
	1	1	1	1	7	10	20
	2	0	5	47	94	32	2
	3	1	2	6	11	74	1
	4	0	2	8	13	1	10
	5	0	1	5	7	0	35
	6	0	0	4	1	1	45
	7	7	10	14	3	0	10
	8	0	2	5	11	10	2
	9	1	0	1	0	20	36

3.2 Marco experimental estadístico

Para determinar si existen diferencias significativas entre dos o varios algoritmos, se pueden aplicar distintas técnicas estadísticas. Las pruebas son clasificadas generalmente en dos grupos:

- **Pruebas paramétricas:** estas pruebas son las más empleadas. Cuantifican la asociación o independencia entre una variable cuantitativa y una categórica. Exigen además ciertos requisitos previos para su aplicación: la distribución Normal de la variable cuantitativa en los grupos que se comparan, la homogeneidad de varianzas en las poblaciones de las que proceden los grupos y un tamaño de muestra superior a 30. Su incumplimiento implica que se debe recurrir a pruebas estadísticas no paramétricas (Sheskin 2006).
- **Pruebas no paramétricas:** estas pruebas no requieren asumir normalidad de la población y en su mayoría se basan en el ordenamiento de los datos. Ejemplos de estas pruebas son la prueba de *Wilcoxon* (para comparar los resultados de dos algoritmos) y la prueba de *Friedman* (para comparaciones de varios algoritmos) (Sheskin 2006).

Los datos obtenidos para la realización de los experimentos no presentan distribución normal ni varianza homogénea, por lo que no se pueden aplicar pruebas paramétricas, de ahí que surja la necesidad de aplicar pruebas no paramétricas a nuestros resultados.

Para comparar múltiples algoritmos en múltiples problemas, la metodología general recomendada es primero aplicar pruebas para detectar si al menos uno de los algoritmos tiene un rendimiento diferente al de los demás. Posteriormente, si se detecta una diferencia significativa, se aplica una prueba de pares con la corrección post-hoc correspondiente para comparaciones múltiples.

En la siguiente imagen se resumen las pruebas estadísticas recomendadas para diferentes escenarios.

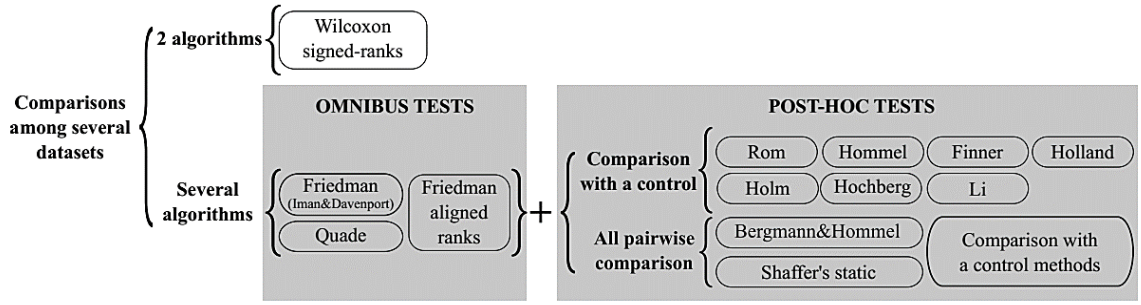

Fig. 3.1 Pruebas estadísticas recomendadas para diferentes escenarios, tomada de (Calvo & Santafé 2016)

La prueba de Friedman con la extensión *Iman* y *Davemport* es probablemente la prueba más popular, y generalmente es una buena opción cuando se comparan más de cinco algoritmos diferentes. Por el contrario, cuando se comparan cinco o menos algoritmos diferentes, los rangos alineados de *Friedman* y la prueba de *Quade* son alternativas más potentes (García et al. 2010).

Prueba de *Friedman*: esta prueba puede utilizarse en aquellas situaciones en las que se seleccionan n grupos de k elementos de forma que los elementos de cada grupo sean lo más parecidos posible entre sí, y a cada uno de los elementos del grupo se le aplica uno de entre k "tratamientos", o bien cuando a cada uno de los elementos de una muestra de tamaño n se le aplican los k "tratamientos". La hipótesis nula que se contrasta es que las respuestas asociadas a cada uno de los "tratamientos" tienen la misma distribución de probabilidad o distribuciones con la misma mediana, frente a la hipótesis alternativa de que por lo menos la distribución de una de las respuestas difiere de las demás (Friedman 1940).

Prueba de Suma de Rangos de *Wilcoxon:* cuando se trata de variables medibles en por lo menos una escala ordinal y pueden suponerse poblaciones continuas la prueba no paramétrica más potente es la de *Wilcoxon*. La hipótesis nula del contraste postula que

las muestras proceden de poblaciones con la misma distribución de probabilidad; la hipótesis alternativa establece que hay diferencias respecto a la tendencia central de las poblaciones y puede ser direccional o no.

La significación asintótica en ambas pruebas es comparada con el valor 0.05, lo cual indica que si la significación es mayor que dicho valor se acepta la hipótesis nula y en caso contrario se acepta la hipótesis alternativa.

3.3 Análisis estadístico de los resultados en ambientes *offline*

En este epígrafe se realizan pruebas estadísticas en ambientes *offline* con dos herramientas diferentes, SPSS y R. La Tabla 3.7 contiene los datos de la simulación realizada a los seis ficheros que fueron detallados en el epígrafe 3.1. En la Tabla 3.8 se muestra un análisis estadístico descriptivo de los datos expuestos en la Tabla 3.7. Los estadísticos descriptivos no son más que un estudio de los datos con el fin de encontrar el promedio, la desviación típica, el mínimo y el máximo de los elementos.

