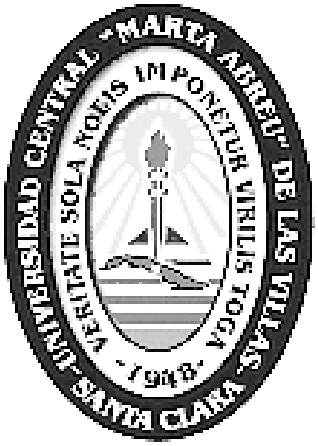
Universidad Central "Marta Abreu" de Las Villas

Facultad Matemática, Física y Computación

Centro de Estudios de Informática

Departamento de Ciencia de la Computación



Enfoque multi-objetivo basado en Aprendizaje Reforzado para problemas de secuenciación de tareas tipo Job Shop (MOQL).

Trabajo de Diploma

Autor: Liliana Ortega Sánchez

Tutoras: MSc. Beatriz M. Méndez Hernández

Dra. Yailen Martínez Jiménez

Santa Clara, 2014.

Dictamen

El que suscribe, Liliana Ortega Sánchez, hago constar que el trabajo titulado **"*Enfoque multi-objetivo basado en Aprendizaje Reforzado para problemas de secuenciación de tareas tipo Job 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 de la Autora

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

Dedicatoria

A mis padres, este trabajo es la recompensa de sus años de esfuerzo y dedicación... gracias por creer en mí, y apoyarme en todo momento de mi vida.

Agradecimientos

Agradezco sinceramente a todas las personas que han contribuido al desarrollo de este trabajo:

- ✓ *A mis padres, por ser mi sostén, mi guía, por todo su amor y dedicación, porque sin ustedes nada de esto sería posible. Gracias por inculcarme los deseos de superación, por ayudarme a crecer como la persona que soy. A ustedes debo toda mi vida, todos mis logros. Para ustedes son mis más cariñosos y grandes agradecimientos.*
- ✓ *A mi hermano que de una forma muy particular me ha dado su amor y cariño, a él gracias simplemente por existir y ser parte de mi vida.*
- ✓ *A mi Juan, que ha sido novio, amigo, compañero. Sin ti nada de esto hubiese sido posible. Gracias por tu amor, paciencia, soporte. Gracias por enseñarme a levantarme y seguir adelante sin importar los inconvenientes. Contigo compartí todos los pesares de estos cinco años y ahora me alegro de compartir toda la dicha de este momento. Gracias!!!!*
- ✓ *A mi familia en general, por su apoyo y amor incondicional, porque son personas muy importantes en mi formación.*
- ✓ *A mis amigas de siempre, que más que nada son hermanas, a Dagmara, a Diana, y Cely por estar a mi lado en todo momento regalándome de forma incondicional su apoyo y cariño.*
- ✓ *A mis tutoras Beatriz Méndez Hernández y Yailen Martínez Jiménez por su ayuda y paciencia, por respaldarme en todo momento y siempre confiar en mí. Por ser más que tutoras, amigas.*
- ✓ *A todos mis amigos de la Universidad que considero son para siempre, especialmente a Claudia, Alejandro, Ana Li y Jeniffer por caminar a mi lado todos estos años y a Lisbet por sus consejos y ánimos.*
- ✓ *A todos los profesores que me formaron profesionalmente a lo largo de la carrera y de mi vida.*

El único autógrafo digno de un hombre es el que deja escrito con sus obras.

José Martí.

RESUMEN

Los problemas de *scheduling* requieren organizar en el tiempo la ejecución de tareas que comparten un conjunto finito de recursos, y que están sujetas a un conjunto de restricciones impuestas por diversos factores. Este tipo de problemas aparecen con frecuencia en la vida real en numerosos entornos productivos y de servicios. El problema consiste en optimizar uno o varios criterios que se representan mediante funciones objetivo.

En esta investigación se estudia el *Job Shop Scheduling* con el objetivo de minimizar el *makespan*, el *flow time* y el *tardiness* de los trabajos. El hecho de considerar todos estos objetivos a la vez es el problema de optimización multi-objetivo en estudio.

