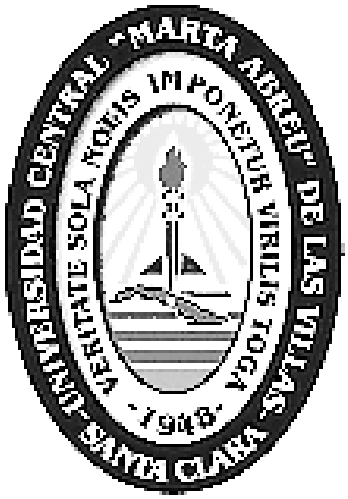
Universidad Central "Marta Abreu" de Las Villas

Facultad Matemática, Física y Computación

Centro de Estudios de Informática

Departamento de Ciencia de la Computación



Enfoque basado en Aprendizaje Reforzado para problemas de secuenciación de tareas tipo *Flow Shop.*

Trabajo de Diploma

Autor: Juan Manuel Bermúdez Cabrera

Tutores: Dra. Yailen Martínez Jiménez

MSc. Yunior Fonseca Reyna

Consultante: MSc. Beatriz Martínez Méndez

Santa Clara, 2014.

Dictamen

El que suscribe, Juan Manuel Bermúdez Cabrera, hago constar que el trabajo titulado ***Enfoque basado en Aprendizaje Reforzado para problemas de secuenciación de tareas tipo Flow Shop*** fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de los estudios de la especialidad de Licenciatura en Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

_____

Firma del Autor

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

_____ _____

Firma del Tutor Firma del Jefe del Laboratorio

A mis padres, guía y estímulo en cada uno de mis pasos.

Agradecimientos

- ✓ *A mis padres por haberme dado la vida e inculcado el respeto, la responsabilidad y los deseos de superación, porque hicieron todo lo posible para que yo pudiera lograr mis sueños, por motivarme y darme la mano cuando sentía que el camino se terminaba, a ustedes por siempre mi corazón y agradecimiento.*
- ✓ *A Suirita, por darme su amor y cariño, gracias simplemente por existir y ser parte de mi vida.*
- ✓ *A mi novia Lili, por su paciencia y apoyo durante el tiempo que duró el desarrollo de este trabajo y principalmente por su infinito amor.*
- ✓ *A Odalis y Abel, por ser unos segundos padres para mí y por su constante preocupación y apoyo.*
- ✓ *A mis tutores Yailen y Yunior, por hacer posible esta tesis y ser una guía y soporte en su elaboración.*
- ✓ *A mis profesores, que influyeron con sus lecciones y experiencias en formarme como una persona de bien y preparada para los retos que impone la vida, a todos y cada uno de ellos les dedico estas páginas.*

De forma general a todas aquellas personas que me ayudaron a concluir este trabajo.

"Son los problemas sin resolver, no los resueltos, los que mantienen activa la mente"

Erwin Guido Kolbenheyer

RESUMEN

El problema de secuenciación tipo *Flow Shop* define un grupo importante de problemas de secuenciación en el campo de planificación de la producción. El problema considerado aquí consiste en encontrar una permutación de tareas que pueda ser procesada secuencialmente en un número de recursos con el objetivo de minimizar el tiempo de completamiento de todas las tareas, conocido en la literatura como *makespan* o C_{max}. Este problema típico de la optimización combinatoria es NP-*Hard* y puede ser encontrado en ambientes de manufactura, donde existen máquinas-herramientas convencionales y se fabrican diferentes tipos de piezas que pueden, en dependencia del escenario, presentar una misma ruta o no. La siguiente investigación presenta un algoritmo de Aprendizaje Reforzado conocido como Q-*Learning* para resolver problemas de tipo *Flow Shop*. Este algoritmo se basa en aprender una función acción-valor que proporciona la utilidad esperada de tomar una acción dada en un estado determinado. Para validar la calidad de las soluciones de este algoritmo se utilizan problemas de la literatura especializada y los resultados obtenidos son comparados con los resultados óptimos reportados. Además se propone una herramienta para trabajar con estos tipos de problemas y que puede servir como entorno de trabajo integrado para resolver otros tipos de problemas de secuenciación de tareas usando Aprendizaje Reforzado.

Palabras Clave: *Flow Shop*, Q-*Learning*, NEH, *makespan, scheduling*.

ABSTRACT

The Flow Shop Scheduling Problem outlines an important group of scheduling problems in the field of production planning. The problem considered here is to find a permutation of tasks that can be processed sequentially on a number of resources to minimize the completion time of all tasks, known in the literature as makespan or C_{max}. This typical combinatorial optimization problem is NP-*Hard* and can be found in manufacturing environments, where there are conventional machine tools and different types of parts that can, depending on the stage, presenting the same route or not made. The following research presents a Reinforcement Learning algorithm known as Q-Learning to solve problems of the Flow Shop category. This algorithm is based on learning action-value function that gives the expected utility of taking a given action in a given state. To validate the quality of the solutions, test cases of the specialized literature are used and the results obtained were compared with the reported optimal results. Also a tool for working with these kinds of problems and can serve as integrated working environment to solve other scheduling problems using Reinforced Learning is proposed.

Keywords: *Flow Shop*, Q-*Learning*, NEH, *makespan, scheduling*.

TABLA DE CONTENIDOS

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

La secuenciación de tareas (*scheduling*) es un proceso de toma de decisiones que se usa de forma regular en aquellas situaciones donde diversas tareas tienen que ser ejecutadas en un conjunto de recursos limitados para optimizar ciertos objetivos y satisfacer determinadas restricciones. Los problemas de secuenciación son de aplicación en las organizaciones y en la industria y en consecuencia tienen un fuerte impacto económico y social. Algunos ejemplos son: la secuenciación de las puertas de los aeropuertos, de las tripulaciones de los aviones y de los procesos de manufactura. Este trabajo se va a centrar en este último tipo de secuenciación, donde el proceso de construcción de secuencias juega un papel importante, debido a que puede tener un fuerte impacto en la productividad de la empresa. En el entorno competitivo actual, una secuenciación efectiva se ha convertido en una necesidad para sobrevivir en el mercado, a la vez que las empresas tienen que usar sus recursos de una manera eficiente (Pinedo, 2008).

El problema de secuenciación de tareas tipo *Flow Shop* consiste en la programación temporal en m máquinas de un conjunto de trabajos, compuesto cada uno por m operaciones, teniendo en cuenta que cada máquina solamente puede ejecutar una tarea simultáneamente y que todos los trabajos pasan por todas las máquinas en el mismo orden. Se busca aquella solución que dé lugar a un tiempo total de ejecución (*makespan*) mínimo, aunque pueden existir otras funciones objetivo como la minimización de la suma de tiempos de espera o la suma de los tiempos de ejecución. El problema, para minimización del *makespan* es NP-Completo, sólo algunos casos especiales ($m \leq 2$) pueden ser resueltos eficientemente (Brucker, 2007).

Se han propuesto muchos enfoques aproximados para encontrar secuencias cercanas al óptimo en un tiempo razonable, como es el uso de Algoritmos Genéticos, Recocido Simulado, Búsqueda Tabú y métodos de Búsqueda Local. Muchos problemas de secuenciación de tareas sugieren una formulación natural como tareas distribuidas de toma de decisiones, por tanto, el empleo de Sistemas Multi-Agente (SMA) representa un enfoque evidente, los cuales pueden aprender a través del Aprendizaje Reforzado (AR). Además, dada la conocida complejidad de este tipo de problemas, la aplicación de enfoques descentralizados en su solución puede resultar una opción prometedora (Gabel, 2009).

El AR es aprender qué hacer (cómo asociar situaciones a acciones) de tal forma que se maximice una señal numérica de recompensa. El aprendiz no sabe de antemano qué acciones tomar, al

igual que en la mayoría de las formas de Aprendizaje Automático (*Machine Learning*), en cambio debe descubrir que acciones producen la mejor recompensa probándolas. En los casos más interesantes las acciones no sólo afectan la recompensa inmediata sino también las situaciones siguientes y por tanto las recompensas futuras (Barto, 1998).

Teniendo en cuenta lo expuesto con anterioridad se plantea el siguiente **problema de investigación:**

La complejidad de los problemas de tipo *Flow Shop* así como el modelado de algunas restricciones hace que estos métodos no sean los más apropiados para problemas reales, esto ilustra la necesidad de investigar el uso de nuevas técnicas, por lo que se propone un enfoque genérico basado en Agentes Inteligentes y AR.

Para dar solución al problema científico se plantea como **objetivo general** de esta tesis:

Desarrollar una herramienta que permita solucionar problemas de secuenciación de tareas en ambientes de manufactura usando Agentes Inteligentes y AR.

El cual se desglosa en los siguientes **objetivos específicos**:

- Modelar el problema de secuenciación de tareas tipo *Flow Shop* para resolverlo utilizando AR y Agentes Inteligentes.
- Implementar un algoritmo basado en AR para resolver el problema tipo *Flow Shop*.
- Comparar los resultados del algoritmo con los reportados por otras técnicas de la literatura en términos de la calidad de las soluciones, específicamente el *makespan*.
- Integrar en una herramienta el algoritmo implementado con el ambiente de trabajo tipo *Job Shop*.

Para la solución al problema de investigación y a los objetivos específicos de esta tesis se dará respuesta a las siguientes **preguntas de investigación**:

- ¿Cómo modelar el problema de secuenciación de tareas tipo *Flow Shop* para resolverlo utilizando AR?
- ¿Se pueden obtener resultados comparables a los obtenidos por otras técnicas de la literatura para ambientes tipo *Flow Shop*?

- ¿Cómo integrar los algoritmos que resuelven problemas de secuenciación con distintos ambientes de trabajo?

En el ambiente competitivo actual, un proceso de secuenciación efectivo se convierte una necesidad en aras de sobrevivir en el mundo del mercado, ya que las compañías tienen que cumplir con las fechas de entrega y usar sus recursos de forma eficiente (Pinedo, 2008). Los problemas de secuenciación son usualmente NP-Completo, de ahí que la investigación desarrollada por la comunidad científica a menudo se concentra en problemas de optimización que son en realidad una versión simplificada de lo que ocurre realmente. Esto permite el uso de enfoques sofisticados que en muchos casos garantizan encontrar soluciones óptimas, sin embargo, la exclusión de restricciones del mundo real daña la aplicabilidad de dichos métodos. Lo que la industria necesita son sistemas optimizados de secuenciación que se ajusten a las condiciones exactas de la planta de producción y que generen buenas soluciones en poco tiempo (Urlings et al., 2010).

La tesis está estructurada en tres capítulos, en el CAPÍTULO 1 se realiza un estudio de los problemas de secuenciación de tareas comenzando por sus aspectos fundamentales. Luego se exponen sus diversas clasificaciones, haciendo énfasis en los problemas tipo *Permutation Flow Shop*. Posteriormente se muestra una breve reseña de los diferentes métodos que existen para resolver los problemas de secuenciación. Por último, se presentan las ideas fundamentales detrás del AR y la teoría sobre los SMA. Se introducen los conceptos de agente y de SMA; se explica cómo los agentes pueden interactuar con el ambiente en aras de aprender a resolver una tarea específica y se introducen tres posibles estrategias de selección de acciones. La última sección del capítulo define los conceptos fundamentales del AR haciendo énfasis en el algoritmo Q-*Learning*. Seguidamente, en el CAPÍTULO 2 se hace una descripción de las instancias del problema y se especifican los principales aspectos que se tuvieron en cuenta a la hora de desarrollar el método, por último se propone un algoritmo basado en AR y se describe mediante un ejemplo. Posteriormente, en el CAPÍTULO 3 se describen en el marco experimental las formas de evaluar los resultados obtenidos. Se comparan los resultados para el conjunto de instancias de *Taillard*, con algoritmos presentados en la literatura. El documento culmina con las conclusiones y las recomendaciones del autor.

CAPÍTULO 1: Problemas de secuenciación.

La secuenciación de tareas es un problema de optimización que se presenta con frecuencia en sistemas de producción convencionales automatizados. Este problema involucra la toma de decisiones con respecto a la mejor asignación de recursos a procesos en los cuales se tienen diferentes tipos de restricciones. Este capítulo aborda todo lo referente a la base teórica que fundamenta esta investigación. Se hace una revisión del marco teórico para determinar las técnicas de optimización que han sido utilizadas para resolver problemas de secuenciación de tareas en configuraciones de tipo *Flow Shop*.

1.1 Introducción a los problemas de secuenciación.

Los problemas de *scheduling*, a lo largo de los años, han capturado el interés de muchos investigadores en numerosas comunidades de investigación: Administración, Ingeniería Industrial, Investigación de Operaciones (IO), Inteligencia Artificial (IA), entre otras (Grossman, 2005).

Esta programación es una prioridad dentro del plan de acción de cualquier tipo de empresa porque de ello depende, en buena parte, la rentabilidad de la misma. Dichos problemas deben ser resueltos en una amplia gama de aplicaciones tales como: programación de despacho de vuelos en los aeropuertos, programación de líneas de producción de una fábrica, programación de cirugías en un hospital, reparación de equipos o maquinarias en un taller, entre otros (Toro et al., 2006).

El problema fundamental consiste en encontrar una programación adecuada que pueda reducir significativamente los costos de producción y los tiempos de procesamiento permitiendo cumplir con los compromisos de entrega en un período razonable. De no hacerse inteligentemente, esto puede representar un costo de oportunidad o pérdida económica para cualquier empresa. Por lo tanto, es importante encontrar el orden en el cual se debe de procesar todas las tareas con el fin de reducir al mínimo el tiempo de terminación.

De manera informal, *scheduling* se define como un proceso de optimización que asigna recursos limitados en el tiempo entre tareas que se pueden ejecutar secuencialmente o en paralelo. Esta asignación tiene que cumplir un conjunto de restricciones que reflejan las relaciones temporales entre las tareas y las limitaciones de capacidad del conjunto de recursos compartidos. La

asignación también afecta la optimalidad de una secuenciación respecto a diferentes criterios (Martínez, 2012). Una **tarea** representa un proceso que utiliza **recursos** para producir bienes o proveer servicios. Las tareas tienen tiempo de inicio, tiempo de terminación y duración, que especifican el período durante el que tuvo lugar su ejecución. Las dos clases más importantes de **restricciones** son restricciones temporales y restricciones de recursos (Davenport and Beck, 2000).

En otras palabras, el problema puede ser definido como un conjunto de trabajos que tienen que ser ejecutados en un conjunto de máquinas, con el objetivo de encontrar la mejor secuencia, es decir, una asignación de trabajos a intervalos de tiempo en las máquinas de forma tal que el objetivo deseado sea minimizado.

Los problemas pueden ser clasificados teniendo en cuenta distintas características, por ejemplo, el número de máquinas (una máquina, máquinas paralelas), las características de los trabajos (se pueden interrumpir las actividades o no) y así sucesivamente.

Estos tipos de problemas son típicamente NP-*Hard* (Brucker, 2007, Garey et al., 1976), siendo la secuenciación en los ambientes de manufactura uno de los problemas de *scheduling* más difíciles (Shen, 2002).