Tabla 3.7 Datos de las pruebas estadísticas en ambiente *offline*

Fichero/Alternativa	Pseudo-Aleatorio	NEH	MWKR	Q-*Learning*
Fichero 1	1486043.619	1507456.622	1446676.297	**1418194.8708**
Fichero 2	213976.431	218498.712	213132.546	**212954.391**
Fichero 3	240421.723	253717.145	**239754.938**	**239754.938**
Fichero 4	390923.816	386542.268	439143.066	**386238.711**
Fichero 5	466915.308	487930.899	462347.397	**462294.595**
Fichero 6	524179.116	577369.61	522730.409	**522724.8527**

Tabla 3.8 Estadísticos descriptivos

	N	Media	Desviación típica	Mínimo	Máximo
PseudoAleatorio	6	553743,335500	472804,2187274	213976,4310	1486043,6190
NEH	6	571919,208834	478087,2723856	218498,7120	1507456,6190
MWKR	6	553964,108834	454836,5961377	213132,5460	1446676,2970
QLearning	6	539858,393084	446835,6463170	212954,3910	1418194,8708

3.3.1 Análisis usando la herramienta SPSS

Al aplicar la prueba de Friedman a los resultados de las diferentes alternativas en un ambiente *offline* para el conjunto de datos de la Tabla 3.7, se obtiene una significación asintótica de 0.03 (Tabla 3.9b), por lo que se rechaza la hipótesis nula y se acepta la hipótesis alternativa, concluyendo que existen diferencias significativas entre al menos dos de las alternativas. Luego de analizar los rangos promedio en la Tabla 3.9a, se logra identificar que el Q-*Learning* arroja mejores resultados que los demás.

Tabla 3.9 Prueba de Friedman ambiente *offline*. a) Rangos promedio, b) Estadísticos de contraste.

	Rango promedio
PseudoAleatorio	3.00
NEH	3.67
MWKR	2.33
QLearning	**1.08**

a)

N	6
Chi-cuadrado	14.000
gl	3
Sig. asintót.	.003

b)

El próximo paso es realizar las pruebas de *Wilcoxon* dos a dos, para conocer las diferencias entre las alternativas:

- NEH y Pseudo-Aleatorio (Tabla 3.10a): 0.219 > 0.05, no existen diferencias significativas.
- MWKR y Pseudo-Aleatorio (Tabla 3.10b): 0.345 > 0.05, no existen diferencias significativas.
- Q-*Learning* y Pseudo-Aleatorio (Tabla 3.10c): 0.028 < 0.05, existen diferencias significativas.
- MWKR y NEH (Tabla 3.10d): 0.173 > 0.05, no existen diferencias significativas.
- Q-*Learning* y NEH (Tabla 3.10e): 0.028 < 0.05, existen diferencias significativas.
- Q-*Learning* y MWKR (Tabla 3.10f): 0.028 < 0.05, existen diferencias significativas.

Tabla 3.10 Prueba de *Wilcoxon* dos a dos. a) NEH y Pseudo-Aleatorio, b) MWKR y Pseudo-Aleatorio, c) Q-*Learning* y Pseudo-Aleatorio, d) MWKR y NEH, e) Q-*Learning* y NEH, f) Q-*Learning* y MWKR.

	NEH - PseudoAleatorio
Z	-1.992(a)
Sig. asintót. (bilateral)	.219

a)

	MWKR - PseudoAleatorio
Z	-.943(a)
Sig. asintót. (bilateral)	.345

b)

	QLearning - PseudoAleatorio
Z	-2.201(a)
Sig. asintót. (bilateral)	.028

c)

	MWKR - NEH
Z	-1.363(a)
Sig. asintót. (bilateral)	.173

d)

	QLearning - NEH
Z	-2.201(a)
Sig. asintót. (bilateral)	.028

e)

	QLearning - MWKR
Z	-2.201(a)
Sig. asintót. (bilateral)	.028

f)

a Basado en los rangos positivos.
b Prueba de los rangos con signo de Wilcoxon

Según las pruebas de Wilcoxon dos a dos, se puede concluir que la alternativa utilizando Q-Learning presenta diferencias significativas respecto a las demás y como se demostró utilizando la prueba de Friedman, el QL es el que mejores resultados arroja según su rango promedio.

3.3.2 Análisis usando la herramienta R

En el año 2016 se propuso en (Calvo & Santafé 2016) un paquete de R llamado scmamp (*Statistical Comparison of Multiple Algorithms in Multiple Problems*), el cual está destinado a ser una herramienta que simplifica todo el proceso de análisis de los resultados obtenidos al comparar algoritmos, desde el proceso de cargar los datos hasta la producción de gráficos y tablas.

Siguiendo la sugerencia de (Calvo & Santafé 2016) primero se realiza la prueba de *Iman* y *Davemport* (imanDavenportTest(datos)), que da como resultado una significación asintótica de 0.0001027, demostrando la existencia de diferencias significativas entre al

menos dos de los algoritmos. Luego se realiza el test de *Nemenyi* para determinar la diferencia crítica. Todos aquellos algoritmos cuya diferencia en rendimiento sea mayor que la diferencia crítica, se consideran significativamente diferentes.

nemenyiTest(DatosOffline)

Critical difference = 2.0862, k = 4, df = 20

Una vez ejecutado el test, mediante el comando diff.matrix se puede conocer la diferencia entre cada par de alternativas. La prueba de *Nemenyi* tiene la ventaja de tener un gráfico asociado para representar los resultados de la comparación. Este diagrama se puede obtener de la siguiente manera: plotCD(results.matrix = datos, alpha = 0.05). El resultado se puede ver en la Figura 3.2, donde cada alternativa se ubica en un eje de acuerdo con su rango promedio. Aquellas que no muestran diferencias significativas se agrupan utilizando una línea horizontal. La imagen también muestra el tamaño de la diferencia crítica requerida para considerar dos alternativas como significativamente diferentes. Como se puede apreciar existen diferencias significativas entre el Q-*Learning* y todas las demás.