El uso de agentes y de aprendizaje reforzado para dar solución a este tipo de problemas resulta un enfoque muy natural. El comportamiento de los agentes a la hora de seleccionar una acción estará determinado por uno de los algoritmos de aprendizaje reforzado más usados para resolver este tipo de problemas; el *Q-Learning*. En esta tesis se describen varias funciones de recompensa usadas para cada uno de los objetivos a optimizar.

Para dar solución al problema multi-objetivo se hace uso de la Optimalidad de Pareto obteniéndose un frente de soluciones no dominadas.

El desempeño del algoritmo propuesto es evaluado usando las instancias de problemas *Job Shop* que se encuentran en la OR-Library. Los resultados obtenidos son evaluados en términos de dos de las métricas propuestas en la literatura.

Dos casos de estudio para el objetivo *tardiness* son presentados.

Palabras Clave: *Job Shop*, Q-*Learning*, multi-objetivo, *scheduling,* Pareto.

ABSTRACT

Scheduling problems require the organization in time of a set of tasks which share a finite set of resources, these tasks are subject to a set of constraints imposed by several factors. This type of problem is usually found in real world environments, where the goal is to optimize one or several criteria represented through objective functions.

In this work, the Job Shop scheduling problem is studied with the objective of minimizing the makespan and the mean flow time of the jobs. The simultaneous consideration of these objectives is the multi-objective optimization problem under study.

The use of agents and reinforcement learning to solve this type of problems is a natural approach. One of the most used reinforcement learning algorithms, the Q-Learning, determines the behavior of the agents when selecting the actions. In this thesis, different reward functions are described for each of the objectives to be optimized.

To solve the multi-objective problem the Pareto Optimality is used, obtaining a non-dominated solution front. The performance of the proposed algorithm is evaluated by solving benchmark job shop scheduling problem instances provided by the OR-library. The results obtained are evaluated in terms of two metrics proposed in the literature. Two case studies for the tardiness objective are proposed.

Keywords: Job Shop, Q-Learning, Multiple Objectives, Scheduling, Pareto.

TABLA DE CONTENIDOS

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

El problema de la secuenciación de tareas consiste en la asignación de recursos a diferentes actividades que se ejecutan simultáneamente a lo largo del tiempo. El rango de aplicación de la teoría de secuenciación abarca áreas de conocimiento desde producción hasta computación pasando por logística, servicios y otros.

Los problemas de secuenciación de tareas consisten en la programación temporal de las operaciones o tareas en las que se descomponen un conjunto de trabajos teniendo en cuenta que éstas deben ser ejecutadas en varias máquinas y que cada máquina solamente puede ejecutar una tarea simultáneamente. Se busca aquella solución que dé lugar a un tiempo total de ejecución (C_{max}) mínimo, aunque pueden existir otras funciones objetivo como la minimización de la tardanza de los trabajos o de la suma de los tiempos de finalización de estos. El *Job Shop Scheduling Problem* (JSSP) consta básicamente de un grupo de trabajos, donde cada uno tiene un conjunto de operaciones a ser procesadas en un conjunto de recursos limitados, a los que se denominan máquinas. Cada trabajo tiene el orden en el que se deben ejecutar las operaciones, dichas operaciones tienen un tiempo de procesamiento en cada una de las máquinas y este no es modificable. Se trata de uno de los problemas de optimización combinatoria más difíciles de resolver.

Muchos problemas de la vida real, como el JSSP, persiguen varios objetivos a la vez, por lo que han hecho de la optimización multi-objetivo un área interesante de investigación. Un enfoque común es combinar los objetivos dentro de una función de escalarización donde se optimiza la suma de estos usando un vector de peso. Esta combinación no siempre muestra un buen desempeño debido a que se necesitan hacer varias iteraciones para obtener un buen resultado. Un enfoque multi-objetivo adecuado permitiría buscar la Frontera Óptima de Pareto (Van Veldhuizen and Lamont, 1998).