1.2 Conceptos básicos sobre *scheduling*.

Dentro de la teoría de *scheduling* se pueden distinguir un gran número de problemas. En estos problemas se tiene un conjunto de n trabajos $J_i(i = 1, \dots, n)$ que han de ser procesados sobre un conjunto de m recursos o máquinas físicas $M_j(j = 1, \dots, m)$. Una secuenciación consiste en encontrar para cada trabajo un tiempo o un intervalo de tiempos en los que este pueda procesarse en una o varias máquinas. El objetivo es encontrar una secuencia sujeta a una serie de restricciones que optimice una o varias funciones objetivo (Brucker, 2007). Las secuenciaciones pueden ser representadas mediante diagramas de Gantt como se muestra en la Figura . Los diagramas de Gantt pueden ser orientados a las máquinas Figura (a) o a los trabajos Figura (b).

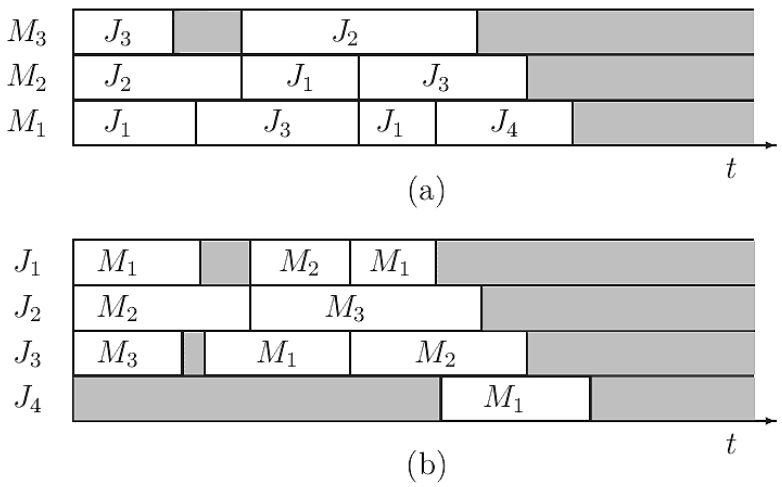


Figura : Diagramas de Gantt orientados a las máquinas y a los trabajos.

En general, en un problema de *scheduling* intervienen los siguientes elementos:

- Trabajos
- Operaciones
- Máquinas
- Tipo de *scheduling* (patrón de flujo)
- Objetivo

Cualquier problema de *scheduling* consta de uno o más trabajos. Un trabajo es el término usado para designar un único elemento o una serie de elementos que tienen que ser procesados en las distintas máquinas. Los trabajos, en la mayoría de los casos, tienen restringido su tiempo de comienzo (*release date*: tiempo a partir del cual puede comenzar la ejecución del trabajo), y su tiempo de terminación (*due date*: tiempo máximo permitido para finalizar la ejecución del trabajo). Si se trata de fabricación por lotes se necesita conocer, además, el número de unidades a fabricar de ese lote. Cada trabajo, a su vez, está compuesto de una o más operaciones. Se denomina operación al procesamiento de un trabajo sobre una máquina en particular, y por lo tanto, constituyen las unidades elementales de procesamiento. Las operaciones pertenecientes a un mismo trabajo guardan un orden de precedencia conocido a priori (restricciones de precedencia o tecnológicas), que debe respetarse, pudiendo haber variaciones en cuanto a posibles solapamientos entre las operaciones de un mismo trabajo, o estar separadas por intervalos de tiempo amplios. El tiempo de procesamiento de cada operación o duración puede

ser fija, variable en un intervalo o estar formado por un conjunto de valores/intervalos disjuntos. Además, en un problema de *scheduling* pueden existir operaciones de distintos tipos, que condicionarán su procesamiento en un determinado tipo de máquina (Fonseca, 2013).

De forma general, la definición de problemas de *scheduling* incluye un conjunto de variables y propiedades. A continuación se mencionan las más estudiadas:

- m Cantidad de máquinas.
- n Cantidad de trabajos.
- M_j Máquina j, donde $j = \{1, \dots, m\}$.
- J_i Trabajo i, donde $i = \{1, \dots, n\}$.
- C_i Tiempo en que se completó el trabajo i.
- o_{ij} j-ésima operación del trabajo i.
- s_{ij} Tiempo de inicio de la operación j del trabajo i.
- c_{ij} Tiempo de finalización de la operación j del trabajo i.
- p_{ij} Tiempo de procesamiento de la operación j del trabajo i.
- r_i Fecha de liberación (*release date*) del trabajo i.
- d_i Fecha deseada para la terminación (*due date*) del trabajo i.
- C_{max} Máximo tiempo de completamiento (*makespan*) de todos los trabajos.
- w_i Peso o ponderación (*weight*) asociado al trabajo i.

En la Figura se muestra un ejemplo de un diagrama de tiempo para la operación o_{ij}, donde se señalan algunas de las propiedades anteriormente mencionadas.

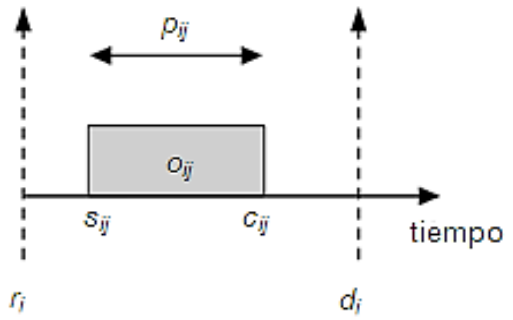


Figura : Diagrama de tiempo de una operación.

Por otro lado es necesario establecer el tipo de *scheduling* que se quiere realizar, lo que determinará el patrón de flujo del problema. Finalmente, es importante definir la función objetivo que se desea optimizar, siendo las más comunes el máximo tiempo de finalización (*makespan*) $\max_{i=\{1,\dots,n\}} C_i$, tiempo total de flujo (*total flow time*) $\sum_{i=1}^{n} C_i$, y tiempo total de flujo ponderado (*weighted total flow time*) $\sum_{i=1}^{n} w_i C_i$ (Brucker, 2007). Otras funciones objetivo dependen de la fecha deseada de terminación (*due date*) d_i, las cuales se asocian a los trabajos J_i. Se definen para cada trabajo J_i los siguientes atributos:

- *Lateness*: Anticipación con la que terminó el trabajo respecto a su fecha de entrega. Se define como $L_i = C_i - d_i$.
- *Tardiness*: Retraso con el que terminó el trabajo con respecto a su fecha de entrega. Se define como $T_i = max\{0, C_i - d_i\}$.

Se puede usar entonces otras funciones objetivo basadas en estas características relacionadas con las fechas de entrega:

- Tardanza total: $\sum_{i=1}^{n} T_i$
- Tardanza máxima: $max\{T_1, \dots, T_n\}$

1.3 Clasificación de los problemas de *scheduling*.

Se pueden encontrar muchas clasificaciones de los problemas de *scheduling*. Por ejemplo, se podrían caracterizar en función de parámetros como: deterministas frente a estocásticos (duraciones de las tareas fijas o probabilísticas), requerimientos de tiempo real (restricciones de

tiempo real estricto frente a procesos *offline*), presencia de restricciones sobre los recursos, objetivos del *scheduling* (minimizar el *makespan*, el costo,…), etc.

En los problemas de *scheduling*, la capacidad de cada máquina (o recurso) está definida sobre un cierto número de intervalos temporales y el problema consiste en cubrir las demandas de recursos de las operaciones a lo largo del tiempo, sin exceder sus capacidades disponibles. En estos problemas se conoce a priori qué máquina va a utilizar exactamente cada una de las operaciones. Se pueden distinguir entonces cuatro patrones de flujo dependiendo del uso que hagan las operaciones de los recursos:

- *Open Shop*: No existe ninguna restricción en cuanto al orden de utilización de las máquinas por las operaciones de cada uno de los trabajos.
- *Job Shop*: Cada trabajo debe utilizar los recursos siguiendo el orden determinado (restricciones tecnológicas); n trabajos deben ser procesados, una sola vez, por m máquinas y durante un tiempo dado.
- *Flow Shop*: Todos los trabajos utilizan los recursos en el mismo orden. Es un caso particular del *Job Shop*.
- *Permutation Flow Shop*[1]: Todos los trabajos utilizan los recursos en el mismo orden. Todas las máquinas procesan los trabajos en el mismo orden. Es un caso particular del *Flow Shop*.

Además, se puede imponer la restricción adicional de que cada una de las tareas pueda interrumpirse y ser reanudada más tarde, después de que la máquina le vuelva a ser asignada. En este caso se estaría hablando de *scheduling* con reemplazo (*preemptive scheduling*). Si la ejecución de cada tarea tiene que realizarse en su totalidad antes de abandonar el recurso asignado, se estaría ante un tipo de *scheduling* denominado sin reemplazo (*non-preemptive scheduling*) (Alfonso, 2001).

[1]En este trabajo únicamente se va a centrar en problemas del tipo *Permutation Flow Shop*; además se considerará *scheduling* sin reemplazo.

1.4 Problemas de tipo *Permutation Flow Shop*.

Uno de los problemas de secuenciación estudiados más a profundidad son los de tipo *Flow Shop*, en estos problemas, un conjunto $N = \{1, \dots, n\}$ de trabajos independientes tiene que ser procesado en un conjunto $M = \{1, \dots, m\}$ de máquinas. Cada trabajo $i, i \in N$, requiere un tiempo de procesamiento fijo y no negativo p_{ij} en cada máquina $j, j \in M$. En los problemas tipo *Flow Shop* los n trabajos tienen que ser procesados en las m máquinas, siguiendo el mismo orden. Esto significa que los trabajos siguen el mismo camino tecnológico comenzando por la máquina 1 y terminando en la m. El objetivo es encontrar una secuencia para procesar los trabajos de tal forma que optimice el *makespan*[2]. Una simplificación común en los problemas *Flow Shop* consiste en evitar el adelanto de trabajos en la secuencia, lo que implica que la secuencia de procesamiento de los trabajos en la primera máquina es mantenida a través del resto de las máquinas. El problema resultante es llamado *Permutation Flow Shop*, existen $n!$ secuencias posibles y se sabe que pertenece a la clase NP-Completo para el caso de minimización del *makespan*; esto significa que cualquier algoritmo de solución emplea un tiempo de ejecución que aumenta, en el peor de los casos, exponencialmente con el tamaño del problema (Ruiz and Stützle, 2007).

Un ejemplo de este tipo de problema de secuenciación es el que se muestra en la Figura , donde se tiene un grupo de tres trabajos que deben ser procesados en un conjunto de cuatro máquinas, cada tarea tiene el mismo camino tecnológico a través de los recursos, es decir, cada uno de los trabajos debe ser procesado primero en la máquina 1, luego en la máquina 2, y así sucesivamente hasta llegar a la máquina 4.

[2]En el transcurso de este trabajo sólo se tendrá en cuenta el *makespan* como función objetivo a optimizar, aunque existen otras (*total flow time*,…).

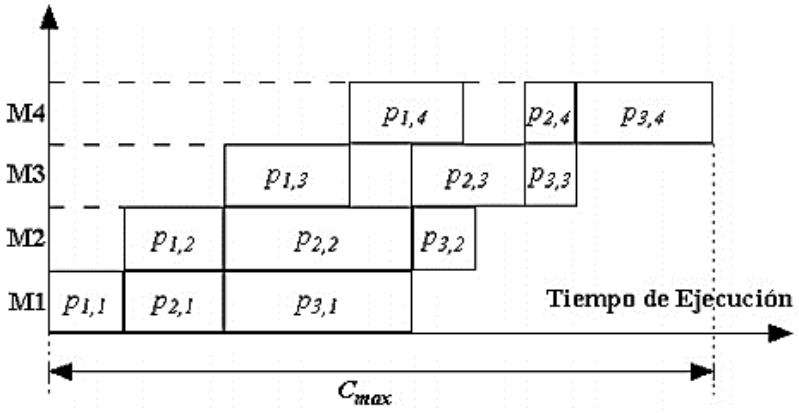


Figura : Secuenciación de 3 trabajos en 4 máquinas en un ambiente *Permutation Flow Shop*.

1.5 Métodos de solución para los problemas de *scheduling*.

En las últimas décadas se han propuesto un gran número de enfoques para modelar y solucionar el problema de secuenciación de tareas con diferentes grados de éxito. El área de IO ofrece diferentes enfoques matemáticos para resolver los problemas de secuenciación, por ejemplo Programación Lineal, Programación Dinámica y métodos de Ramificación y Acotamiento. Cuando el tamaño del problema no es muy grande estos métodos pueden proveer soluciones óptimas en una cantidad de tiempo razonable.

La mayoría de los problemas de *scheduling* del mundo real son NP-*Hard*, y su tamaño por lo general no es pequeño, es por eso que los métodos de optimización fallan en obtener soluciones óptimas en un período de tiempo razonable. Es aquí donde los métodos heurísticos se convierten en el centro de atención. Estos pueden obtener buenas soluciones de una forma eficiente. La IA se convirtió en una importante herramienta para resolver problemas de secuenciación reales en los inicios de la década de los ’80. Algunos de los métodos que han sido utilizados son Recocido Simulado, Búsqueda Tabú, Algoritmos Genéticos, etc. (Martínez, 2012).

Generalmente, los métodos para resolver los problemas *Permutation Flow Shop* se agrupan en dos categorías: heurísticas constructivas y de mejora. Las heurísticas constructivas van confeccionando paso a paso la secuencia de trabajos, basándose en reglas de decisión específicas. Por otro lado, las heurísticas de mejora crean mejores soluciones a través de reglas específicas, comenzando con una solución válida existente, en la mayoría de los casos, encontrada por las técnicas constructivas.

1.5.1 Heurísticas constructivas.

El algoritmo propuesto por (Johnson, 1954) es un método clásico que encuentra el óptimo para el problema de secuenciar n trabajos en dos máquinas. Además de este método, que presenta muchas limitaciones, han surgido varios algoritmos heurísticos que tratan de resolver el problema general de encontrar la secuencia óptima para procesar n trabajos en m máquinas.

Uno de ellos es el algoritmo CDS de (Campbell et al., 1970), que divide las m máquinas en dos grupos que son considerados como dos máquinas virtuales. Entonces el problema es resuelto aplicando el algoritmo de *Johnson*. Luego se escoge consecutivamente la mejor solución entre las $m-1$ secuencias de procesamiento. Muchos otros métodos heurísticos como los propuestos por (Palmer, 1965), (Gupta, 1971) y (Hundal and Rajgopal, 1988) hacen uso de un *slope index* (índice de inclinación) que se le asigna a cada trabajo. Estos algoritmos heurísticos ordenan la lista de trabajos usando dicho peso como clave de ordenamiento para crear una secuenciación válida.

El algoritmo NEH creado por (Nawaz et al., 1983) es reconocido por (Taillard, 1990) como uno de los métodos heurísticos eficientes en este campo. NEH no se basa ni en el algoritmo de *Johnson* ni en técnicas de asignación de pesos. Primero que todo, el algoritmo calcula el tiempo de completamiento de cada trabajo individualmente. Luego, los trabajos se ordenan de forma descendente según estos valores. Se seleccionan los dos primeros trabajos y se escoge la mejor de las dos variantes posibles (J_1, J_2 o J_2, J_1). Luego le sigue el tercer trabajo de la lista ordenada. Para este trabajo existen tres alternativas posibles dentro de la secuencia de procesamiento. Al igual que en el caso anterior se selecciona la alternativa con menor tiempo de completamiento (*makespan*). De esta forma se va introduciendo cada trabajo $J_i (2 < i < n)$ en la posición más favorable de la secuencia $J_1, J_2, \dots, J_{i-1}$ ya formada, hasta que todos los trabajos sean secuenciados (Ancău, 2012).