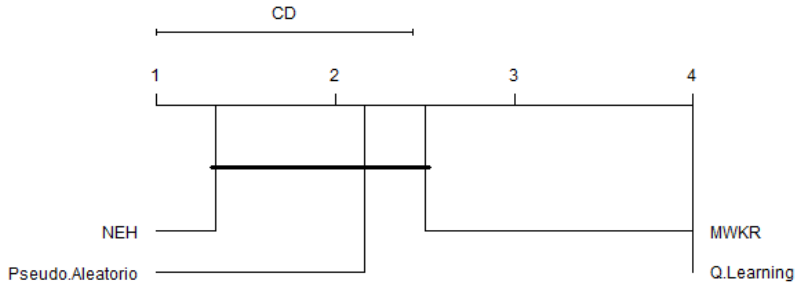

Fig. 3.2 Diagrama de la prueba de Nemenyi, ambientes *offline.*

Siguiendo los pasos descritos en (Calvo & Santafé 2016), también se realiza la prueba post hoc de *Friedman* con la corrección de *Bergmann y Hommel*. De acuerdo con la documentación del paquete de R, ambos pasos se pueden llevar a cabo en una sola línea de código.

test.res<-postHocTest(data = datos, test = 'friedman', correct = 'bergmann')

Esto ejecuta la prueba post-hoc, calculando el p-valor sin procesar para cada par de alternativas. Estos p-valores también se corrigen para pruebas múltiples usando la corrección de *Bergman y Hommel*. Además, se obtiene un resumen con los valores promedio de cada variante en todo el conjunto de datos.

$summary

Pseudo.Aleatorio	NEH	MWKR	Q.Learning
553743.3	570252.5	554797.4	539858.3

$raw.pval

	Pseudo.Aleatorio	NEH	MWKR	Q.Learning
Pseudo.Aleatorio	NA	0.2635524773	0.65472085	0.0139062969
NEH	0.2635525	NA	0.11752487	0.0003466194
MWKR	0.6547208	0.1175248681	NA	0.0441713449
Q.Learning	0.0139063	0.0003466194	0.04417134	NA

$corrected.pval

	Pseudo.Aleatorio	NEH	MWKR	Q.Learning
Pseudo.Aleatorio	NA	0.352574604	0.65472085	0.041718891
NEH	0.35257460	NA	0.35257460	0.002079716
MWKR	0.65472085	0.352574604	NA	0.048342690
Q.Learning	0.04171889	0.002079716	0.04834269	NA

3.4 Análisis estadístico de los resultados en ambientes *online*

En este epígrafe al igual que el anterior, se realizan pruebas estadísticas utilizando las herramientas SPSS y R, con la diferencia de que en este caso el ambiente en que se procesan los reportes es *online.*

La Tabla 3.11 muestra los datos del ambiente *online*, a los cuales se le aplicarán las pruebas estadísticas. Dichos datos están conformados por el *makespan* total resultante de la ejecución de las distintas alternativas sobre los ficheros detallados en el epígrafe 3.1.2. En la Tabla 3.12 se realiza un análisis estadístico descriptivo de los datos de la Tabla 3.11.

Tabla 3.11 Datos de las pruebas estadísticas en ambiente *online*

Fichero/Alternativa	Pseudo-Aleatorio	NEH	MWKR	Q-*Learning*
2 días	453853.645	464736.816	452423.5322	**452153.653**
3 días	648549.929	663821.367	646112.093	**645115.361**
4 días	1105584.015	1164879.599	1102181.104	**1100101.323**
5 días	2181669.357	1198283.552	1135730.171	**1115620.114**
6 días	3256345.00	2660122.01	2113600.236	**2053652.12**

Tabla 3.12 Estadísticos descriptivos

	N	Media	Desviación típica	Mínimo	Máximo
PseudoAleatorio	5	1531200.00	1172878.990	453853.645	3256345
NEH	5	1246368.20	863430.291	464736.816	2660122.01
MWKR	5	1110009.20	646274.268	452423.532	2113600.236
QLearning	5	1072728.20	619544.547	452153.653	2053652.12

3.4.1 Análisis usando la herramienta SPSS

A cada juego de datos se le aplican pruebas no paramétricas, exactamente la prueba de *Friedman* para una valoración general de las diferencias significativas entre las alternativas. En este ambiente se comprobó que existen diferencias entre las mismas. Luego se realizó el test de *Wilcoxon* dos a dos, para saber con exactitud entre cuales existen esas diferencias. Estas pruebas se aplicaron usando el SPSS y serán explicadas más adelante en este epígrafe.

Al aplicar la prueba de *Friedman* a los resultados de las alternativas en el ambiente *online* para el conjunto de datos de la Tabla 3.11, se obtiene una significación asintótica de 0.007 (Tabla 3.13b), por lo que se rechaza la hipótesis nula y se acepta la hipótesis alternativa, concluyendo que existen diferencias significativas entre al menos dos de los enfoques. En un análisis realizado a los rangos promedio en la Tabla 3.13a, se logra identificar que al igual que en el ambiente *offline* el Q-*Learning* arroja mejores resultados que las demás alternativas.

Tabla 3.13 Prueba de Friedman ambiente *online.* a) Rangos promedio, b) Estadísticos de contraste.