La mayoría de los algoritmos que existen para tratar estas problemáticas (Bagchi, 1999, Coello, 2006, Fonseca and Fleming, 1993, Horn et al., 1994, Knowles and Corne, 2000b, Kumar and Rockett, 2002, Schaffer, 1985) están basados en algoritmos evolutivos, esencialmente algoritmos genéticos. Estos tienen buenos resultados pero necesitan una codificación del problema a resolver que no siempre es fácil de obtener, también se necesitan parámetros asociados a este tipo de algoritmo como el factor de peso usado en funciones de

agregación, las prioridades usadas en el ordenamiento lexicográfico y en el ranking de Pareto. Encontrar los valores apropiados para estos parámetros es la mayor parte del tiempo tan difícil como encontrar la solución del problema.

Recientemente el Aprendizaje Reforzado (Barto, 1998) (RL por sus siglas en inglés) ha recibido considerable atención. Existen varias propuestas conocidas para resolver problemas de optimización multi-objetivo con RL (Horn et al., 1994, Mariano and Morales, 1999a, Mariano and Morales, 2000, Mariano and Morales, 1999b).

Debido a lo antes expuesto se plantea el siguiente problema de investigación: ***La ausencia de un algoritmo multi-objetivo basado en la técnica del aprendizaje reforzado para resolver problemas de secuenciación de tareas tipo Job Shop teniendo en cuenta las restricciones reales de estos problemas.***

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

Diseñar un enfoque multi-objetivo basado en aprendizaje reforzado y la Frontera de Pareto para resolver problemas de secuenciación de tareas tipo *Job Shop* que optimice tres de los principales objetivos existentes.

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

- Definir los objetivos que van a ser optimizados de acuerdo a la importancia de los mismos en la literatura.
- Implementar un algoritmo multi-objetivo usando RL y basado en la Frontera de Pareto que optimice los objetivos definidos anteriormente.
- Validar el algoritmo de acuerdo a las métricas propuestas en la literatura y mediante la proposición de casos de estudio.

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

- ¿Cuáles son los principales objetivos a optimizar de acuerdo al criterio de los expertos y a la literatura?
- ¿Cumple la Frontera de Pareto hallada por nuestro algoritmo para una instancia específica las métricas propuestas por la literatura?

- ¿Se pueden obtener con nuestro algoritmo resultados comparables con los obtenidos por otras técnicas multi-objetivo para ambientes tipo *Job Shop*?

Uno de los ambientes de manufactura en el que es necesario optimizar el proceso de producción es en la industria azucarera. La producción cubana de azúcar cobra fuerzas, alentada por nuevos precios mundiales y sobre la base de un reajuste integral a la estructura administrativa y agroindustrial del sector, no hay mejor coyuntura para pensar bien cada movimiento, a fin de convertir el más mínimo detalle en una invariable garantía de eficiencia.

Entre los pasos que aún requieren calibración está la prestación de servicios técnicos a la industria azucarera, cuyas deudas en cuanto a planificación, suficiencia y eficiencia, todavía ponen cierto freno al despliegue necesario.

Resulta de gran interés contar con enfoques computacionales que resuelvan problemas reales de optimización, pues esto permite simular la ejecución de la planificación de las tareas a ser ejecutadas y experimentar distintas configuraciones sin agotar recursos materiales. Además, permite obtener distintas soluciones en dependencia de los objetivos a ser optimizados.

La tesis está estructurada en tres capítulos. En el capítulo uno se realiza un estudio de los problemas de secuenciación de tareas, su clasificación y los diferentes métodos que existen para resolverlos, también se hace una análisis de los diversos objetivos que pueden ser optimizados para este tipo de problemas, junto con la optimización multi-objetivo y los métodos que se utilizan para resolverla. Por último, se hace un estudio de los sistemas multi-agente y de RL haciendo énfasis en el algoritmo *Q-Learning* y sus variantes para resolver problemas multi-objetivo. Seguidamente, en el capítulo dos se hacen una descripción de las instancias del problema y se especifican los principales aspectos que se tuvieron en cuenta a la hora de desarrollar el algoritmo. Finalmente, se propone un algoritmo multi-objetivo basado en RL y que forma una Frontera de Pareto (*Multi Objetive Q-Learning,* MOQL) y se describe dicho algoritmo mediante un ejemplo. Posteriormente, en el capítulo tres se describen en el marco experimental las formas de evaluar los resultados obtenidos, se comparan los resultados para dos de los objetivos con algoritmos presentados en la literatura y se propone un caso de estudio para futuras investigaciones. El documento culmina con las conclusiones y las recomendaciones del autor.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