El algoritmo SPIRIT de (Widmer and Hertz, 1989) es también una heurística constructiva. Este método está basado en una analogía con el Problema del Viajero Vendedor. A través de una fórmula propuesta por los autores, el tiempo productivo de trabajos relacionados se convierte en la distancia entre ellos. Por lo tanto, el problema de secuenciar los trabajos se transforma en el de hallar un recorrido que pase por todos ellos con una distancia transitada mínima.

Varios algoritmos con heurísticas constructivas como (Rajendran and Ziegler, 2004) y (Gajpal and Rajendran, 2006) usan técnicas de inserción como el principio de las Colonias de Hormigas para diseñar la secuencia óptima de trabajos.

El uso de SMA y AR en los últimos años se ha empleado de igual manera para resolver este problema. El AR es una rama del Aprendizaje Automatizado que ha sido utilizado en la solución de problemas de optimización combinatoria, obteniendo buenos resultados (Latorre, 2011, Stéfan, 2003).

1.5.2 Heurísticas de mejora.

A diferencia de las heurísticas constructivas, los algoritmos con heurísticas de mejora parten de una secuencia de procesamiento previamente elaborada y tratan de optimizarla aplicando diferentes procedimientos. Las técnicas se basan fundamentalmente en la permutación de trabajos adyacentes como en (Dannenbring, 1977) con las variantes *Rapid Access with Close Order Search (RACS)* y *Rapid Access with Extensive Search (RAES)*, comenzando ambas a partir de una secuenciación inicial creada por *Rapid Access (RA)*. Otras técnicas como la propuesta por (Ho and Chang, 1991) usan como criterio de optimización algunos tiempos de procesamiento intermedios. En (Agarwal et al., 2006) se utilizan tres heurísticas constructivas muy conocidas para construir una buena solución inicial. Luego las tres heurísticas son modificadas usando un parámetro de peso que perturba la solución inicial para obtener una mejor. El parámetro de peso es ajustado constantemente siguiendo una estrategia de aprendizaje adaptativo (Ancău, 2012).

Es válido mencionar muchas otras heurísticas cuyas técnicas son usadas para resolver otros problemas de optimización combinatoria y se aplican en este caso a los problemas de tipo *Flow Shop*. Entre las más conocidas se encuentran las técnicas basadas en Recocido Simulado (Taillard, 1990), las de Búsqueda Tabú (Nowicki and Smutnicki, 1996) o los Algoritmos Genéticos (Ponnambalam et al., 2001).

1.6 Agentes Inteligentes.

En el campo de las ciencias computacionales, una entidad que pueda actuar en un ambiente de forma autónoma es considerada un agente. Aunque existe un debate expandido acerca del significado exacto de este término, se adopta la definición de (Jennings et al., 1998) ya que es la que mejor se ajusta a este trabajo:

Definición 1 (Agente): Un agente es un sistema computarizado situado en algún ambiente, que es capaz de actuar de manera autónoma y flexible en dicho ambiente en aras de alcanzar sus objetivos.

Si se puede garantizar que un ambiente en específico es fijo, entonces es relativamente fácil diseñar un agente para operar en él. Pero el mundo real no es tan estático, el entorno está constantemente cambiando, la información usualmente no está completa y es por esto que la posibilidad de fallo debe tenerse en cuenta. Un agente reactivo es aquel que mantiene una interacción constante con su ambiente y responde a los cambios que ocurren en él de forma tal que la respuesta aún es útil.

Se quiere que los agentes sean reactivos, pero también que trabajen en función de objetivos a largo plazo, por tanto, es importante mantener un buen balance entre pro-actividad y reactividad. Sin embargo, algunos objetivos sólo pueden ser alcanzados a través de la cooperación con otros agentes y es aquí donde la habilidad social entra a jugar un papel importante. Los agentes deben ser capaces de interactuar con otros agentes situados en el mismo ambiente en aras de cumplir con su objetivo.

Para poder aprender de la experiencia, los agentes pueden ser entrenados, por ejemplo, a través del aprendizaje supervisado, donde se le brindan ejemplos de pares estado-acción, junto con una indicación que dice si la acción fue correcta o incorrecta. El objetivo en el aprendizaje supervisado es inducir una política a partir de los ejemplos de entrenamiento, la cual es suficientemente general para lidiar con ejemplos no vistos. De este modo, el aprendizaje supervisado requiere de un "profesor" que pueda proveer ejemplos correctamente etiquetados. Por su parte, el AR puede ser aplicado a problemas donde el conocimiento no está disponible o es difícil de obtener. No requiere conocimiento previo sobre decisiones correctas o incorrectas, por consiguiente, el agente tiene que explorar activamente su ambiente para observar los efectos de sus acciones, donde por cada acción que tome recibe una señal numérica indicando cuán buena fue (Martínez, 2012). Esta interacción "prueba y error" con el ambiente es más apropiada para el tipo de problema que se resuelve en este trabajo.

1.6.1 Estrategias de selección de acciones.

Uno de los retos que se plantean en el AR y no en otros tipos de aprendizaje es el equilibrio entre la exploración y la explotación. Para obtener una alta recompensa, un agente con AR debe

preferir acciones que se han intentado en el pasado y han demostrado ser eficaces en la producción de recompensa. Pero para descubrir este tipo de acciones, tiene que probar las acciones que no se han seleccionado antes. El agente tiene que explotar lo que ya se conoce con el fin de obtener la recompensa, pero también tiene que explorar con el fin de realizar las mejores selecciones de acciones en el futuro. El dilema es que ni la exploración ni explotación pueden ser perseguidas exclusivamente sin fallar en la tarea. El agente debe experimentar una variedad de acciones y progresivamente favorecer a aquellas que parecen ser mejores. En una tarea estocástica, cada acción debe ser probada muchas veces para obtener una apreciación fiable de su recompensa esperada. El control apropiado del equilibrio entre la exploración y explotación es importante con el fin de construir un método de aprendizaje eficiente (Barto, 1998). A continuación se exponen brevemente tres estrategias de selección muy comunes, *greedy*, ε-*greedy* y *softmax*.

Si el agente decide escoger la mejor de todas las acciones, entones se puede afirmar que está siguiendo un mecanismo de selección avaricioso (*greedy*). Pero escoger siempre la mejor acción puede conducir a un desempeño inferior al óptimo, dependiendo de la variación de las recompensas de las acciones.

Una alternativa a este comportamiento es seguir la estrategia ε-*greedy*. Este método de selección le indica al agente que escoja la mejor acción la mayoría del tiempo, pero que en algunas ocasiones, escoja una acción aleatoria (con igual probabilidad para cada posible acción a en el estado s). El valor de ε determina la probabilidad de escoger una acción aleatoria. Aunque ε-*greedy* es un mecanismo efectivo y muy popular para equilibrar exploración y explotación en el AR, una desventaja es que cuando explora elige de forma equitativa entre todas las acciones, por lo que puede seleccionar entre la peor y la segunda mejor acción con la misma probabilidad (Martínez, 2012).

Una opción es variar la probabilidad de las acciones como una función proporcionada según sus valores estimados, que es lo que hace la estrategia de selección *softmax*. La acción avariciosa aún tiene la mayor probabilidad, pero las otras están ordenadas según sus valores estimados. Esto significa que la probabilidad de escoger la acción a entre las m posibles acciones está dada por la distribución de *Boltzmann*, que es la distribución más común cuando se usa este mecanismo de selección.

$$Pr(a) = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{m} e^{Q_t(b)/\tau}}$$

En esta ecuación τ es un parámetro positivo llamado **temperatura**, que controla cuan avaricioso será el agente, m representa la cantidad de acciones disponibles, y $Q_t(a)$ el estimado de la acción a en el instante de tiempo t. Altas temperaturas provocan que todas las acciones sean (casi) equis-probables. Temperaturas bajas causan una mayor diferencia en la probabilidad de selección de las acciones que difieren en sus valores estimados, en otras palabras, bajas temperaturas hacen que el agente se comporte más avariciosamente (Barto, 1998).

1.7 Sistemas Multi-Agente.

Muchos sistemas biológicos o computarizados pueden ser vistos como múltiples agentes con objetivos comunes. Algunos ejemplos de la biología son las colonias de insectos, grupos de animales, y multitudes humanas. Otros ejemplos incluyen redes de computadoras y enjambres de robots. Aunque los sistemas computacionales se pueden convertir en inteligentes usando técnicas de IA, los agentes individuales a menudo están restringidos en cuanto sus facultades y cuentan con un conocimiento limitado de su ambiente. Por otra parte, el grupo como un todo es capaz de ejecutar tareas más complejas que las que puede realizar un solo agente. Por tanto, los agentes necesitan usar su habilidad social en aras de cumplir sus objetivos generalmente complejos (Mihaylov, 2012). Por ejemplo, una sola hormiga no conoce la localización exacta de la fuente de alimentos, y puede cargar una cantidad limitada de comida. Un grupo de hormigas, por otro lado, a través del esfuerzo colectivo, es capaz de recoger comida para la colonia entera. Tales SMA son comunes en la naturaleza y están ampliamente estudiados en las ciencias computacionales. En (Jennings et al., 1998) se define un SMA como sigue:

Definición 2 (Sistema Multi-Agente): Un SMA es una red de agentes que trabajan juntos para resolver problemas que están más allá de las capacidades individuales o el conocimiento de cada agente.

Existen dos posibles escenarios cuando se trabaja con múltiples agentes, ellos pueden trabajar juntos tratando de alcanzar un objetivo común, o pueden tener sus propios objetivos e intereses que entran en conflicto, lo que significa que se pueden tener ya sea aprendices independientes o en conjunto. Cuando se utilizan aprendices en conjunto se asume que las acciones que toman

los otros agentes pueden ser observadas. Los aprendices independientes no necesitan observar las acciones tomadas por los otros agentes (Martínez, 2012).

1.8 Aprendizaje Reforzado.

La idea de que aprendemos interactuando con nuestro ambiente es probablemente lo primero que se nos ocurre cuando pensamos en la naturaleza del aprendizaje. Cuando un niño juega, mueve sus manos o busca a su alrededor, no tiene un maestro específico, pero si tiene una conexión motora-sensorial con su alrededor. Ejercitar esta conexión produce un caudal de información de causa y efecto, acerca de las consecuencias de las acciones, y de qué hacer para poder alcanzar objetivos. En el transcurso de nuestras vidas, tales interacciones son indudablemente una enorme fuente de conocimiento acerca de todo lo que nos rodea y de nosotros mismos. Ya sea que aprendemos a conducir un auto o a mantener una conversación, nosotros nos mantenemos muy conscientes de cómo nuestro entorno responde a los que hacemos, y buscamos cambiar el resultado a través de nuestro comportamiento. El aprendizaje a partir de la interacción es una idea fundamental cercana a todas las teorías sobre aprendizaje e inteligencia.

El AR es aprender qué hacer (cómo asociar situaciones a acciones) de tal forma que se maximice una señal numérica de recompensa. El aprendiz no sabe de antemano que acciones tomar, al igual que en la mayoría de las formas de Aprendizaje Automático (*Machine Learning*), en cambio debe descubrir que acciones producen la mejor recompensa probándolas. En los casos más interesantes las acciones no sólo afectan la recompensa inmediata sino también las situaciones siguientes y por tanto las recompensas futuras. Estas dos características, búsqueda de prueba y error y recompensa retardada son los aspectos más distintivos del AR (Barto, 1998).

En el modelo estándar de AR, un agente está conectado al ambiente que le rodea a través de percepción y acción como se muestra en la Figura . En cada paso de interacción, el agente observa el estado actual s del ambiente, y selecciona una acción a para cambiar dicho estado. Esta transición genera una señal de refuerzo r, la cual es recibida por el agente. La tarea del agente es aprender una política para seleccionar acciones en cada estado para recibir la mayor recompensa acumulada a largo plazo (Zhang, 1996).

Figura : Modelo estándar de Aprendizaje Reforzado.

De manera formal, el modelo básico de AR consiste en:

- Un conjunto de estados del ambiente S.
- Un conjunto de acciones A
- Un conjunto de "recompensas" en $\mathbb{R}$
- Una función de transición T

En cada instante t, el agente observa el estado $s_t \in S$ y el conjunto de posibles acciones $A(s_t)$. Escoge una acción $a \in A(s_t)$ y recibe del ambiente el nuevo estado s_{t+1} y una recompensa r_{t+1}, esto significa que el agente implementa una asociación entre estados y probabilidades de seleccionar cada posible acción. Esta asociación es la política del agente y se denota π_t, donde $\pi_t(s, a)$ es la probabilidad de seleccionar la acción a en el estado s en el instante t.

La **función de recompensa** define el objetivo en un problema de AR. Asocia cada estado observado (o par estado-acción) del ambiente con un valor numérico, una **recompensa**, indicando la deseabilidad de esa acción en ese estado. El objetivo de un agente de AR es maximizar la cantidad total de recompensa que recibe a largo plazo. La función de recompensa define cuáles son los eventos buenos y malos para el agente.

De la misma forma en que la función de recompensa indica que es "inmediatamente" bueno, la **función de valor** especifica que es bueno a largo plazo. En otras palabras, el valor de un estado es la cantidad total de recompensa descontada que un agente puede esperar acumular en el futuro, comenzando en ese estado. Por ejemplo, un estado puede siempre conducir a bajas recompensas inmediatas, pero aún tener un alto valor porque es normalmente seguido por otros estados que conducen a recompensas altas (Martínez, 2012).

Para detallar más el funcionamiento del AR, la Tabla Tabla : Interacción entre un agente y el ambiente que le rodea. muestra un ejemplo de la interacción entre un agente y el ambiente que le rodea, tomado de (Kaelbling et al., 1996).

Ambiente:	Estás en el estado 65. Tienes 4 acciones posibles.
Agente:	Tomaré la acción 2.
Ambiente:	Has recibido un refuerzo de 7 unidades. Estás ahora en el estado 15. Tienes 2 acciones posibles.
Agente:	Tomaré la acción 1.
Ambiente:	Has recibido un refuerzo de -4 unidades. Estás ahora en el estado 65. Tienes 4 acciones posibles.
Agente:	Tomaré la acción 2.
Ambiente:	Has recibido un refuerzo de 5 unidades. Estás ahora en el estado 44. Tienes 5 acciones posibles.
…	

Tabla : Interacción entre un agente y el ambiente que le rodea.

1.8.1 Q-*Learning*.

Q-*Learning* es una técnica propuesta por (Watkins, 1989) que tiene su origen en el método *Value Iteration* y está basada en la utilización de un valor q que representa el costo esperado de seleccionar una acción en un estado determinado y a partir de ahí seguir una política óptima.

Este es uno de los métodos más populares de AR. Existen algunas razones para esto. Primero, fue el primer método de AR en ser estudiado en profundidad para propósitos de control, con una fuerte demostración de convergencia establecida en (Jaakkola et al., 1994). Segundo, es la extensión por excelencia del concepto de aprendizaje autónomo al Control Óptimo, en el sentido de que es la técnica más simple que calcula directamente la política de acciones óptima sin un paso intermedio de evaluación de costo y sin utilizar un modelo. La tercera razón para la popularidad del Q-*Learning* son algunas evidencias tempranas de que se desempeña mejor que otros métodos de AR (Ribeiro, 1999).