	Rango promedio
PseudoAleatorio	3.60
NEH	3.20
MWKR	2.20
QLearning	**1.00**

a)

N	5
Chi-cuadrado	12.120
gl	3
Sig. asintót.	.007

b)

Luego de realizado el test de *Friedman* se pasa a ejecutar las pruebas de *Wilcoxon* dos a dos, para conocer exactamente entre que alternativas existen las diferencias significativas:

- NEH y Pseudo-Aleatorio (Tabla 3.14a): 0.345 > 0.05, no existen diferencias significativas.
- MWKR y Pseudo-Aleatorio (Tabla 3.14b): 0.043 < 0.05, existen diferencias significativas.
- Q-*Learning* y Pseudo-Aleatorio (Tabla 3.14c): 0.043 < 0.05, existen diferencias significativas.
- MWKR y NEH (Tabla 3.14d): 0.080 > 0.05, no existen diferencias significativas.
- Q-*Learning* y NEH (Tabla 3.14e): 0.043 < 0.05, existen diferencias significativas.
- Q-*Learning* y MWKR (Tabla 3.14f): 0.043 < 0.05, existen diferencias significativas.

Tabla 3.14 Prueba de *Wilcoxon* dos a dos, ambiente *online*. a) NEH y Pseudo-Aleatorio, b) MWKR y Pseudo-Aleatorio, c) Q-*Learning* y Pseudo-Aleatorio, d) MWKR y NEH, e) Q-*Learning* y NEH, f) Q-*Learning* y MWKR.

	NEH - PseudoAleatorio
Z	-.944(a)
Sig. asintót. (bilateral)	.345

a)

	MWKR - PseudoAleatorio
Z	-2.023(a)
Sig. asintót. (bilateral)	.043

b)

	QLearning - PseudoAleatorio
Z	-2.023(a)
Sig. asintót. (bilateral)	.043

c)

	MWKR - NEH
Z	-1.604(a)
Sig. asintót. (bilateral)	.080

d)

	QLearning - NEH
Z	-2.023(a)
Sig. asintót. (bilateral)	.043

e)

	QLearning - MWKR
Z	-2.023(a)
Sig. asintót. (bilateral)	.043

f)

a Basado en los rangos positivos.
b Prueba de los rangos con signo de Wilcoxon

Según las pruebas de *Wilcoxon* dos a dos, se puede apreciar que el Q-*Learning* presenta diferencias significativas respecto a las demás y como se demostró en *Friedman,* el QL es el que mejores resultados arroja según su rango promedio.

3.4.2 Análisis usando la herramienta R

En el epígrafe anterior se explicaron las características del paquete *scmamp* para el análisis estadístico de datos, por lo que se procederá directamente a mostrar los resultados para el ambiente *online.*

El test *Iman y Davenport* da una significación de 0.0007291, la cual se compara con 0.05, lo que demuestra que existen diferencias significativas entre las alternativas. Luego se realiza el test de *Nemenyi* para conocer las diferencias críticas que se muestran en la Figura 3.3

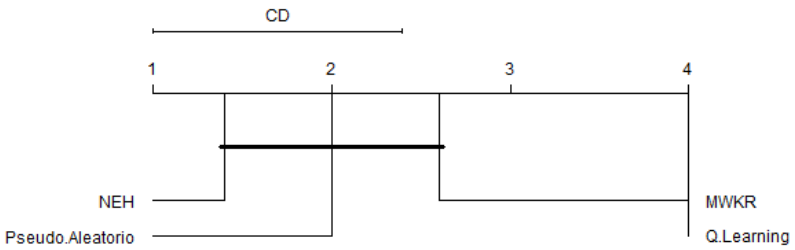

Fig. 3.3 Diagrama de prueba de Nemenyi, ambientes *online.*

Los enfoques que no muestran diferencias significativas están agrupados mediante una línea horizontal. Como se puede apreciar, en la Figura 3.3 se confirman las diferencias significativas del Q-*Learning* con respecto a las demás alternativas.

Si aplicamos la prueba post hoc de Friedman con la corrección de Bergmann y Hommel, podemos arribar a las mismas conclusiones sobre las diferencias entre las alternativas. A continuación, se muestran los resultados de dichas pruebas.

test.res <- postHocTest(data = DatosOffline, test = 'friedman', correct = 'bergmann')

Esto ejecuta la prueba post-hoc, calculando el p-valor sin procesar para cada par de alternativas. Estos p-valores también se corrigen para pruebas múltiples usando la corrección de *Bergman y Hommel*. Además, se obtiene un resumen con los valores promedio de cada variante en todo el conjunto de datos.

$summary

Pseudo.Aleatorio	**NEH**	**MWKR**	**Q.Learning**
1209200	1200369	1210209	1073329

$raw.pval

	Pseudo.Aleatorio	NEH	MWKR	Q.Learning
Pseudo.Aleatorio	NA	0.462432726	0.46243273	0.014305878
NEH	0.46243273	NA	0.14164469	0.001450862
MWKR	0.46243273	0.141644690	NA	0.086410733
Q.Learning	0.01430588	0.001450862	0.08641073	NA

$corrected.pval

	Pseudo.Aleatorio	NEH	MWKR	Q.Learning
Pseudo.Aleatorio	NA	0.46243273	0.4624327	0.03382150
NEH	0.46243273	NA	0.4249341	0.00870517
MWKR	0.46243273	0.42493407	NA	0.04291764
Q.Learning	0.03382150	0.00870517	0.04291764	NA

3.5 Conclusiones parciales

En este capítulo se realizaron diversas pruebas estadísticas con el objetivo de comparar el comportamiento del algoritmo implementado utilizando cada una de las cuatro alternativas para la distribución de la cantidad de reportes por máquina. Se utilizaron ficheros con datos reales, los cuales diferían en la cantidad total de reportes a ejecutar. Además, se realizó una simulación en los dos tipos de ambientes clásicos en problemas de secuenciación de tareas: determinístico y el estocástico.