En este capítulo se aborda todo lo referente a la base teórica que fundamenta esta investigación. Se hace una revisión de la bibliografía para determinar las técnicas de optimización que han sido utilizadas para resolver problemas de secuenciación de tareas en configuraciones de tipo *Job Shop* y un estudio sobre cuáles son los objetivos más deseables a optimizar para este tipo de problemas.

1.1 Problemas de secuenciación de tareas

Los problemas de secuenciación de tareas están presentes en muchos de los procesos que se llevan a cabo en la rutina diaria, donde usualmente se hace necesario realizar una serie de operaciones en un espacio de tiempo determinado y usando recursos limitados, tratando siempre de cumplir todos los objetivos.

Estos problemas son una subclase de problemas de satisfacción de restricciones que aparecen con frecuencia en entornos reales. Ejemplos de estos podemos encontrarlos en muchas áreas de la industria, la administración y la ciencia; algunos que van desde la organización de la producción hasta la planificación de multiprocesadores, industrias de semiconductores, asignación de puertas de embarque en un aeropuerto, o planificación de horarios ferroviarios. Otros pudieran ser (González, 2011):

- Fabricación de obleas para circuitos semiconductores, donde cada oblea precisa de una serie de tareas como limpieza, oxidación, metalización, etc. Los objetivos pueden ser maximizar la utilización de algunas máquinas que son cuello de botella o minimizar el tiempo de ejecución.
- Planificar el aterrizaje de un conjunto de aviones sujetos a restricciones temporales que dependen de las características de los aviones. Los objetivos pueden ser minimizar la penalización por desvío con respecto al horario planeado para los aviones o maximizar las condiciones de seguridad.
- Enrutamiento de paquetes de datos a través de líneas de comunicación, donde se trata de maximizar el uso de la red y de minimizar los tiempos de llegada de los mensajes.

Todos estos problemas son de naturaleza combinatoria; es decir, hay que elegir una entre un conjunto exponencialmente grande de combinaciones posibles, y por lo tanto los problemas de

secuenciación de tareas precisan de algoritmos de búsqueda inteligente para encontrar soluciones aceptables en un tiempo razonable. Estos problemas son típicamente clasificados como NP-Completos (Garey et al., 1976), lo que significa que encontrar una solución óptima es imposible si no se usa un algoritmo esencialmente enumerativo, y el tiempo de cómputo se incrementa exponencialmente de acuerdo al tamaño del problema, representando los problemas de secuenciación en ambientes de manufactura uno de los más complejos de resolver (Martínez, 2012, Shen, 2002). Para su resolución se han aplicado prácticamente todas las técnicas de Inteligencia Artificial, con mayor o menor éxito (González, 2011)

Los problemas de secuenciación se pueden clasificar de acuerdo al ambiente de las máquinas, las características de los trabajos y la función objetivo. Esta clasificación se conoce normalmente como $\alpha|\beta|\gamma$ (Graham et al., 1979), donde α representa el ambiente de las máquinas, β las características de los trabajos y γ el criterio a optimizar (Martínez, 2012).

De forma general el problema de secuenciación de tareas se puede describir como: un conjunto de N trabajos $\{J_1, \dots, J_n\}$ y un conjunto R de M recursos físicos o máquinas $\{R_1, \dots\dots, R_M\}$. Cada trabajo J_i consta de un conjunto de tareas u operaciones $\{\theta_{i1}, \dots, \theta_{iM}\}$. Cada una de las operaciones Θ_{ij} tiene un tiempo de procesamiento $P_{\Theta_{ij}}$ y debe ser procesada sobre una única máquina del conjunto R. Cada trabajo sólo puede ser procesado sobre una máquina y cada máquina sólo puede procesar un trabajo en un instante (González, 2011).