1.9 Conclusiones parciales.

Los problemas de secuenciación son diversos y ocupan múltiples esferas de la vida real. Su forma de solución exhaustiva es muy costosa, por lo que encontrar una vía óptima de solución es un problema latente para los que investigan en esta temática.

La aplicación del AR ha reportado buenos resultados en la solución de problemas de *scheduling*. El *Flow Shop*, como problema de secuenciación no queda exento de los comentarios anteriores, por lo que el planteamiento de una posible forma de solucionar este problema utilizando el algoritmo Q-*Learning*, perteneciente al AR, constituye el punto de partida en el capítulo que sigue.

CAPÍTULO 2: Q-*Learning* aplicado a los problemas *Permutation Flow Shop*.

En este capítulo se muestran las instancias que se usarán para evaluar el algoritmo. Se propone una aplicación del método Q-*Learning* para dar solución al problema *Permutation Flow Shop* basada en el algoritmo NEH y que se apoya en la capacidad de aprendizaje de los agentes asociados al método. Finalmente se expone la implementación una aplicación con el objetivo de dar solución a los problemas de *scheduling* mediante AR.

2.1 Instancias del problema *Flow Shop Scheduling*.

Para medir la efectividad del algoritmo propuesto se usará el conjunto de instancias de *Taillard* propuestas en (Beasley, 1990) y que tienen el siguiente formato:

En la primera línea contiene la cantidad de trabajos y la cantidad de máquinas y luego una línea para cada trabajo que lista su tiempo de procesamiento en cada una de las máquinas. Máquinas y trabajos se enumeran desde cero. En la Figura se muestra una instancia de ejemplo de cuatro trabajos y 2 máquinas.

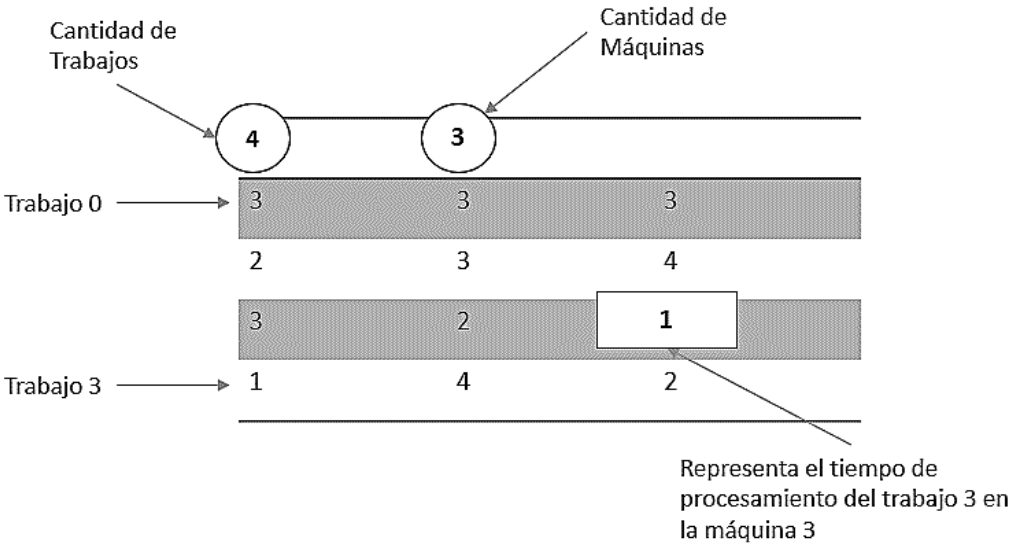


Figura : Formato de las instancias de *Taillard*.

La primera línea contiene dos números enteros (m, n) que son el número de trabajos y máquinas respectivamente existentes en el problema.

Cada una de las siguientes n líneas corresponde a un trabajo; la segunda línea corresponde al trabajo 0, la tercera al trabajo 1 y así sucesivamente; cada trabajo contiene una lista de m

operaciones a ser procesadas y que son descritas en la instancia por la cantidad de unidades de tiempos que demoran.

2.2 Algoritmo NEH (*Nawaz*, *Enscore*, *Ham*).

El algoritmo NEH (Nawaz et al., 1983) es probablemente la heurística constructiva más conocida para los problemas de tipo *Permutation Flow Shop*. En NEH, la idea básica es primero secuenciar de forma óptima los dos trabajos de mayor tiempo de procesamiento usando la regla de *Johnson*. Luego introducir los trabajos restantes, en orden decreciente de sus tiempos totales de procesamiento, uno a uno, en uno de los espacios dentro de los trabajos ya secuenciados, de forma tal que el *makespan* total se minimice en cada caso. La Tabla Tabla : Algoritmo NEH. muestra los pasos del algoritmo NEH.

Paso 1.	Ordenar los n trabajos de forma decreciente según sus tiempos de procesamiento.
Paso 2.	Tomar los primeros dos trabajos y secuenciarlos con el objetivo de minimizar el *makespan* parcial como si existieran solo esos dos trabajos.
Paso 3.	Para $k = 3$ hasta n hacer **Paso 4**
Paso 4.	Insertar el k^{th} trabajo en la posición que minimice el *makespan* parcial entre las k posibles.

Tabla : Algoritmo NEH.

2.2.1 Ejemplo de funcionamiento.

La siguiente tabla muestra una instancia de problema que es usada a modo de ejemplo:

	M0	**M1**	**M2**	**M3**	**M4**
J0	5	9	8	10	1
J1	9	3	10	1	8
J2	9	4	5	8	6
J3	4	8	8	7	2

Tabla : Instancia de ejemplo para el algoritmo NEH.

- **Paso 1**: Ordenar los cuatro trabajos de forma decreciente según sus tiempos de procesamiento.

  T0 = 5 + 9 + 8 + 10 + 1 = 33

  T1 = 9 + 3 + 10 + 1 + 8 = 31

  T2 = 9 + 4 + 5 + 8 + 6 = 32

  T3 = 4 + 8 + 8 + 7 + 2 = 29

  Trabajos ordenados: **J0, J2, J1, J3**

- **Paso 2:** Tomar los primeros dos trabajos y secuenciarlos con el objetivo de minimizar el *makespan* parcial como si existieran solo esos dos trabajos.

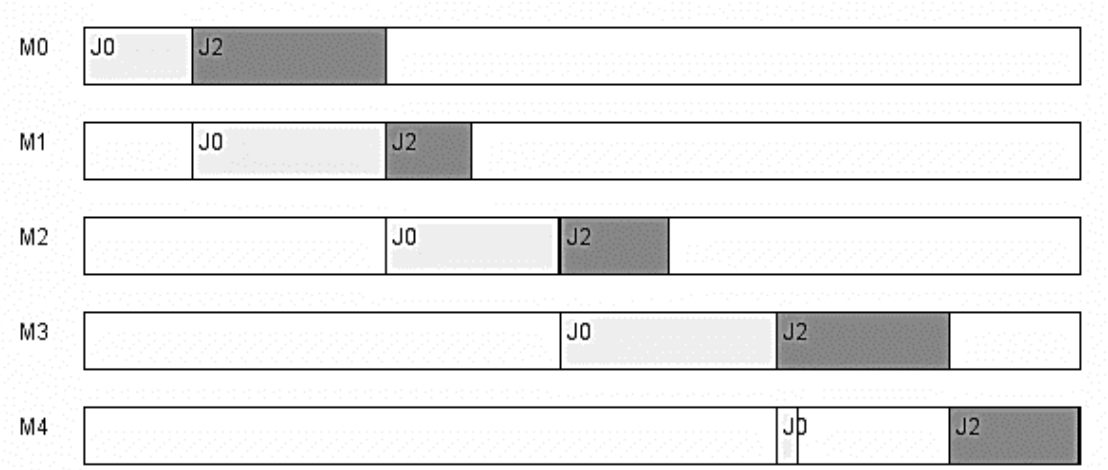


Figura : Secuencia J0-J2, *makespan* 46.

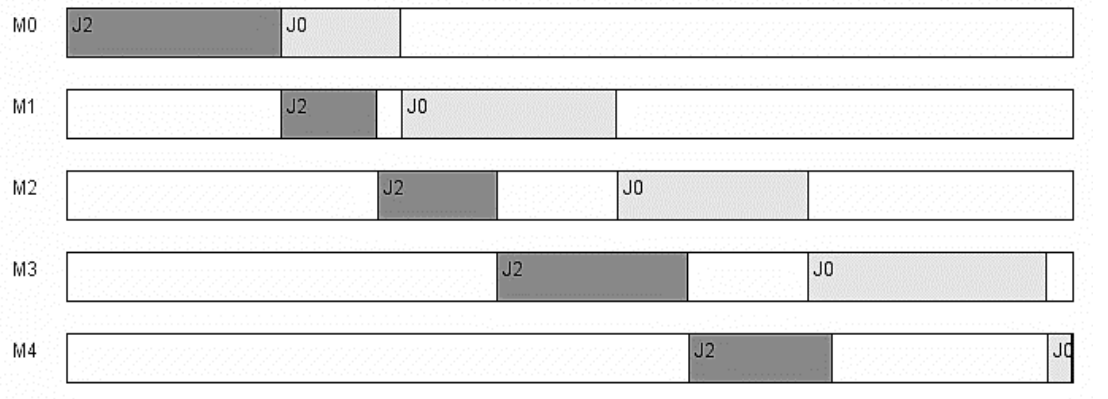


Figura : Secuencia J2-J0, *makespan* 42.

La mejor secuencia es **J2-J0** con *makespan* 42.

- **Paso 3:** Hacer $k = 3$.
- **Paso 4:** Seleccionar J1 (trabajo en la 3^{ra} posición de la lista del **Paso 1**) y encontrar la mejor secuencia insertando J1 en las 3 posibles posiciones en la secuencia parcial J2-J0 obtenida en el **Paso 2**.

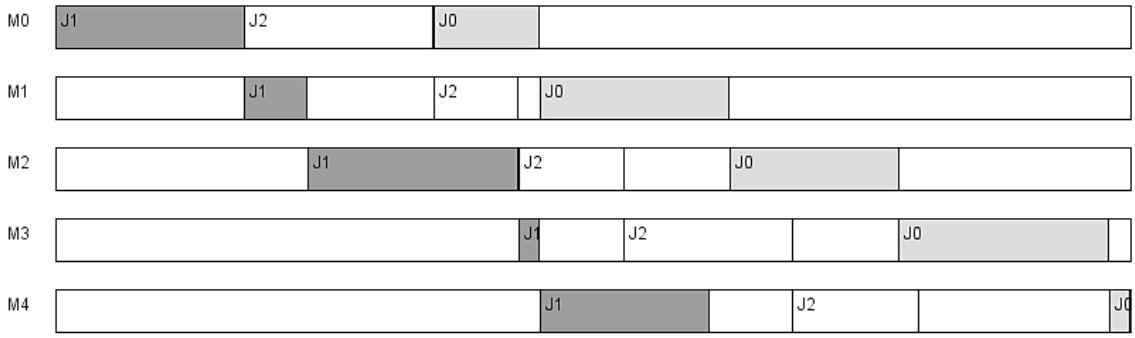


Figura : Secuencia J1-J2-J0, *makespan* 51.

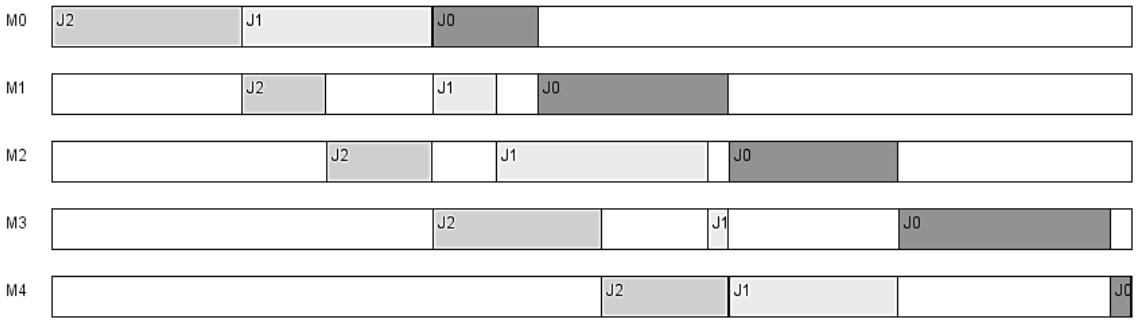


Figura : Secuencia J2-J1-J0, *makespan* 51.

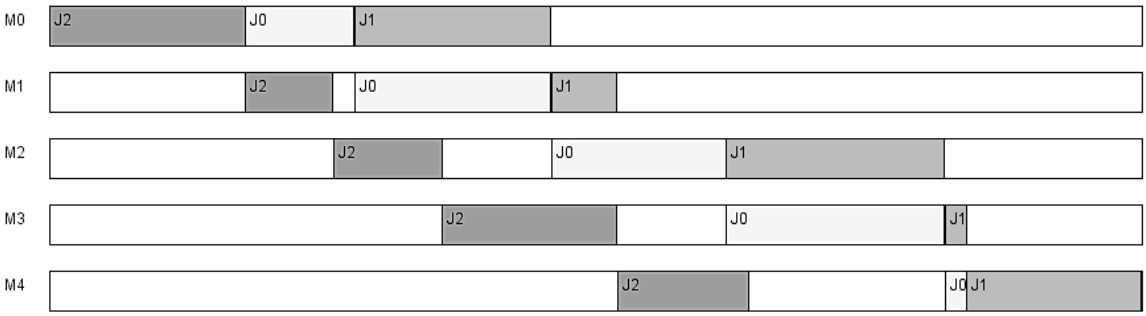


Figura : Secuencia J2-J0-J1, *makespan* 50.

La mejor secuencia es **J2-J0-J1** con *makespan* 50.

- **Paso 3a:** Hacer $k = 4$.

- **Paso 4a:** Seleccionar J3 (trabajo en la 4^{ta} posición de la lista del **Paso 1**) y encontrar la mejor secuencia insertando J3 en las 4 posibles posiciones en la secuencia parcial J2-J0-J1 obtenida en el **Paso 3**.

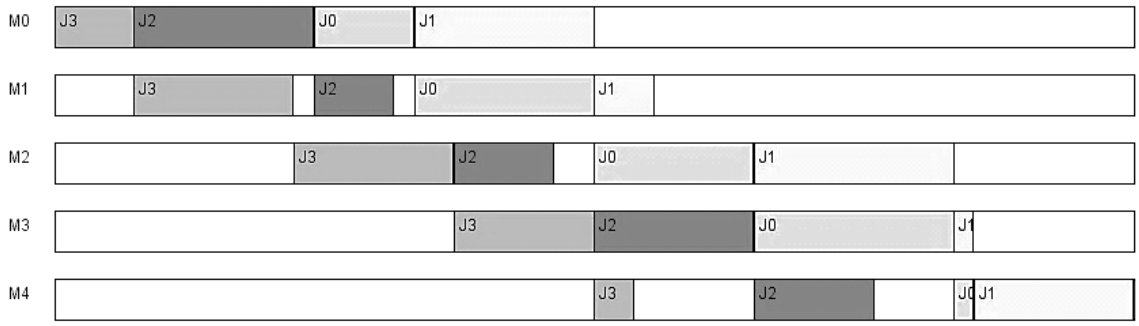


Figura : Secuencia J3-J2-J0-J1, *makespan* 54.