Para el análisis estadístico de los datos se utilizaron dos herramientas, el SPSS y el R. En ambos casos se puede concluir la existencia de diferencias significativas entre las alternativas, específicamente entre el Q-*Learning* y cada uno de los restantes enfoques, demostrando así la superioridad del Q-*Learning* en la solución a problemas de secuenciación de tareas en la industria.

Conclusiones

Como resultado de esta investigación se desarrolló un sistema para la optimización del proceso de secuenciación de reportes utilizando cuatro enfoques de solución; cumpliéndose de esta forma el objetivo general de esta tesis.

Del análisis realizado a los diferentes algoritmos reportados en la literatura para resolver problemas de secuenciación de tareas, se seleccionaron cuatro enfoques de los más utilizados para analizar su desempeño en la solución del problema planteado, uno basado en pseudo-aleatoriedad, otro en la regla de despacho MWKR, un tercero que sigue los principios de la heurística NEH y por último el Aprendizaje Reforzado.

El algoritmo propuesto para la secuenciación de reportes se implementó utilizando los cuatro enfoques seleccionados para la distribución de los reportes que deben procesarse por las máquinas: Pseudo-Aleatoria, NEH, MWKR y Q-Learning. Se mostró la eficacia del algoritmo propuesto en la optimización de problemas de secuenciación de reportes, tanto en ambientes offline como online.

Los cuatro enfoques utilizados para la distribución de los reportes en las máquinas fueron sometidos a pruebas mediante una simulación con datos reales, destacándose el desempeño del Q-Learning por encima de los otros enfoques.

Recomendaciones

1. Someter a prueba otras funciones de retroalimentación en la alternativa Q-*Learning*.
2. Valorar la utilización de otras heurísticas o metaheurísticas en la solución del problema de secuenciación de reportes.
3. Modificar el algoritmo para optimizar nuevos objetivos como la tardanza, debido a su importancia en la industria manufacturera.

Referencias Bibliográficas

Abiri, M.B., Zandieh, M. & Alem-Tabriz, A., 2009. A Tabu Search approach to hybrid flow shops scheduling with sequence-dependent setup times. *Journal of Applied Science*, 9(9), pp.1740–1745.

Adressi, A., Tasouji Hassanpour, S. & Azizi, V., 2016. Solving group scheduling problem in no-wait flexible flowshop with random machine breakdown. *Decision Science Letters*, 5, pp.157–168.

Agarwal, A., Colak, S. & Eryarsoy, E., 2006. Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. *European Journal of Operational Research*, 169(3), pp.801–815. Available at: http://www.sciencedirect.com/science/article/pii/S0377221705001463.

Albers, S., 1997. Better bounds for online scheduling. *In Proc. 29th Annual ACM Symposium on Theory of Computing*, pp.130–139.

Albers, S. & Schröder, B., 2001. An Experimental Study of Online Scheduling Algorithms. *Lecture Notes in Computer Science*, 1982, pp.11–22.

Alberto, A. et al., 2008. Secuenciación de operaciones para configuraciones de planta tipo flexible Job Shop: Estado del arte. , 5(3).

Alcaide López de Pablo, D., 1995. Problemas de planificación y secuenciación determinística: modelización y técnicas de resolución. In *Curso de ciencia y tecnología. Universidad de la Laguna.*

Alisantoso, D., Khoo, L.P. & Jiang, P.Y., 2003. An immune algorithm approach to the scheduling of a flexible PCB flow shop. *The International Journal of Advanced Manufacturing Technology*, 22(11), pp.819–827. Available at: https://doi.org/10.1007/s00170-002-1498-5.

Allahverdi, A., 2015. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2), pp.345–378. Available at: http://www.sciencedirect.com/science/article/pii/S0377221715002763.

Al-Turki, U., Andijani, A. & Arifulsalam, S., 2014. A New Dispatching Rule for the Stochastic Single-Machine Scheduling Problem. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 80, pp.165–170.

Alvarez, M., Toro, E. & Gallego, R., 2008. Simulated Annealing Heuristic For Flow Shop Scheduling Problems. *Scientia et Technica*, XIV(40), pp.159–164.

Bartal, Y. et al., 1995. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51, pp.359–366.

Bertel, S. & Billaut, J.-C., 2004. A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. *European Journal of Operational Research*, 159(3), pp.651–662. Available at: http://www.sciencedirect.com/science/article/pii/S037722170300434X.

Blackstone, J.H., Phillips, D.T. & Hogg, G.L., 1982. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research*, 20, pp.27–45.

Bozejko, W., Pempera, J. & Smutnicki, C., 2013. Parallel tabu search algorithm for the hybrid flow shop problem. *Computer & Industrial Engineering*, 65, pp.466–474.

Brasel, H. et al., 2009. Heuristic Constructive Algorithms for Open Shop Scheduling to Minimize Mean Flow Time.

Calvo, B. & Santafé, G., 2016. scmamp : Statistical Comparison of Multiple Algorithms in Multiple Problems. , 8, pp.248–256.

Castillo, I. & Roberts, C.A., 2001. Real-time control/scheduling for multi-purpose batch plants. *Computers & industrial engineering*, 41(2), pp.211–225.

Conway, R.W., 1965. Priority dispatching and job lateness in a job shop. *Journal of Industrial Engineering*, 16, pp.228–237.

Coto Palacio, J. et al., 2018. Algoritmo para la optimización del proceso de secuenciación de reportes. *Informática 2018*.