Una planificación consiste en encontrar para cada trabajo un tiempo o un intervalo de tiempos en los que éste puede procesarse en una o varias máquinas. El objetivo es encontrar una planificación sujeta a una serie de restricciones y que optimice una o varias funciones objetivo.

El enfoque clásico para resolver un problema de secuenciación consiste en construir un modelo matemático, usualmente basado en Programación Mixta en Enteros y luego aplicar un algoritmo de búsqueda como el algoritmo de Ramificación y Acotamiento para encontrar la solución óptima (Mendez et al., 2006). Sin embargo, a medida que el número de recursos disponibles en el problema se incrementa, o el número de operaciones a ser planificadas crece, este enfoque no será capaz de encontrar la solución óptima en un tiempo computacional razonable (Ruiz et al., 2008). Es aquí donde los métodos heurísticos se convierten en el centro

de atención, pues son capaces de encontrar buenas soluciones de forma eficiente (Martínez, 2012, Zhang, 1996).

1.1.1 Tipos de problemas de secuenciación de tareas que existen

El término secuenciación de tareas abarca una amplia clase de problemas de optimización combinatoria, muy distintos entre sí por complejidad y estructura. Todos ellos se presentan como formas tangibles en la optimización de procesos de producción industrial.

Estos problemas son una subclase de problemas de satisfacción de restricciones que aparecen con frecuencia en entornos reales. Se pueden dividir los problemas de secuenciación con respecto a las restricciones y a las funciones objetivo a optimizar.

En el primero de los casos se encuentran los siguientes tipos de restricciones:

- Fecha de liberación rj (*release date*). Indica el instante de tiempo (a partir de un instante inicial 0) antes del cual no es posible iniciar la ejecución del trabajo j. Por ejemplo, si la materia prima necesaria para efectuar el trabajo j arriba dentro de tres días, el trabajo j no puede comenzarse hasta entonces.
- Fecha deseada para la terminación dj (*due date*). Indica el instante de tiempo (a partir de un instante inicial 0) antes del cual la ejecución del trabajo j debería estar terminada. En general, la violación de un tiempo de esta conlleva algún costo (pérdida de confianza por parte del cliente, etc.)
- Peso wj (*weight*). Representa la importancia relativa del trabajo j respecto de los otros.
- Tiempo de configuración sij (*set-up*). Esta característica está presente sobre todo en problemas de una única máquina e indica que seguirá al trabajo i el trabajo j, entonces, entre estos dos trabajos, es necesario reconfigurar la máquina, lo cual requiere un tiempo sij.
- Preferencia (*preemption*). En ciertos casos, se consiente en interrumpir un trabajo para permitir la ejecución de uno más urgente. El problema en esos casos se llama *preemptive.*

- Vínculos de precedencia entre trabajos. Pueden existir relaciones de precedencia entre trabajos. Por ejemplo, un vínculo de este tipo entre los trabajos i y j marca que el trabajo i no puede comenzar hasta haber finalizado el trabajo j.
- Bloqueo (*blocking*). En un problema de *flow shop*, los trabajos en espera para ser efectuados en una máquina, son hospedados en un buffer. Si en un instante dado el buffer de la máquina *i* está lleno, un trabajo terminado en la máquina $i-1$ no puede encontrar un puesto en el buffer de la máquina *i* y está por lo tanto condenada a permanecer bloqueando la máquina $i-1$, la cual no podrá iniciar una nueva tarea hasta tanto no se desembarace del trabajo. Este fenómeno se llama *blocking*.
- Opción sin espera (*no-wait*). Si hubiese un buffer a la entrada de la máquina *i*, la máquina $i-1$ estaría ocupada hasta que se liberase la máquina *i*. Aún más restrictiva es la situación *no-wait* en la cual a los trabajos no se les permite siquiera esperar dentro de una máquina y en cambio, debe garantizarse que al instante de completar una operación en una máquina, la máquina sucesiva estará ya disponible para continuar el trabajo.