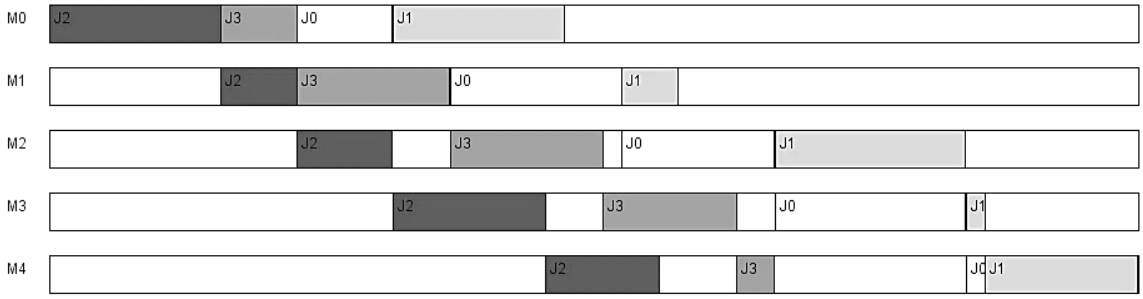


Figura : Secuencia J2-J3-J0-J1, *makespan* 57.

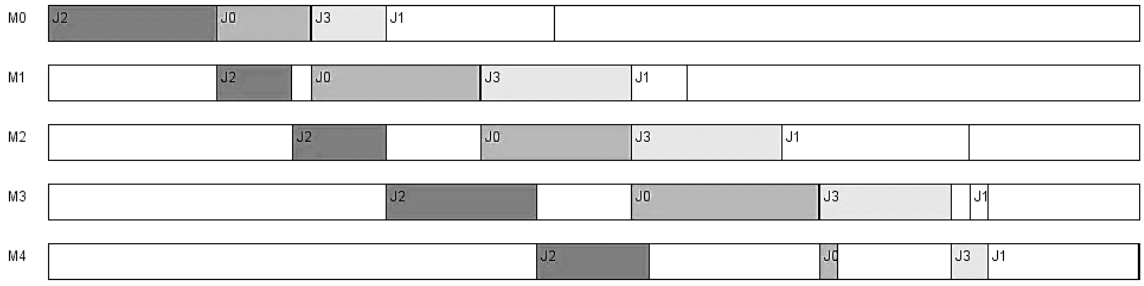


Figura : Secuencia J2-J0-J3-J1, *makespan* 58.

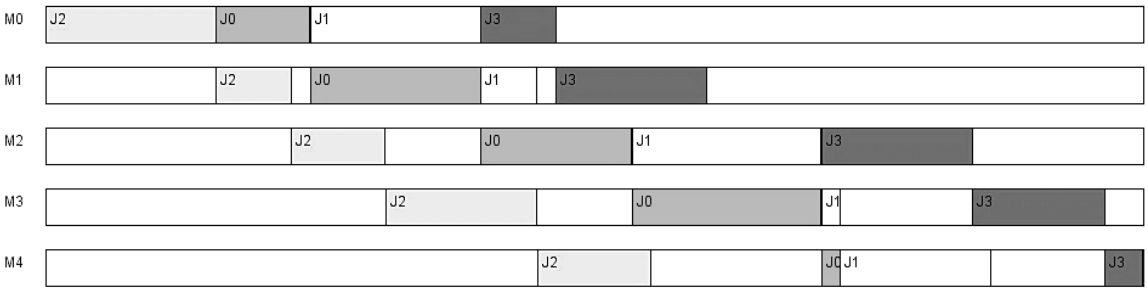


Figura : Secuencia J2-J0-J1-J3, *makespan* 58.

La mejor secuencia es **J3-J2-J0-J1** con *makespan* 54.

2.3 Q-*Learning*.

Este algoritmo se basa en aprender una función acción-valor que devuelva la ganancia esperada de tomar una acción determinada en cierto estado. 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 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:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

En esta expresión, $\alpha \in [0,1]$ representa la velocidad de 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. Por ejemplo, si $\alpha = 0$, entonces no existe actualización, y si $a = 1$, entonces el valor anterior es reemplazado por el nuevo estimado. 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 $(0 \leq \gamma \leq 1)$, 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 algoritmo puede resumirse como sigue:

Inicializar $Q(s, a)$ arbitrariamente.

Para cada episodio:

Inicializar s.

Repetir (para cada paso del episodio):

Escoger a desde s usando una política ε-*greedy*.

Tomar acción a, observar r, s'

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

$$s \leftarrow s'$$

Hasta que s sea terminal.

Tabla : Algoritmo Q-*Learning*.

El algoritmo anterior es usado por los agentes para aprender de la experiencia o el entrenamiento, donde cada episodio es equivalente a una sesión de entrenamiento. En cada iteración el agente explora el ambiente y obtiene señales numéricas hasta que alcanza el estado objetivo. El propósito del entrenamiento es incrementar el conocimiento del agente, representado en este caso 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 (Martínez, 2012).

El agente necesitar balancear exploración y explotación. La estrategia de selección ε-*greedy* instruye al agente a seguir la política actual π la mayoría del tiempo, pero a algunas veces, seleccionar una acción aleatoria (con igual probabilidad para cada posible acción a en el estado actual s). La probabilidad ε determina cuando seleccionar una acción aleatoria; esto permite un equilibrio entre exploración y explotación.

2.4 Aplicación del Q-*Learning* a los problemas *Permutation Flow Shop*.

El enfoque de solución al *Permutation Flow Shop* brindado en este trabajo está basado en el algoritmo NEH. Cuando se aplica AR para resolver el *Permutation Flow Shop* se utiliza un agente como secuenciador para el problema. Este agente decide globalmente las acciones a ejecutar.

Para el uso del algoritmo Q-*Learning* existen elementos importantes a tomar en consideración, es importante definir los estados, las acciones y la estructura de datos para almacenar los Q-valores, así como las recompensas que recibirán los agentes.

Estados: Para el agente la definición de estado va a coincidir con los trabajos que han finalizado y el orden en que lo hicieron, es decir la secuencia que se ha ido construyendo. Esto quiere decir que al comienzo de un episodio, el estado del ambiente es vacío, ya que no se ha añadido ningún trabajo a la secuencia; por consiguiente al finalizar el episodio el estado del ambiente estará dado por todos los trabajos secuenciados.

Acciones: De forma similar al algoritmo NEH la toma de una acción por parte del agente equivale a decidir en qué posición de la secuencia ya construida va a ser introducido el trabajo

con mayor tiempo de procesamiento total del conjunto de trabajos que se encuentran sin procesar. Esto quiere decir que al comienzo de un episodio sólo existe una acción posible, tomar el trabajo con mayor tiempo de procesamiento y ubicarlo de primero en la secuencia; por consiguiente al finalizar el episodio no existe ninguna acción disponible, ya que todos los trabajos han sido procesados.

Recompensas: Como se ha descrito, el estado está representado por la secuencia que se ha ido construyendo, una acción constituye en insertar un nuevo trabajo en dicha secuencia y teniendo en cuenta que nuestro objetivo final es minimizar el *makespan*; definimos como recompensa $1/makespan$, nótese que a medida que el *makespan* de la secuencia sea mayor, menor será la recompensa para la acción seleccionada. Cabe señalar entonces que el mayor Q-valor determina la mejor acción en un estado cualquiera.

Q-valores: Uno de los supuestos del algoritmo Q-*Learning* es que es posible enumerar todos los pares (s, a) que puedan aparecer en el transcurso de la resolución del problema. Para los problemas de secuenciación de tareas, excepto para instancias muy pequeñas, esto implica requerimientos de memoria inaccesibles en la práctica. Por lo que nuestro algoritmo solo almacena aquellos estados que son necesarios, es decir, se almacenan solo los estados en que se encontraba el agente cuando se seleccionaron las acciones.

El algoritmo propuesto, denominado QL-NEH, puede resumirse como sigue:

Inicializar:

$Q(s, a) = \{\}$

$Mejor = \{\}$

Para cada episodio:

$s = \{\}$

Mientras existan trabajos sin procesar:

Seleccionar entre los trabajos pendientes, el de mayor tiempo de procesamiento total J_m.

Inicializar $acciones$ como el conjunto de posibles puntos de inserción de J_m dentro de s.

Si número aleatorio $\leq \varepsilon$:

Seleccionar una acción a al azar dentro de $acciones$.

En otro caso:

Seleccionar la acción a devuelta por $Mejor\_acción(s)$.

Tomar acción a, observar s' y r como $1/\text{makespan}\, s'$.

Hacer $a' \leftarrow Mejor\_acción(s')$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

$s \leftarrow s'$

Si makespan $s <$ makespan $Mejor$:

$Mejor \leftarrow s$

Tabla : Algoritmo propuesto (QL-NEH).

$Mejor\_acción(s)$:

Inicializar $acciones$ como el conjunto de posibles puntos de inserción de J_m dentro de s.

Si Q contiene s:

Seleccionar a como $\max_{a'} Q(s,a')$ tal que $a \in acciones$.

En otro caso:

Para cada a en $acciones$:

Tomar acción a, observar s' y r como $1/\text{makespan}\, s'$.

Seleccionar la a que mejor r obtenga.

$Q(s,a) \leftarrow r$.

Tabla : Procedimiento para escoger la mejor acción.

2.4.1 Ejemplo de funcionamiento.

A continuación se ilustra el proceso de resolución del problema a través de un ejemplo pequeño de solo tres trabajos y dos máquinas. Los parámetros del algoritmo serán los siguientes: dos episodios, ritmo de aprendizaje $(\alpha) = 0.2$, factor de descuento $(\gamma) = 0.7$ y $\varepsilon = 0.1$.

	M0	**M1**	**Tiempo de procesamiento**
J0	2	3	**5**
J1	1	2	**3**
J2	3	5	**8**

Primero se inicializan los Q-valores:

Estado	Acción	Q-valor

Episodio 1:

Iteración 1:

- El agente observa el estado del ambiente $s = \{\}$.
- Se selecciona el trabajo pendiente con mayor tiempo de procesamiento total $J_m = J2$.
- Evaluar el conjunto de acciones posibles, en este caso hay solo una opción $acciones = \{J2: 0\}$.
- Supongamos que se genera un valor aleatorio menor que ε y por tanto se escoge una acción al azar, en este caso solo existe la opción $a = J2: 0$.
- Realizar la acción seleccionada:

  $s' = \{J2\}$

  $r = 1/\text{makespan}\, s' = 1/8 = 0.125$

- Mejor acción para el estado futuro $a' \leftarrow J0: 0$ con un Q-valor de 0.1.

- Se actualiza el Q-valor correspondiente:

$$Q(\{\}, J2{:}\,0) = 0 + 0.2[0.125 + 0.7 * 0.1 - 0] = 0.039$$

Estado	Acción	Q-valor
$\{J2\}$	$J0{:}\,0$	0.1
{}	$J2{:}\,0$	0.039

Iteración 2:

- $s = \{J2\},\ J_m = J0,\ acciones = \{J0{:}\,0, J0{:}\,1\}$.
- Supongamos que se genera un valor aleatorio menor que ε y por tanto se escoge una acción al azar, en este caso la opción $a = J0{:}\,0$.
- Realizar la acción seleccionada:

$$s' = \{J0 - J2\}$$

$$r = 1/\text{makespan}\, s' = 1/10 = 0.1$$

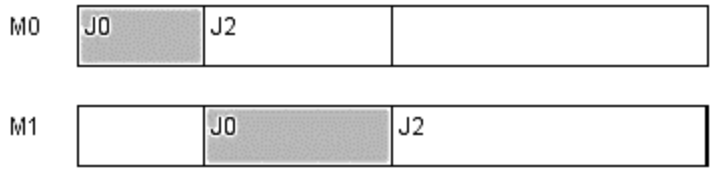


Figura : Secuencia J0-J2, *makespan* 10.

- $a' \leftarrow J1{:}\,0$ con un Q-valor de 0.09
- Se actualiza el Q-valor correspondiente:

$$Q(\{J2\}, J0{:}\,0) = 0.1 + 0.2[0.1 + 0.7 * 0.09 - 0.1] = 0.1126$$

Estado	Acción	Q-valor
$\{J2\}$	$J0{:}\,0$	0.1126
{}	$J2{:}\,0$	0.039
$\{J0 - J2\}$	$J1{:}\,0$	0.09

Iteración 3:

- $s = \{J0 - J2\},\ J_m = J1,\ acciones = \{J1{:}\,0, J1{:}\,1, J1{:}\,2\}$.
- Supongamos que se genera un valor aleatorio menor que ε y por tanto se escoge una acción al azar, en este caso la opción $a = J1{:}\,2$.
- Realizar la acción seleccionada:

$$s' = \{J0 - J2 - J1\}$$

$$r = 1/\text{makespan}\, s' = 1/12 = 0.083$$

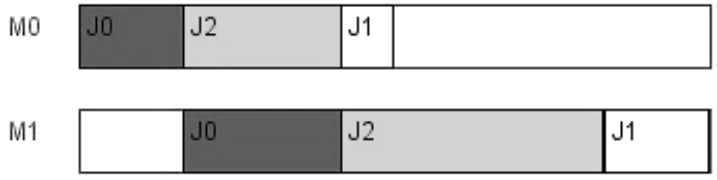


Figura : Secuencia J0-J2-J1, *makespan* 12.

- Todos los trabajos han sido procesados en el estado futuro por lo que no existe acción posible.
- Se actualiza el Q-valor correspondiente:

$$Q(\{J0 - J2\}, J1{:}\,2) = 0 + 0.2[0.083 + 0.7 * 0 - 0] = 0.0166$$

Estado	Acción	Q-valor
$\{J2\}$	$J0{:}\,0$	0.1126
{}	$J2{:}\,0$	0.039
$\{J0 - J2\}$	$J1{:}\,0$	0.09
$J0 - J2$	$J1{:}\,2$	0.0166

Episodio 2:

Iteración 1:

- $s = \{\},\ J_m = J2,\ acciones = \{J2{:}\,0\}$.
- Supongamos que se genera un valor aleatorio mayor que ε y por tanto se escoge la mejor acción conocida (acción con mayor Q-valor en el estado s), en este caso la opción $a = J2{:}\,0$.

- Realizar la acción seleccionada:

$$s' = \{J2\}$$

$$r = 1/\text{makespan}\, s' = 1/8 = 0.125$$

- $a' \leftarrow J0{:}\,0$ con un Q-valor de 0.1126

- Se actualiza el Q-valor correspondiente:

$$Q(\{\}, J2{:}\,0) = 0.039 + 0.2[0.125 + 0.7 * 0.1126 - 0.039] = 0.071964$$

Estado	Acción	Q-valor
$\{J2\}$	$J0{:}\,0$	0.1126
$\{\}$	$J2{:}\,0$	0.071964
$\{J0 - J2\}$	$J1{:}\,0$	0.09
$\{J0 - J2\}$	$J1{:}\,2$	0.0166

Iteración 2:

- $s = \{J2\},\ J_m = J0,\ acciones = \{J0{:}\,0, J0{:}\,1\}$.