Djellab, H. & Djellab, K., 2002. Preemptive Hybrid Flowshop Scheduling problem of interval orders. *European Journal of Operational Research*, 137(1), pp.37–49. Available at: http://www.sciencedirect.com/science/article/pii/S0377221701000947.

El-Bouri, A., Azizi, N. & Zolfaghari, S., 2007. A comparative study of a new heuristic based on adaptive memory programming and simulated annealing: The case of job shop scheduling. *European Journal of Operational Research*, 177(3), pp.1894–1910. Available at: http://www.sciencedirect.com/science/article/pii/S037722170500843X.

Fernandez-viagas, V. et al., 2015. NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. *Computers and Operation Research*. Available at: http://dx.doi.org/10.1016/j.cor.2015.02.002.

Fonseca Reyna, Y.C. et al., 2015. A REINFORCEMENT LEARNING APPROACH FOR SCHEDULING PROBLEMS. *Revista Investigacion Operacional*, 36(3), pp.225–231.

Fonseca Reyna, Y.C. & Márquez Delgado, J.E., 2012. Comparación de un algoritmo genético y la técnica nube de partículas en la solución del flow shop scheduling. *Revista Cubana de Ciencias Informaticas*, 6(4), pp.17–26.

Friedman, M., 1940. A Comparison of Alternative Tests of Significance for the Problem of m Rankings. *Ann. Math. Statist.*, 11(1), pp.86–92. Available at: http://dx.doi.org/10.1214/aoms/1177731944.

G, J.R. & Rajaram, A., 2016. Improved NEH-Heuristic Job Scheduling for An Optimal System Using Meta-Heuristic GA – INSMG. , 9(7), pp.213–226.

Gabel, T., 2009. *Multi-Agent Reinforcement Learning Approaches for Distributed Job-Shop Scheduling Problems*. Universität Osnabrück.

Gabel, T. & Riedmiller, M., 2007. On a Successful Application of Multi-Agent Reinforcement Learning to Operations Research Benchmarks. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. Honolulu , USA, pp. 68–75.

García, S. et al., 2010. Advanced non parametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10), pp.2044–2064.

Garey, M.R., Johnson, D.S. & Sethi, R., 1976. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1(2), pp.117–129.

Ge, H.W. et al., 2006. A NOVEL PARTICLE SWARM OPTIMIZATION-BASED APPROACH FOR JOB-SHOP SCHEDULING. In *Computational Methods*. Springer, pp. 1093–1098. Available at: http://www.springerlink.com/index/RRQ81H046665P045.pdf.

Glass, C.A., Potts, C.N. & Shade, P., 1994. Unrelated paral-lel machine scheduling using local search. *Mathematical and Computer Modelling*, 20, pp.41–52.

Graham, R.L. et al., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5, pp.287–326.

Grossmann, I., 2005. Enterprise-wide optimization: A new frontier in process systems engineering. *AIChE Journal*, 51(7), pp.1846–1857.

Haupt, R., 1989. A survey of priority rule-based scheduling. *OR Spektrum*, 11, pp.3–16.

Herroelen, W. & Leus, R., 2005. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2), pp.289–306.

Hmida, J. Ben et al., 2014. Production scheduling for continuous manufacturing systems with quality constraints. *Production & Manufacturing Research*, 2(1), pp.95–111. Available at: http://dx.doi.org/10.1080/21693277.2014.892846.

Ho, N.B., Tay, J.C. & Lai, E.M.K., 2007. An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, 179(2), pp.316–333.

Jennings, N.R., Sycara, K. & Wooldridge, M., 1998. A roadmap of agent research and development. *Autonomous Agents and Multi-agent Systems*, 1, pp.7–38.

Johnson, S.M., 1954. Optimal two and three stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1, pp.402–452.

Jungwattanakit, J. et al., 2008. Algorithms for flexible flow shop problems with unrelated parallel machines , setup times , and dual criteria. *The International Journal of Advanced Manufacturing Technology*, 37(3-4), pp.1–13.

Kalczynski, P.J. & Kamburowski, J., 2007. On the NEH heuristic for minimizing the makespan in permutation flow shops. *Omega*, 35, pp.53–60.

Karger, D.R., Phillips, S.J. & Torng, E., 1996. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20, pp.400–430.

Khan, B. & Govindan, K., 2011. A multi-objective simulated annealing algorithm for permutation flow shop scheduling problem. *International Journal of Advanced Operations Management*, 3(1), pp.88–100.

Lawler, E.L. et al., 1993. Secuencing and scheduling: Algorithms and complexity. *Handbooks in Operations Research and Management Science*.

Liu, W., Jin, Y. & Price, M., 2017. A new improved NEH heuristic for permutation flowshop scheduling problems. *International Journal of Production Economics*. Available at: http://dx.doi.org/10.1016/j.ijpe.2017.06.026.

Máñez, R.E., 2015. *Concepciones sobre la Aleatoriedad de estudiantes de Educación Secundaria Obligatoria*. Universidad de Granada.

Mao, W., Kincaid, R.K. & Rifkin, A., 1995. On-line Algorithms for a Single Machine Scheduling Problem. *The Impact of Emerging Technologies on Computer Science and Operations Research*, pp.157–172.

Márquez Delgado, J.E. et al., 2012. Algoritmo genético aplicado a la programación en talleres de maquinado. *Ingenieria Mecanica*, 15(3), pp.201–212.

Martínez Jiménez, Y., 2012. *A Generic Multi-Agent Reinforcement Learning Approach for Scheduling Problems*. Brussels: Vrije Universiteit Brussel.