En cuanto a los posibles objetivos a optimizar se tratarán más adelante en el acápite 1.2.

Los problemas que derivan del problema general de secuenciación de tareas son casos especiales de éste que añaden alguna restricción. Los problemas derivados más referenciados en la literatura, ordenados de más general a más particular, son los siguientes:

El ***Open Shop***, el ***Job Shop Scheduling Problem*** **(JSSP)** y el ***Flow Shop*** donde la principal diferencia entre ellos es el orden en que se ejecutan los trabajos y la forma de asignar estos a las máquinas.

En el ***Open Shop*** cada una de las tareas de un trabajo se procesará sobre una máquina diferente, no existiendo relaciones de precedencia entre las operaciones. El problema consiste en encontrar órdenes de los trabajos (ordenaciones de las operaciones pertenecientes al mismo trabajo) y órdenes de las máquinas (ordenaciones de las operaciones que han de ser procesadas en la misma máquina) que optimicen una cierta función objetivo. En el ***Flow Shop*** existen restricciones de precedencia que consisten en que las tareas de todos los trabajos deben utilizar las máquinas en orden secuencial; es decir, la primera tarea de todos los trabajos se debe

ejecutar en la máquina 1, la segunda tarea en la máquina 2, y así sucesivamente. Entonces, el problema consiste en encontrar un orden de trabajos que optimice una determinada función objetivo (González, 2011).

Este epígrafe se enfocará en los problemas de tipo ***Job Shop*** que es uno de los más conocidos e investigados debido a su dificultad y aplicabilidad a una gran cantidad de situaciones reales.

Los JSSP tienen la característica de que los trabajos son unidireccionales. Otra característica importante es que la máquina que inicia no lo hace precisamente con el primer trabajo, ya que el orden de procesamiento de los trabajos en las máquinas tiene una secuencia diferente para cada instancia (Rivera and Eléctrica, 2004).

En (González, 2011) se define el JSSP clásico como sigue: cada trabajo J_i está compuesto por M operaciones $\{\Theta_{i1}, \ldots\ldots, \Theta_{iM}\}$ con tiempos de procesamiento $P_{\Theta_{ij}}$ donde cada tarea $\theta_{ij}(j = 1, \ldots, M)$ debe ser procesada en una máquina distinta. En el caso de este problema también se añade una restricción de precedencia, y es que las operaciones dentro de un determinado trabajo tienen un cierto orden y deben ejecutarse de forma secuencial. El problema consiste en encontrar un orden de tareas para cada máquina que optimice una cierta función objetivo.

Se listan a continuación las restricciones y definiciones que distinguen el JSSP:

- **Máquina**: Es cada uno de los entes que realizan alguna tarea. Disponemos de un grupo de ellas y cada una con una función específica. Supóngase que una máquina cualquiera no puede realizar más de una única operación a la vez.
- **Operación**: Es cada uno de los pasos necesarios para lograr un producto terminado. Una operación consiste en usar una máquina determinada durante un período determinado de tiempo ininterrumpido.
- **Trabajo**: Establece el objetivo de lograr un producto en particular. Por consiguiente, indica el conjunto de operaciones para lograr dicha meta.
- **Programa**: Es una asignación que fija a cada operación un intervalo de tiempo para ser efectuada.
- No está permitido que dos operaciones del mismo trabajo se procesen simultáneamente.

- Ninguna operación tiene prioridad sobre las demás.
- Ningún trabajo puede ser procesado más de una vez en la misma máquina.
- Cada trabajo es procesado hasta concluirse, aunque haya que esperar y retardarse entre las operaciones procesadas.
- Un trabajo puede iniciarse en cualquier momento siempre y cuando esté disponible la máquina y no se haya especificado un tiempo de inicio.
- Los trabajos tienen que esperar a que la siguiente máquina esté disponible para que este sea procesado.
- Hay solo una máquina de cada tipo.
- Las máquinas pueden estar ociosas en cualquier momento del plan de trabajo.
- Las máquinas están disponibles en cualquier momento.
- Las restricciones tecnológicas están bien definidas y son previamente conocidas, además de que son inamovibles (Rivera and Eléctrica, 2004).