- Supongamos que se genera un valor aleatorio mayor que ε y por tanto se escoge la mejor acción conocida (acción con mayor Q-valor en el estado s), en este caso la opción $a = J0{:}\,0$.

- Realizar la acción seleccionada:

$$s' = \{J0 - J2\}$$

$$r = 1/\text{makespan}\, s' = 1/10 = 0.1$$

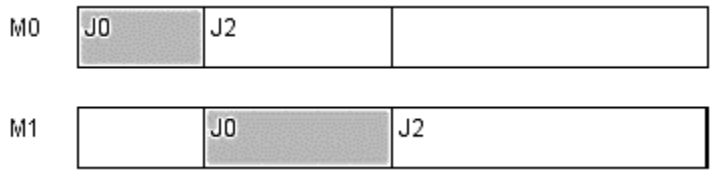


Figura : Secuencia J0-J2, *makespan* 10.

- $a' \leftarrow J1{:}\,0$ con un Q-valor de 0.09

- Se actualiza el Q-valor correspondiente:

$$Q(\{J2\}, J0:0) = 0.1126 + 0.2[0.1 + 0.7 * 0.09 - 0.1126] = 0.12268$$

Estado	Acción	Q-valor
$\{J2\}$	$J0:0$	0.12268
{}	$J2:0$	0.071964
$\{J0 - J2\}$	$J1:0$	0.09
$\{J0 - J2\}$	$J1:2$	0.0166

Iteración 3:

- $s = \{J0 - J2\},\ J_m = J1,\ acciones = \{J1:0, J1:1, J1:2\}$.
- Supongamos que se genera un valor aleatorio menor que ε y por tanto se escoge una acción al azar, en este caso la opción $a = J1:0$.
- Realizar la acción seleccionada:

$$s' = \{J1 - J0 - J2\}$$

$$r = 1/\text{makespan}\, s' = 1/11 = 0.09$$

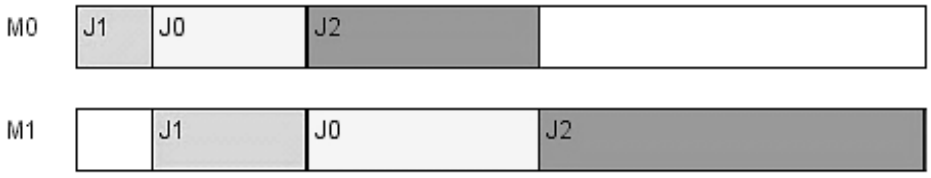


Figura : Secuencia J1-J0-J2, *makespan* 11.

- Todos los trabajos han sido procesados en el estado futuro por lo que no existe acción posible.
- Se actualiza el Q-valor correspondiente:

$$Q(\{J0 - J2\}, J1:0) = 0.09 + 0.2[0.09 + 0.7 * 0 - 0.09] = 0.09$$

Estado	Acción	Q-valor
$\{J2\}$	$J0:0$	0.12268

{}	$J2:0$	0.071964
$\{J0-J2\}$	$J1:0$	0.09
$\{J0-J2\}$	$J1:2$	0.0166

Finalmente, se tiene que tras dos episodios completos, la mejor secuenciación obtenida es J1-J0-J2 con un *makespan* de 11 unidades de tiempo.

2.5 Propuesta de aplicación.

Con el objetivo de facilitar el trabajo con los problemas de *scheduling* se creó una aplicación que le brinda al usuario una forma más intuitiva y cómoda de lidiar con dichos problemas y que le permita además resolverlos usando técnicas de AR.

Esta aplicación se desarrolló con el objetivo de proponer una interfaz gráfica para la investigación principal de este proyecto que es la resolución del problema *Permutation Flow Shop* usando AR.

La aplicación ofrece la posibilidad al usuario de cargar las instancias de los problemas de secuenciación que desea resolver mediante AR, además puede configurar de forma sencilla todos los parámetros necesarios del algoritmo. Una de las utilidades que se brindan es precisamente la visualización de las secuencias obtenidas como resultado de aplicar el método a las instancias seleccionadas, además de guardarlas en archivos de texto. Se implementaron además del *makespan*, otros objetivos que pueden ser de interés para el usuario y se hizo de forma tal que este pueda elegir cuál de los objetivos desea optimizar. Todas estas opciones se reflejan en el siguiente diagrama de casos de usos:

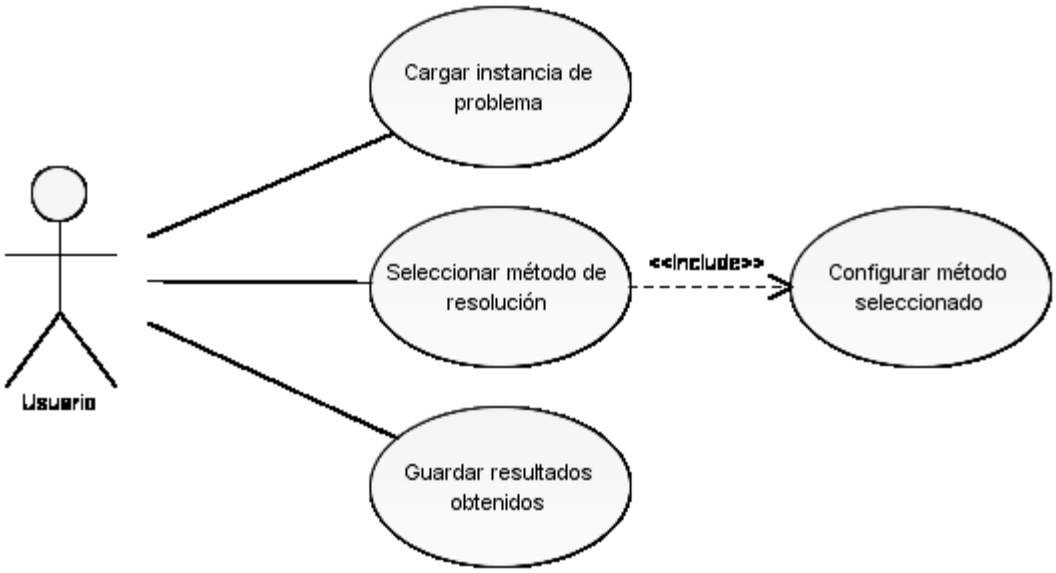


Figura : Diagrama de casos de uso de la aplicación propuesta.

La Figura 20 muestra un diagrama de clases que representa las principales entidades que componen al sistema y las relaciones entre ellas.

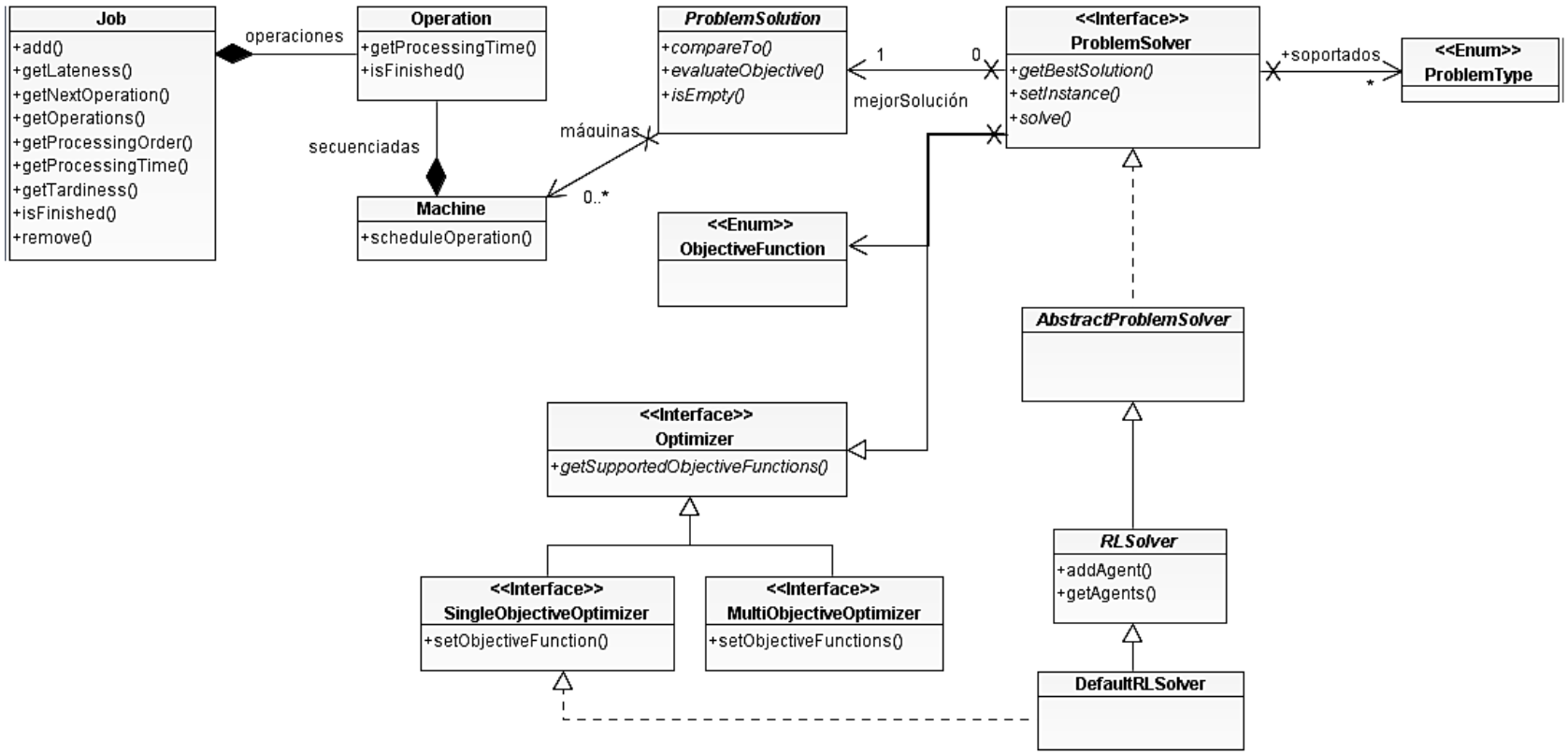


Figura : Diagrama de las principales clases del sistema.

Como se puede apreciar, se tienen las clases que representan el modelo del sistema (*Job*, *Operation* y *Machine*), un trabajo (*Job*) está compuesto por operaciones (*Operation*), las cuales se agregan a las colas de ejecución de las máquinas (*Machine*). Una solución (*ProblemSolution*) a un problema de *scheduling* está conformada por las máquinas que lo componen con el historial de las operaciones que estas procesaron.

Por otro lado se tiene la jerarquía de clases encargadas de resolver el problema de secuenciación. Esta está conformada por la interfaz de los resolvedores de problemas (*ProblemSolver*) que es implementada por la case abstracta que representa a los métodos de Aprendizaje Reforzado (*RLSolver*). En la Figura 21 se muestra un diagrama de clases para profundizar en el diseño del módulo de AR.

El componente principal de este módulo es la interfaz *Agent* que representa un agente, este colabora con otras clases como *EnvironmentState* (estado del ambiente) y *AgentAction* (acción tomada por el agente). Para llevar a cabo el aprendizaje, el agente guarda toda su experiencia, basada en estados y acciones en una memoria (*Memory*). La interfaz *SelectionStrategy* representa las diversas estrategias de selección que puede usar un agente para decidir entre varias acciones a ejecutar.

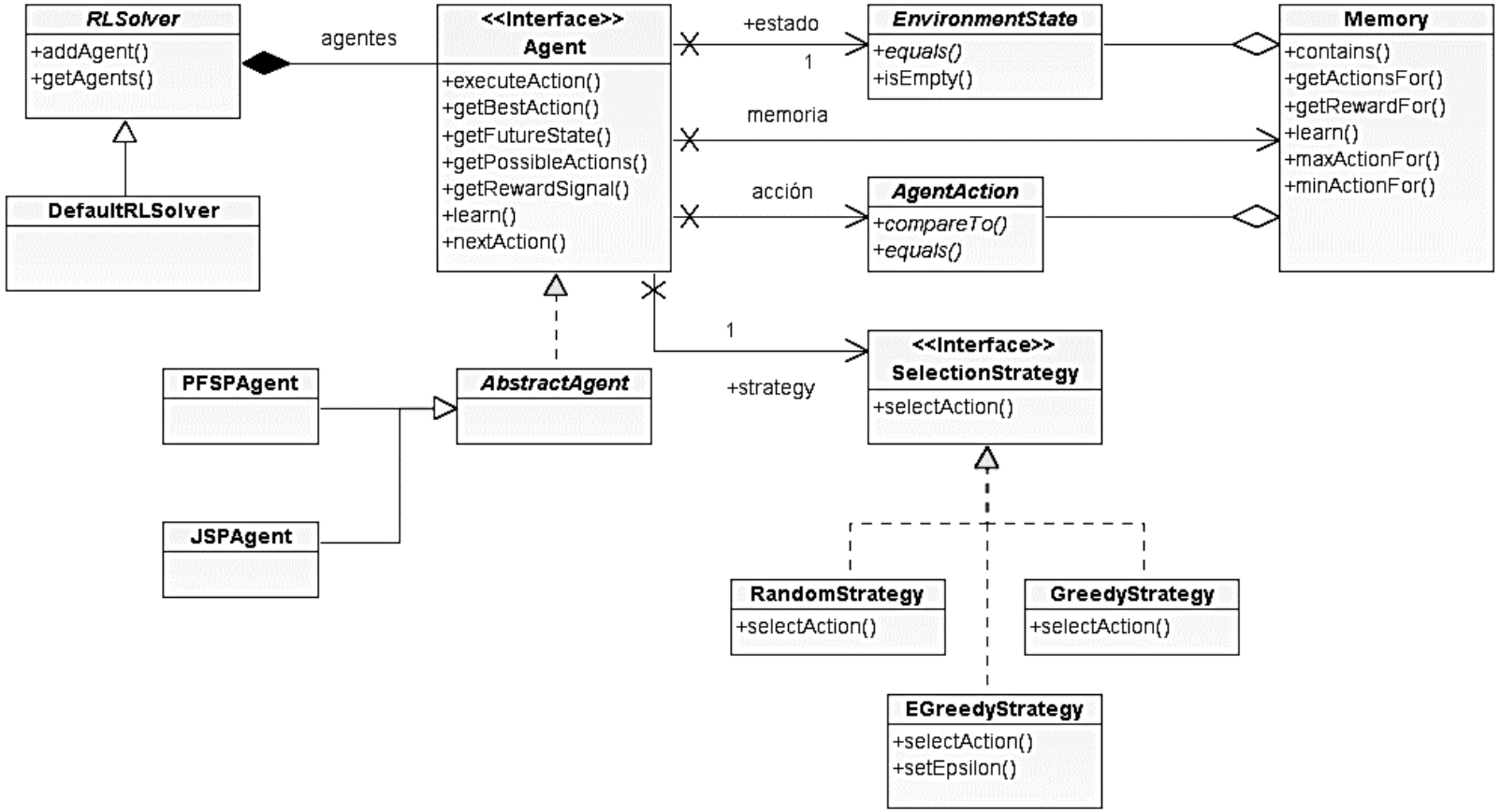


Figura : Diagrama de clases del módulo de AR.