Martínez, Y. et al., 2011. A Reinforcement Learning Approach for the Flexible Job Shop Scheduling Problem. *Lecture Notes in Computer Science. Proceedings of the fifth International Conference on Learning and Intelligent Optimization(LION5).*, Vol. 6683, pp.pp. 253–262.

Martinez-Jimenez, Y. & Fonseca Reyna, Y.C., 2017. Adapting a Reinforcement Learning Approach for the Flow Shop Environment with sequence-dependent setup time. *Revista Cubana de Ciencias Informaticas*, 11(1). Available at: https://rcci.uci.cu/?journal=rcci&page=article&op=view&path[]=970.

Medina P, Cruz E, H.-R.J., 2008. Works programming in one machine uses a model Integer linear programming. *Scientia et Technica*, XIV(40), pp.111–116.

Megow, N., Uetz, M. & Vredeveld, T., 2006. Models and Algorithms for Stochastic Online Scheduling. *Mathematics of Operations Research*, 31(3), pp.513–525.

Mendez, C. et al., 2006. State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers & Chemical Engineering*, 30(6-7), pp.913–946.

Mendez-Hernandez, B.M. et al., 2017. Enfoque bi-objetivo basado en Aprendizaje Reforzado para Job Shop scheduling. *Revista Cubana de Ciencias Informaticas*, 11, pp.175–188. Available at: http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2227-18992017000200013&nrm=iso.

Moniz, S. et al., 2014. Solution Methodology for Scheduling Problems in Batch Plants. *Industrial & Engineering Chemistry Research*, 53.

Moniz, S., Barbosa Póvoa, A.P. & Pinho de Sousa, J., 2013. New general discrete-time scheduling model for multipurpose batch plants. *Industrial & Engineering Chemistry Research*, 52, p.17206−17220.

Moonsri, K., Sethanan, K. & Sangsawang, C., 2015. Metaheuristics for Scheduling Unrelated Parallel Machines with Sequence-Dependent Setup Time and Machine Eligibility. *Special Issue on Logistics and Supply Chain Systems*, 14(4), pp.431–446.

Moreira da Silva, F.A., Moretti, A.C. & Tavares De Azevedo, A., 2014. A Scheduling Problem in the Baking Industry. *Applied Mathematics*, 2014, p.14.

Nagano, M.S., da Silva, A.A. & Lorena, L.A.N., 2012. A new evolutionary clustering search for a no-wait flow shop problem with set-up times. *Engineering Applications of Artificial Intelligence*, 25(6), pp.1114–1120. Available at: http://www.sciencedirect.com/science/article/pii/S0952197612001236.

Najarro, R. et al., 2017. Un Algoritmo Genético Híbrido para la Optimización del Flow Shop Scheduling bajo Restricciones de Entornos Reales. *Enfoque UTE*, 8(5), pp.14–25.

Nawaz, M., Enscore, E. & Ham, I., 1983. A heuristic algorithm for the m-machine, n-job fowshop sequencing problem. *Omega- The International Journal of Management Science*, 11(1), pp.91–95.

Neto, R.F.T., Filho, M.G. & da Silva, F.M., 2015. An ant colony optimization approach for the parallel machine scheduling problem with outsourcing allowed. *J. Intelligent Manufacturing*, 26(3), pp.527–538. Available at: https://doi.org/10.1007/s10845-013-0811-5.

Nhu Binh, H.O. & Joc Cing, T.A.Y., 2005. Evolving Dispatching Rules for solving the Flexible Job-Shop Problem. In *IEEE Congress on Evolutionary Computation (CEC2005)*. pp. 2848–2855.

Nhu Binh HO, J.C.T.A.Y., 2005. Evolving Dispatching Rules for solving the Flexible Job-Shop Problem. *The 2005 IEEE Congress on Evolutionary Computation.*, 3, pp.2848–2855.

Özgüven, C., Yavuz, Y. & Özbakir, L., 2012. Mixed integer goal programming models for the fexible job-shops cheduling problems with separable and non-separable sequence dependent setup times. *Applied Mathematical Modelling*, 36(2), pp.846–858.

Palacio, J.C. et al., 2017. Algoritmos para problemas de secuenciación de tareas en ambientes online. *Investigación Operacional*, 38(4), pp.400–406.

Peeters, M., 2008. *Solving Multi-Agent Sequential Decision Problems Using Learning Automata*. Vrije Universiteit Brussel.

Pinedo, M., 1995. Scheduling: theory, algorithms and systems. *Englewood cliffs, NJ: PrenticeHall.*

Pinedo, M.L., 2008. *Scheduling: Theory, Algorithms, and Systems* 3rd ed., Springer Publishing Company, Incorporated.

Puris, A. et al., 2007. Two-Stage ACO to Solve the Job Shop Scheduling Problem. In L. Rueda, D. Mery, & J. Kittler, eds. *Progress in Pattern Recognition, Image Analysis and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 447–456.

Rajendran, C. & Ziegler, H., 2004. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155(2), pp.426–438. Available at: http://www.sciencedirect.com/science/article/pii/S0377221702009086.

Ramasesh, R., 1990. Dynamic job shop scheduling: A survey of simulation research. *Omega*, 18, pp.43–57.

Ramezanian, R., Aryanezhad, M. & Heydar, M., 2010. A Mathematical Programming Model for Flow Shop Scheduling Problems for Considering Just in Time Production. *International Journal of Industrial Engineering & Production Research*, 21(2), pp.97–104.

Recabal Guiraldes, P., 2009. *Aprendizaje Reforzado Orientado a la Toma de Decisiones en el Fútbol Robótico*. Universidad de Chile.