Un ejemplo de un problema de secuenciación tipo *Job Shop* con tres trabajos y tres máquinas es presentado en la **Tabla 1.1**. Cada fila indica la secuencia en la que pasarán las operaciones por las máquinas y asociado a cada operación está el tiempo en que debe ser procesada.

Trabajo	**Máquina (tiempo de procesamiento)**		
1	1(3)	2(3)	3(3)
2	1(2)	3(3)	2(4)
3	2(3)	1(2)	3(1)

Tabla 1.1 JSSP de 3x3.

Como se dice en (Garrido et al., 2000) el JSSP es uno de los problemas NP-Completos más complejos y posee consecuentemente un constante reto intelectual.

1.1.2 Formas de representación de un JSSP

La forma más típica de representar soluciones para problemas de planificación de tareas es el diagrama de Gantt. En estos diagramas el eje *X* representa el tiempo y el eje *Y* el recurso.

Dentro del diagrama hay un recuadro por cada tarea de cada trabajo. Dicho recuadro se sitúa en la fila que corresponde a su recurso, y en el eje *X* abarca todo el tiempo comprendido desde el inicio de la ejecución de dicha tarea hasta su final. Dentro de dichos recuadros se puede introducir algo de información adicional, como la identificación de la tarea (González, 2011). Un ejemplo de una solución al problema reflejado en la **Tabla 1.1** se representa en la **Figura 1.1.**

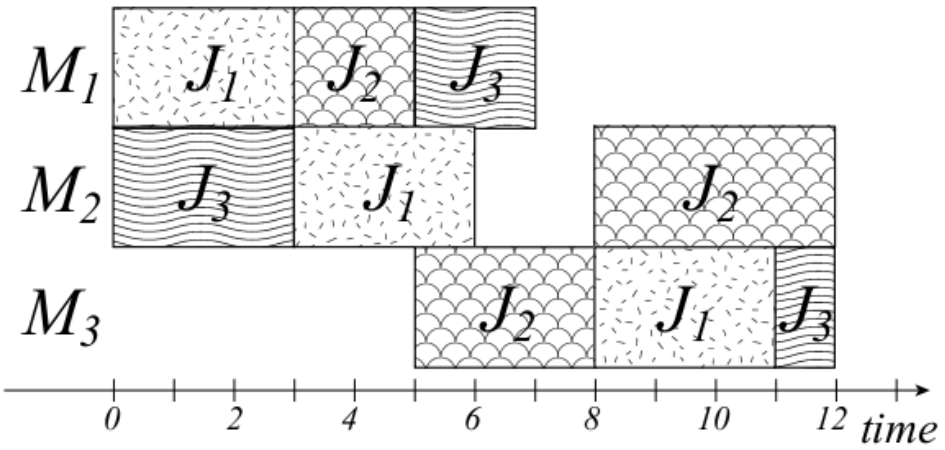


Figura 1.1 Diagrama de Gantt para un JSSP de 3x3.

Otra forma de representar una solución de un problema de secuenciación de tareas es mediante una matriz, en donde cada fila representa los recursos, y las columnas indican el orden de planificación dentro de dicho recurso. Esta representación es mucho más simple que el diagrama de Gantt, pero tiene el inconveniente de que no incluye ninguna información sobre tiempos de inicio y finalización de las tareas. Observar que la matriz se puede deducir de forma trivial a partir del diagrama de Gantt. Por ejemplo, la solución representada por el anterior diagrama de Gantt correspondería a la siguiente matriz:

$$\begin{pmatrix} j_1 & j_2 & j_3 \\ j_3 & j_1 & j_2 \\ j_2 & j_1 & j_3 \end{pmatrix}$$