2.5.1 Patrones de diseño utilizados.

Con el objetivo de que la aplicación se pueda adaptar sin muchas dificultades a otros tipos de problemas de *scheduling* y que sea posible integrar en el futuro distintos métodos de resolución se consideró el uso de algunos patrones de diseño para lograr así un alto grado de cohesión entre los componentes del sistema.

Uno de los patrones de diseño usado es el *Singleton* o instancia única, cuyo objetivo es garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. Otro ejemplo se manifiesta en la jerarquía de estrategias de selección (*SelectionStrategy*) con el uso del patrón Estrategia que en efecto permite disponer de varios métodos para tratar un problema y elegir cuál utilizar en tiempo de ejecución.

En la implementación de la interfaz gráfica de usuario estuvieron involucrados varios patrones de diseño que se muestran en el siguiente diagrama de clases.

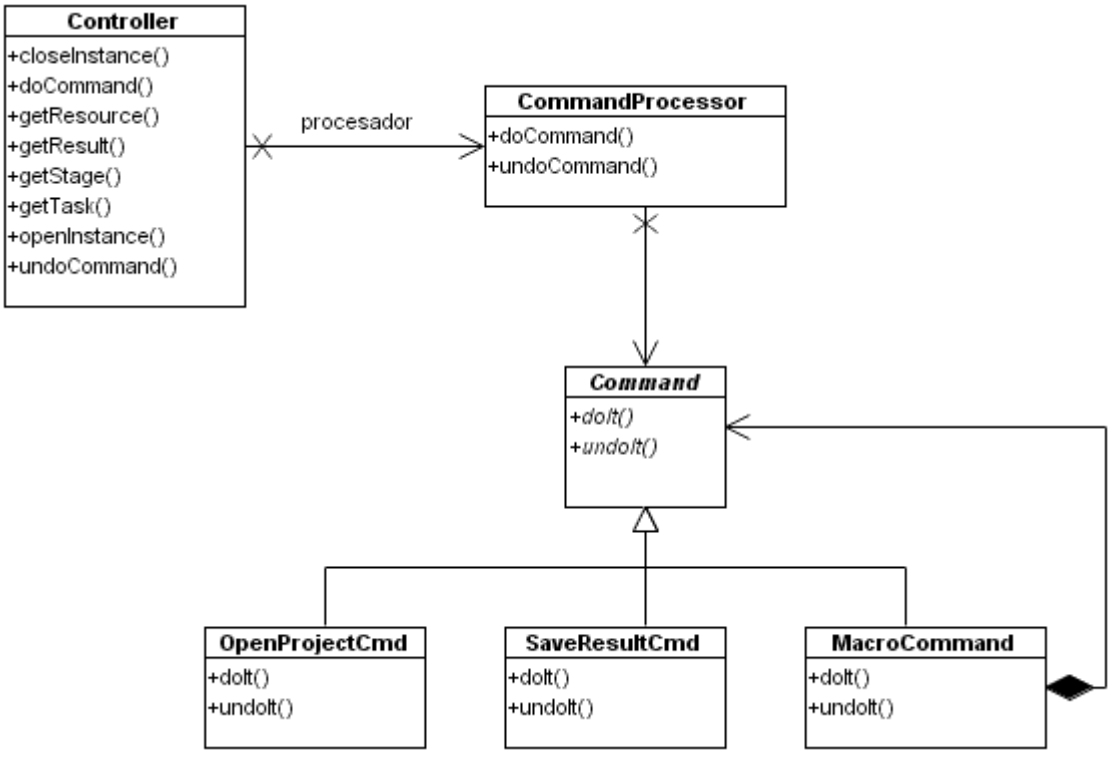


Figura : Diagrama de clase de las principales clases del módulo de interfaz gráfica.

De forma general se usó el patrón Modelo-Vista-Controlador (MVC) para separar los datos de la aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos. Se empleó el patrón *Command* que permite solicitar una operación a un objeto sin conocer

realmente el contenido de esta operación, ni el receptor real de la misma y para extender la funcionalidad de este se combinó con el patrón *Composite* que sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol; esto simplifica el tratamiento de los objetos creados, ya que al poseer todos ellos una interfaz común, se tratan todos de la misma manera, su uso en el sistema se pone de manifiesto en la clase *MacroCommand* que nos permite ejecutar un comando complejo a partir de la ejecución de otros más simples.

A continuación se muestran algunas imágenes que exponen las características esenciales del sistema, así como la integración con los problemas de tipo *Job Shop*.

Figura : Vista principal de la aplicación con una instancia del problema *Flow Shop*.

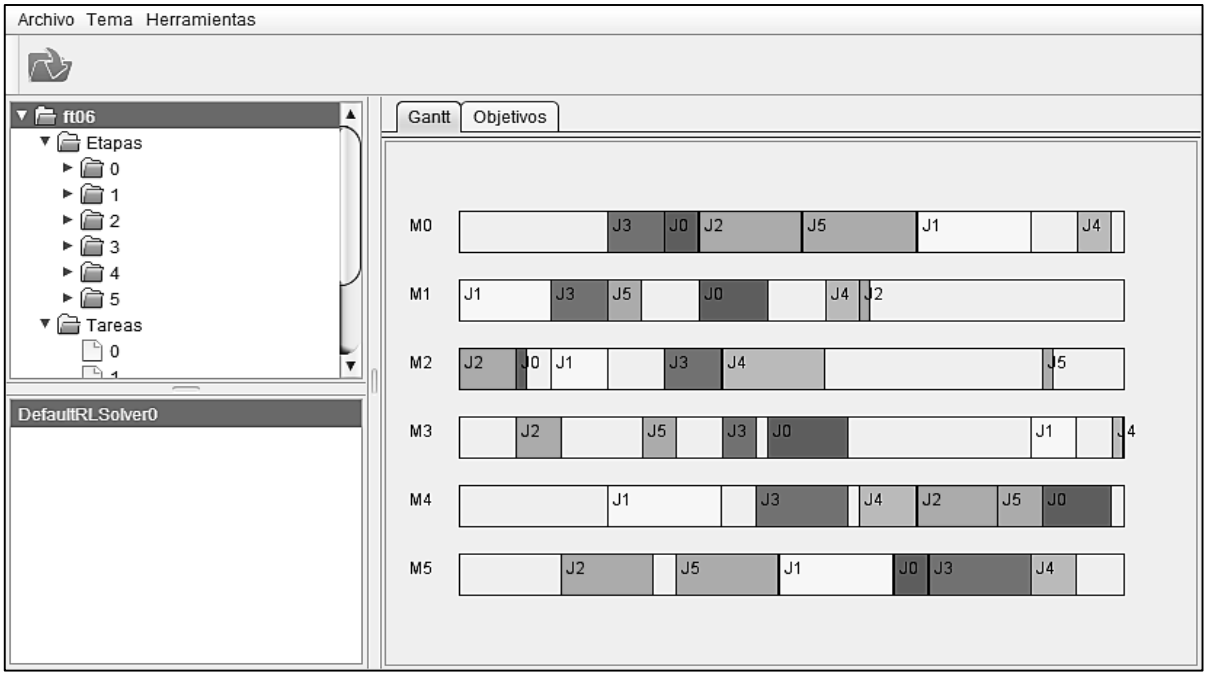


Figura : Vista principal de la aplicación con una instancia del problema *Job Shop*.

Figura : Vista de la configuración del Aprendizaje Reforzado.

2.6 Conclusiones parciales.

En este capítulo se presentaron las modificaciones necesarias que deben hacerse al algoritmo Q-*Learning* para aplicarlo al problema de secuenciación de tipo *Flow Shop*.

Se propone un enfoque de solución a los problemas de tipo *Permutation Flow Shop* con la utilización del Q-*Learning* y basado en el algoritmo NEH como mecanismo de selección de acciones y otorgamiento de recompensas.

Se implementó una aplicación de fácil utilización e interfaz amigable que le facilita el trabajo al usuario a la hora de resolver problemas de secuenciación de tareas.

CAPÍTULO 3: Prueba experimental y análisis de los resultados.

En este capítulo se muestran los experimentos realizados con la variante del algoritmo Q-*Learning* propuesta (QL-NEH) para la función objetivo *makespan*. Las instancias utilizadas en los experimentos proceden de la OR-Library (Beasley, 1990). Se realiza una validación estadística de la propuesta desarrollada en comparación con otros algoritmos reportados en la literatura que abordan los problemas de secuenciación tipo *Permutation Flow Shop*.

3.1 Marco experimental estadístico.

No existe un procedimiento establecido y aceptado para comparar algoritmos sobre múltiples conjuntos de datos. Un motivo importante es el comportamiento no determinístico de éstos, por lo que la diferencia detectada entre los resultados de dos algoritmos podría deberse a factores aleatorios, y no a una mejora real (Garcia and Herrera, 2008, Demšar, 2006).

Para determinar si las diferencias encontradas entre dos algoritmos son significativas, los investigadores pueden aplicar distintas técnicas estadísticas. Las pruebas son clasificadas generalmente en dos grupos:

- **Pruebas paramétricas:** estas pruebas (García et al., 2009, Luengo et al., 2009) son las más empleadas. Se caracterizan en que utilizan por cada algoritmo y función, el error medio alcanzado por un conjunto de ejecuciones y lo utilizan para identificar si la diferencia entre dos algoritmos es estadísticamente significativa. Un ejemplo de estas pruebas son el t-*Student* (para comparar dos algoritmos) y ANOVA (para comparar múltiples algoritmos).
- **Pruebas no paramétricas:** estas pruebas (Shesking, 2006) utilizan una representación ordinal de los valores basada en el orden de los algoritmos para cada uno de los problemas. Un ejemplo de estas pruebas son *Wilcoxon* (para comparar los resultados de dos algoritmos) y *Bonferroni-Dunn* o *Holm* y la prueba de *Friedman* (para comparaciones de varios).

Para aplicar las pruebas paramétricas es necesario que los datos tengan una distribución normal y una varianza homogénea. Sin embargo, los resultados obtenidos por las metaheurísticas no cumplen con estas características, de ahí que surja la necesidad de aplicar pruebas no paramétricas al algoritmo propuesto.

3.2 Pruebas estadísticas utilizadas.

A continuación se hace una breve descripción de las pruebas estadísticas a usar para el análisis del comportamiento del algoritmo propuesto en este trabajo.

- **Error Relativo (ER):** es la desviación relativa entre una solución, tomada como referencia, y las propuestas de soluciones analizadas.

$$ER = (MK - CS)/CS * 100$$

  Donde MK es el mejor *makespan* obtenido por el enfoque propuesto y CS es el mejor resultado reportado.

- **Prueba de Ranking de Signos de *Wilcoxon*:** se utiliza para comparar la media de dos muestras relacionadas y determinar si existen diferencias entre ellas. Se utiliza como alternativa a la prueba de t-*Student* cuando no se puede suponer la normalidad de dichas muestras. Se propone una hipótesis H0 (H0: no existen diferencias significativas) la cual se rechaza si la significación de la prueba es menor que 0.05 ($sig < 0.05$), en caso contrario la hipótesis propuesta es aceptada.

3.3 Resultados experimentales.

Para medir la efectividad del algoritmo propuesto se usarán las instancias localizadas en la biblioteca OR (Beasley, 1990). Esta biblioteca está conformada por un conjunto de instancias que abarcan varios problemas en el área de la IO. Para este trabajo, se usan diferentes casos que sirven para comparar los resultados obtenidos con la solución ofrecida en esta investigación. Se han tomado 90 instancias del conjunto de datos de *Taillard*, 10 de cada tamaño 20x5, 50x5, 100x5, 20x10, 50x10, 100x10, 20x20, 50x20, 100x20. Para los propósitos de la comparación se usaron las mejores cotas superiores para dichos problemas. El algoritmo QL-NEH fue programado en Java y ejecutado en una computadora con un CPU *Core* i3 3.5 GHz y 2 GB de RAM.

Primero se hace una comparación entre los mejores resultados reportados para las instancias usadas y los alcanzados por la variante propuesta en este trabajo. Luego se realiza un test de *Wilcoxon* para comparar dichos resultados con los obtenidos por el algoritmo NEH (Nawaz et al., 1983) y con los de una heurística de mejora conocida como ALA propuesta por (Agarwal et al., 2006). Finalmente se compara el método propuesto con otros tres procedimientos heurísticos

en cuanto al Error Medio Relativo. Han sido seleccionados estos enfoques principalmente porque usan las mismas instancias y reportan resultados para todas ellas.

3.3.1 Estudio comparativo entre QL-NEH y las cotas superiores conocidas.

Las tablas presentadas más adelante resumen los resultados para las 90 instancias de *Taillard*. Cada prueba fue repetida con 10 ejecuciones para cada instancia y se seleccionó la mejor solución; por lo que fueron 900 ejecuciones en total.

Las instancias están agrupadas por número de trabajos y de máquinas y se muestra el Error Medio Relativo (EMR) al final de los resultados. El EMR toma en cuenta el promedio de los resultados para el conjunto completo de instancias.

Instancia	Cota superior (CS)	QL-NEH	Error relativo (ER)
ta001	1278	**1278**	0,000%
ta002	1359	**1359**	0,000%
ta003	1081	**1081**	0,000%
ta004	1293	**1293**	0,000%
ta005	1235	**1235**	0,000%
ta006	1195	**1195**	0,000%
ta007	1239	**1239**	0,000%
ta008	1206	**1206**	0,000%
ta009	1230	**1230**	0,000%
ta010	1108	**1108**	0,000%
EMR			**0,000%**

Tabla : Desempeño de las soluciones obtenidas para las instancias de 20x5.

Instancia	Cota superior (CS)	QL-NEH	Error relativo (ER)
ta011	1582	1583	0,063%
ta012	1659	1669	0,603%

ta013	1496	1501	0,334%
ta014	1378	1382	0,290%
ta015	1419	1424	0,352%
ta016	1397	1404	0,501%
ta017	1484	**1484**	0,000%
ta018	1538	1545	0,455%
ta019	1593	1601	0,502%
ta020	1591	1598	0,440%
EMR			**0,354%**

Tabla : Desempeño de las soluciones obtenidas para las instancias de 20x10.

Instancia	**Cota superior (CS)**	**QL-NEH**	**Error relativo (ER)**
ta021	2297	2304	0,305%
ta022	2099	**2099**	0,000%
ta023	2326	2343	0,731%
ta024	2223	2229	0,270%
ta025	2291	2305	0,611%
ta026	2226	2246	0,898%
ta027	2273	2295	0,968%
ta028	2200	2222	1,000%
ta029	2237	2241	0,179%
ta030	2178	2200	1,010%
EMR			**0,597%**

Tabla : Desempeño de las soluciones obtenidas para las instancias de 20x20.

Instancia	Cota superior (CS)	QL-NEH	Error relativo (ER)
ta031	2724	2724	0,000%
ta032	2834	2839	0,176%
ta033	2621	**2621**	0,000%
ta034	2751	2753	0,073%
ta035	2863	**2863**	0,000%
ta036	2829	2831	0,071%
ta037	2725	2728	0,110%
ta038	2683	**2683**	0,000%
ta039	2552	2554	0,078%
ta040	2782	**2782**	0,000%
EMR			**0,051%**

Tabla : Desempeño de las soluciones obtenidas para las instancias de 50x5.