Rochette, R. & Sadowski, R.P., 1976. A statistical comparison of the performance of simple dispatching rules for a particular set of job shops. *International Journal of Production Research*, 14, pp.63–75.

Rodriguez, F.J. et al., 2013. An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. *Computers & Operations Research*, 40, pp.1829–1841.

Ruiz, R., Şerifoğlu, F.S. & Urlings, T., 2008. Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, 35(4), pp.1151–1175. Available at: http://www.sciencedirect.com/science/article/pii/S0305054806001560.

Ruiz, R. & Vázquez-Rodríguez, J.A., 2010. The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1), pp.1–18.

Shahzad, A. & Mebarki, N., 2016. Learning Dispatching Rules for Scheduling : A Synergistic View Comprising Decision Trees , Tabu Search and Simulation. *Computers*. Available at: www.mdpi.com/journal/computers.

Shen, W., 2002. Distributed Manufacturing Scheduling Using Intelligent Agents. *IEEE Intelligent Systems*, 17, pp.88–94.

Sherstov, A.A. & Stone, P., 2005. Function approximation via tile coding: Auto-mating parameter choice. *Lecture Notes in Computer Science: Abstraction, Reformulation and Approximation*, pp.194–205.

Sheskin, D., 2006. *Handbook of parametric and nonparametric statistical procedures*, Chapman & Hall: London/West Palm Beach.

Singhal, E., Singh, S. & Dayma, A., 2012. An Improved Heuristic for Permutation Flow Shop Scheduling (NEH ALGORITHM). *International Journal Of Computational Engineering Research*, 2(6), pp.95–100.

Sivakumar, A.I., 1999. Optimization of a cycle time and utilization in semiconductor test manufacturing using simulation based, on-line, near-real-time scheduling system. In *Winter Simulation Conference*.

Snyman, S. & Bekker, J., 2017. Real-time scheduling in a sensorised factory using cloud-based simulation with mobile device access. *South African Journal of Industrial Engineering*, 28(2017), pp.161–169.

Staeblein, T. & Aoki, K., 2014. Planning and scheduling in the automotive industry : A comparison of industrial practice at German and Japanese makers. *Intern. Journal of Production Economics*, pp.1–14. Available at: http://dx.doi.org/10.1016/j.ijpe.2014.07.005.

Stefansson, H. & Shah, N., 2005. Multi-scale Planning and Scheduling in The Pharmaceutical Industry. In *European Symposium on Computer Arded Process Engineering*. p. 6.

Suarez, J. & Martinez, Y., 2008. Propuesta de solución al problema de secuenciación en múltiples máquinas utilizando la metaheurística ACO. In *IV Conferencia Científica de la Universidad de Ciencias Informáticas, UCIENCIA 2008*. Havana, Cuba.

Sutton, R.S. & Barto, A.G., 1998a. *Reinforcement Learning: An Introduction*, The MIT Press.

Sutton, R.S. & Barto, A.G., 1998b. *Reinforcement Learning: An Introduction*, The MIT Press.

Taillard, E., 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64, pp.278–285.

Taillard, E., 1990. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1), pp.65–74.

Tu, K., Hao, Z. & Chen, M., 2006. PSO with Improved Strategy and Topology for Job Shop Scheduling. *Advances in Natural Computation*, pp.146–155. Available at: http://www.springerlink.com/index/9545871LV8311620.pdf.

Urlings, T., Ruiz, R. & Stützle, T., 2010. Shifting representation search for hybrid flexible flowline problems. *European Journal of Operational Research*, 207(2), pp.1086–1095. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0377221710003978.

Varadharajan, T. & Rajendran, C., 2005. A multi-objective simulated-annealing algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs. *European Journal of Operational Research*, 167(772-795).

Vredeveld, T., 2012. Stochastic online scheduling. *Computer Science — Research and Development*, 27(3), pp.181–187.

Vukovic, D., Nesic, Z. & Nusev, S., 2017. Operational planning and scheduling in manufacturing. In *10th International Scientific Conference "Science and Higher Education in Function of Sustainable Development."*pp. 13–17.

Wang, L. & Shen, W., 2007. Process Planning and Scheduling for Distributed Manufacturing L. Wang & W. Shen, eds. *Springer*, p.429.

Wang, L., Zhang, L. & Zheng, D.-Z., 2006. An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. *Computers & Operations Research*, 33(10), pp.2960–2971. Available at: http://www.sciencedirect.com/science/article/pii/S0305054805000845.

Watkins, C.J.C.H. & Dayan, P., 1992. Technical note -- Q-learning. *Machine learning*, 8(3), pp.279–292.

Yang, Y., Chen, Y. & Long, C., 2016. Flexible robotic manufacturing cell scheduling problem with multiple robots. *International Journal of Production Research*, 54(22), pp.6768–6781.

Yenisey, M.M. & Yagmahan, B., 2014. Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega*,

45, pp.119–135. Available at: http://www.sciencedirect.com/science/article/pii/S0305048313000832.

Zahmani, M.H., Atmani, B. & Bekrar, A., 2015. Efficient dispatching rules based on data mining for the single machine scheduling problem. *Computer Science & Information Technology (CS & IT)*, pp.199–208.

Zhang, W., 1996. *Reinforcement Learning for Job Shop Scheduling*. Oregon State University.

Zhang, W. & Dietterich, T.G., 1995. A Reinforcement Learning Approach to Job SHop Scheduling. In *International Joint Conference on Artificial Intelligence*. pp. 1114–1120. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.8.3850&rep=rep1&type=pdf.

Zobolas, G.I., Tarantilis, C.D. & Ioannou, G., 2009. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, 36(4), pp.1249–1267. Available at: http://www.sciencedirect.com/science/article/pii/S0305054808000129.