Instancia	Cota superior (CS)	QL-NEH	Error relativo (ER)
ta041	3025	3055	0,992%
ta042	2892	2925	1,141%
ta043	2864	2900	1,257%
ta044	3064	3099	1,142%
ta045	2986	3021	1,172%
ta046	3006	3045	1,297%
ta047	3107	3148	1,320%
ta048	3039	3077	1,250%
ta049	2902	2935	1,137%
ta050	3091	3125	1,100%

EMR			**1,181%**

Tabla : Desempeño de las soluciones obtenidas para las instancias de 50x10.

Instancia	**Cota superior (CS)**	**QL-NEH**	**Error relativo (ER)**
ta051	3875	3890	0,387%
ta052	3715	3728	0,350%
ta053	3668	3692	0,654%
ta054	3752	3790	1,013%
ta055	3635	3674	1,073%
ta056	3698	3740	1,136%
ta057	3716	3760	1,184%
ta058	3709	3752	1,159%
ta059	3765	3810	1,195%
ta060	3777	3823	1,218%
EMR			**0,937%**

Tabla : Desempeño de las soluciones obtenidas para las instancias de 50x20.

Instancia	**Cota superior (CS)**	**QL-NEH**	**Error relativo (ER)**
ta061	5493	**5493**	0,000%
ta062	5268	5274	0,114%
ta063	5175	5177	0,039%
ta064	5014	5021	0,140%
ta065	5250	5255	0,095%
ta066	5135	5139	0,078%
ta067	5246	**5246**	0,000%
ta068	5034	5038	0,079%

ta069	5448	5449	0,018%
ta070	5322	**5322**	0,000%
EMR			**0,056%**

Tabla : Desempeño de las soluciones obtenidas para las instancias de 100x5.

Instancia	**Cota superior (CS)**	**QL-NEH**	**Error relativo (ER)**
ta071	5770	5807	0,641%
ta072	5349	5390	0,766%
ta073	5676	5691	0,264%
ta074	5791	5882	1,571%
ta075	5468	5553	1,554%
ta076	5303	5327	0,453%
ta077	5595	5634	0,697%
ta078	5623	5689	1,174%
ta079	5875	5950	1,277%
ta080	5845	5903	0,992%
EMR			**0,939%**

Tabla : Desempeño de las soluciones obtenidas para las instancias de 100x10.

Instancia	**Cota superior (CS)**	**QL-NEH**	**Error relativo (ER)**
ta081	6286	6353	1,066%
ta082	6241	6329	1,410%
ta083	6329	6362	0,521%
ta084	6306	6391	1,348%
ta085	6377	6457	1,255%
ta086	6437	6511	1,150%

ta087	6346	6424	1,229%
ta088	6481	6570	1,373%
ta089	6358	6455	1,526%
ta090	6465	6515	0,773%
EMR			**1,165%**

Tabla : Desempeño de las soluciones obtenidas para las instancias de 100x20.

De las tablas presentadas anteriormente se puede concluir que el algoritmo propuesto es capaz de obtener buenos resultados, ya que logró alcanzar la mejor cota superior conocida en 19 casos y el ER promedio de todas las instancias es menor que 1%.

3.3.2 Estudio comparativo entre QL-NEH y las variantes NEH y ALA.

El desempeño del QL-NEH fue comparado con el algoritmo NEH (Nawaz et al., 1983) y con una heurística de mejora basada en la idea del aprendizaje adaptativo conocida como ALA propuesta por (Agarwal et al., 2006).

ALA emplea una heurística de un paso para ofrecer una buena solución inicial en el espacio de búsqueda y usa un parámetro de peso para perturbar los datos del problema original y obtener así mejores soluciones; este algoritmo ha obtenido buenas soluciones para varios conjuntos de instancias de prueba.

Para esta comparación se utiliza el test de *Wilcoxon* con un nivel de significación de 0.05, los resultados son mostrados en la Tabla .

Variables	R^+	R^-	**Significación**
QL-NEH – NEH	0,00	45,00	**0.000**
QL-NEH – ALA	18,09	41,74	**0.000**

Tabla : Resultados del test de *Wilcoxon* para la comparación entre QL-NEH, NEH y ALA.

Como puede observarse existen diferencias significativas con los dos algoritmos en cuanto a los resultados obtenidos y al tratarse de un problema de minimización es necesario comprobar que los rangos negativos sean mayores que los positivos, lo que ocurre en este caso, por lo que se

puede concluir que el algoritmo propuesto es mejor que las dos metodologías analizadas en esta sección.

3.3.3 Estudio comparativo entre QL-NEH y las variantes M-MMAS, PACO y HGA.

El desempeño del algoritmo QL-NEH fue comparado con las metaheurísticas basadas en colonias de hormigas propuestas por (Rajendran and Ziegler, 2004) conocidas como M-MMAS y PACO, y un algoritmo genético híbrido (HGA) propuesto por (Wang et al., 2006).

El algoritmo HGA usa simultáneamente múltiples operadores genéticos y emplea una estructura de vecindad basada en el modelo de grafo para perfeccionar la búsqueda local. Este método fue comparado con técnicas de Recocido Simulado y Búsqueda Tabú y los resultados obtenidos demostraron la efectividad de HGA.

Para realizar esta comparación se ejecutó el algoritmo QL-NEH durante $n * m$ episodios para cada instancia de *Taillard* y se presenta el desempeño de los tres procedimientos heurísticos mencionados para estas instancias en comparación con las soluciones en cuanto al *makespan*.

La Tabla muestra un estudio comparativo entre nuestro enfoque y los otros tres algoritmos. Se muestra el EMR para cada tamaño de problema y luego se calcula un promedio total para todas las instancias.

n	m	M-MMAS	PACO	HGA	QL-NEH
20	5	0,762%	0,704%	2,770%	0,000%
	10	0,890%	0,843%	3,730%	0,354%
	20	0,721%	0,720%	2,890%	0,597%
50	5	0,144%	0,090%	4,230%	0,051%
	10	1,118%	0,746%	4,410%	1,181%
	20	2,013%	1,855%	2,790%	0,937%
100	5	0,084%	0,072%	6,020%	0,056%
	10	0,451%	0,404%	4,920%	0,939%
	20	1,030%	0,985%	3,960%	1,165%

Promedio	**0,801%**	**0,713%**	**3,969%**	**0,587%**

Tabla : Desempeño de los algoritmos QL-NEH, M-MMAS, PACO y HGA.

El desempeño del QL-NEH fue mejor que el de todos los métodos propuestos. Para siete grupos de instancias el EMR del enfoque propuesto fue menor que 1%. Solo para dos grupos de problemas el EMR fue mayor que 1%. Aun así el EMR promedio fue menor que 1% mostrando que nuestro enfoque obtuvo mejores resultados.

3.4 Conclusiones parciales.

Del análisis estadístico realizado a los resultados obtenidos por el algoritmo QL-NEH aplicado al problema de secuenciación tipo *Flow Shop* se concluye que:

- Los resultados obtenidos por QL-NEH son significativamente superiores a los obtenidos por los enfoques NEH y NEH-ALA.
- Comparando los resultados del algoritmo QL-NEH con los reportados por las variantes M-MMAS, PACO y HGA, se concluye que el primero muestra resultados superiores con respecto a todas las técnicas analizadas.

CONCLUSIONES

La aplicación del Aprendizaje Reforzado ha reportado buenos resultados en la solución de problemas de secuenciación de tareas. En particular, en este trabajo se seleccionó el algoritmo Q-*Learning* para solucionar problemas de secuenciación de tipo *Flow Shop*, arribando a las conclusiones siguientes:

- Este método constituye una interesante alternativa para resolver problemas complejos de optimización, específicamente, problemas de *scheduling*.
- Las soluciones obtenidas por la variante propuesta se comprobaron estadísticamente obteniéndose diferencias significativas entre el algoritmo propuesto y otros enfoques de la literatura especializada que resuelven dicho problema, mostrando resultados superiores.
- Se implementó una aplicación de fácil utilización e interfaz amigable que le facilita el trabajo al usuario y que puede servir como entorno de trabajo integrado para resolver otros tipos de problemas de secuenciación de tareas usando Aprendizaje Reforzado.

RECOMENDACIONES

Aplicar el enfoque propuesto al *Hybrid Flow Shop Scheduling Problem*, el cual es un problema similar al problema de secuenciación de tipo *Flow Shop*, pero difiere principalmente en que en cada etapa puede existir más de un recurso disponible para ejecutar una operación y estos no necesariamente son idénticos, además, los trabajos pueden saltar etapas, por tanto se necesita un nivel extra de decisión para poder realizar una buena asignación máquina-operación.

Extender la herramienta propuesta con otros tipos de problemas de secuenciación, así como con técnicas de robustez para las soluciones, con el objetivo de proveer un ambiente de trabajo integrado para resolver cualquier tipo de problema de *scheduling* usando AR.

BIBLIOGRAFÍA

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**,** 801-815.

ALFONSO, M. 2001. *Un modelo de integración de técnicas de CLAUSURA y CSP de restricciones temporales: Aplicación a problemasde Scheduling.* Doctorado, Universidad de Alicante.

ANCĂU, M. 2012. On Solving Flowshop Scheduling Problems. *Proceedings of the Romanian Academy. Series A,* 13**,** 71-79.

BARTO, A. G. 1998. *Reinforcement learning: An introduction*, MIT press.

BEASLEY, J. E. 1990. OR-Library: distributing test problems by electronic mail. *Journal of the operational research society***,** 1069-1072.

BRUCKER, P. 2007. *Scheduling Algorithms,* Berlin, Springer-Verlag.

CAMPBELL, H. G., DUDEK, R. A. & SMITH, M. L. 1970. A heuristic algorithm for the n job, m machine sequencing problem. *Management science,* 16**,** B-630-B-637.

DANNENBRING, D. G. 1977. An evaluation of flow shop sequencing heuristics. *Management science,* 23**,** 1174-1182.

DAVENPORT, A. J. & BECK, J. C. 2000. A survey of techniques for scheduling with uncertainty. *Unpublished manuscript. Available from http://tidel. mie. utoronto. ca/publications. php*.

DEMŠAR, J. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research,* 7**,** 1-30.

FONSECA, Y. C. 2013. Influencia de los parámetros principales de un Algoritmo Genético para el Flow Shop Scheduling. *Revista Cubana de Ciencias Informáticas,* 8.

GABEL, T. 2009. Multi-agent reinforcement learning approaches for distributed job-shop scheduling problems.

GAJPAL, Y. & RAJENDRAN, C. 2006. An ant-colony optimization algorithm for minimizing the completion-time variance of jobs in flowshops. *International Journal of Production Economics,* 101**,** 259-272.

GARCÍA, S., FERNÁNDEZ, A., LUENGO, J. & HERRERA, F. 2009. A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. *Soft Computing,* 13**,** 959-977.

GARCIA, S. & HERRERA, F. 2008. An Extension on" Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. *Journal of Machine Learning Research,* 9.

GAREY, M. R., JOHNSON, D. S. & SETHI, R. 1976. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research,* 1**,** 117-129.

GROSSMAN, I. 2005. Enterprise-wide optimization: A new frontier in process systems engineering. *AIChE Journal,* 51**,** 1846-1857.

GUPTA, J. N. 1971. A functional heuristic algorithm for the flowshop scheduling problem. *Operational Research Quarterly***,** 39-47.

HO, J. C. & CHANG, Y.-L. 1991. A new heuristic for the n-job, M-machine flow-shop problem. *European Journal of Operational Research,* 52**,** 194-202.

HUNDAL, T. S. & RAJGOPAL, J. 1988. An extension of Palmer's heuristic for the flow shop scheduling problem. *The International Journal Of Production Research,* 26**,** 1119-1124.

JAAKKOLA, T., JORDAN, M. I. & SINGH, S. P. 1994. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation,* 6**,** 1185-1201.

JENNINGS, N. R., SYCARA, K. & WOOLDRIDGE, M. 1998. A roadmap of agent research and development. *Autonomous agents and multi-agent systems,* 1**,** 7-38.

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

KAELBLING, L. P., LITTMAN, M. L. & MOORE, A. W. 1996. Reinforcement learning: A survey. *arXiv preprint cs/9605103*.

LATORRE, G. 2011. *El problema Flow Shop Flexible en dos etapas: Programación de las intervenciones quirúrjicas en un hospital.* Universidad del Bio-Bio.

LUENGO, J., GARCÍA, S. & HERRERA, F. 2009. A study on the use of statistical tests for experimentation with neural networks: Analysis of parametric test conditions and non-parametric tests. *Expert Systems with Applications,* 36**,** 7798-7808.

MARTÍNEZ, Y. 2012. *A Generic Multi-Agent Reinforcement Learning Approach Scheduling Problems.* Vrije Universiteit Brussel.

MIHAYLOV, M. 2012. *Decentralized Coordination in Multi-Agent Systems.* PhD thesis, Vrije Universiteit Brussel, Brussels.

NAWAZ, M., ENSCORE JR, E. E. & HAM, I. 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega,* 11**,** 91-95.

NOWICKI, E. & SMUTNICKI, C. 1996. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research,* 91**,** 160-175.

PALMER, D. 1965. Sequencing jobs through a multi-stage process in the minimum total time--a quick method of obtaining a near optimum. *OR***,** 101-107.

PINEDO, M. 2008. *Scheduling Theory, Algorithms, and Systems,* New Jersey, Prentice Hall Inc., Englewood Cliffs, U.S.A.

PONNAMBALAM, S., ARAVINDAN, P. & CHANDRASEKARAN, S. 2001. Constructive and improvement flow shop scheduling heuristics: an extensive evaluation. *Production Planning & Control,* 12**,** 335-344.

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**,** 426-438.

RIBEIRO, C. H. C. A tutorial on reinforcement learning techniques. Supervised Learning track tutorials of the 1999 International Joint Conference on Neuronal Networks, 1999.

RUIZ, R. & STÜTZLE, T. 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research,* 177**,** 2033-2049.

SHEN, W. 2002. Distributed manufacturing scheduling using intelligent agents. *Intelligent Systems, IEEE,* 17**,** 88-94.

SHESKING, D. 2006. *Handbook of parametric and nonparametric statistical procedures,* London, Chapman & Hall.

STÉFAN, P. 2003. Flow-Shop Scheduling based on Reinforcement Learning Algorithm. *Production Systems and Information Engineering,* 1**,** 83-90.

TAILLARD, E. 1990. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research,* 47**,** 65-74.

TORO, M., RESTREPO, G. & GRANADA, M. 2006. Adaptación de la técnica de Particle Swarm al problema de secuenciación de tareas. *Scientia et Technica UTP,* XII**,** 307-313.

URLINGS, T., RUIZ, R. & STÜTZLE, T. 2010. Shifting representation search for hybrid flexible flowline problems. *European journal of operational research,* 207**,** 1086-1095.

WANG, L., ZHANG, L. & ZHENG, D.-Z. 2006. An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. *Computers & Operations Research,* 33**,** 2960-2971.

WATKINS, C. J. C. H. 1989. Learning from Delayed Rewards.

WIDMER, M. & HERTZ, A. 1989. A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research,* 41**,** 186-193.

ZHANG, W. 1996. Reinforcement Learning for Job Shop Scheduling. *Architecture.*