Otra representación posible de una solución es mediante un vector que contendrá el orden de procesamiento de todas las tareas. Este orden puede codificarse también mediante una permutación con repetición, y en este último caso se incluirá únicamente información del trabajo de la tarea correspondiente. En concreto, la primera aparición del identificador de un trabajo hace referencia a la primera tarea de dicho trabajo, la segunda vez que aparece hace

referencia a la segunda tarea del trabajo, y así sucesivamente. Por ejemplo, si se nombra el primer trabajo como j_1, el segundo trabajo como j_2 y el tercer trabajo como j_3, una representación de la solución de la matriz anterior podría ser la siguiente:

$$(j_1, j_2, j_3, j_3, j_1, j_2, j_2, j_1, j_3)$$

1.1.3 Métodos de solución para problemas de secuenciación

Existen una gran cantidad de técnicas heurísticas aproximadas que, pese a que su proceso de búsqueda no garantiza la obtención de la solución óptima, ofrecen soluciones aceptables en tiempos de ejecución muy competitivos.

Dentro de los métodos existentes para resolver problemas de secuenciación se encuentran los exactos, que para problemas de pequeñas dimensiones encuentran una solución óptima del problema, pero para problemas de mayor magnitud no es factible su utilización pues no encuentran el óptimo en un tiempo considerable, estos son los métodos de programación lineal, programación dinámica, los de ramificación y acotamiento (Brucker et al., 1997), entre otros.

Dado que la mayoría de estos problemas tienen la característica de que a medida que aumentan su tamaño crece desproporcionadamente el tiempo de cómputo necesario para resolverlos, los métodos exactos dejan de ser una opción factible para problemas de la vida real donde existen instancias muy grandes y donde no siempre se trabaja en un ambiente controlado.

La idea entonces, es comprometer la calidad de las soluciones aspirando a soluciones cuasi-óptimas en pos de asegurar la posibilidad de alcanzarlas en intervalos de tiempo admisibles.

Por todo esto, se hace necesario tratar este tipo de problemas con un enfoque diferente y ahí es donde surgen los enfoques heurísticos estudiados en la Inteligencia Artificial para resolver este tipo de problemas de forma eficiente. En el caso de la Inteligencia Artificial se han usado métodos de Búsqueda Local como el Recocido Simulado (Dowsland and Adenso-Díaz, 2003) y la Búsqueda Tabú (Laguna, 1994), también se han desarrollado investigaciones antes citadas usando los Algoritmos Genéticos (Martínez, 2012).

Aún hoy continúan las investigaciones en aras de obtener algoritmos vertiginosos que a su vez aporten soluciones lo más cercanas posible a la óptima, para resolver problemas de la vida real que cada vez son más complejos.

1.2 Objetivos a optimizar en un problema tipo *Job Shop*

Los recursos y las tareas pueden tomar diferentes formas. Así pues, por ejemplo, los recursos pueden ser máquinas en un taller, pistas de aterrizaje en un aeropuerto, el equipo de trabajo en una obra en construcción, las unidades de procesamiento en un entorno de computación, y así sucesivamente. Las tareas pueden ser unidades de trabajo en un proceso de producción, los despegues y aterrizajes en un aeropuerto, las fases de un proyecto de construcción, ejecución de programas de ordenador, etc. Cada tarea puede tener asignada una prioridad, y un tiempo de inicio más temprano y un tiempo de fin límite (González, 2011).

Los objetivos también pueden tomar diversas formas. Por ejemplo, la minimización del tiempo de fin de la última tarea (criterio de optimización más típico que se conoce con el nombre de *makespan*), o la minimización del retraso de las tareas (criterio conocido como minimización del *tardiness*, o del *weighted tardiness* si asignamos una determinada prioridad a cada tarea). Otra posible función objetivo puede ser el *total flow time* que consiste en minimizar la suma de los tiempos de finalización de todos los trabajos.
Teniendo en cuenta el tipo de problema de secuenciación de tareas que se quiera resolver se pueden optimizar también los siguientes objetivos:

- *makespan* $C_{máx} = máx_{i\in\{1..n\}}C_i$. Máximo tiempo de completamiento, es equivalente al tiempo en que el último trabajo abandona el sistema.
- *total flow time* $F_{\Sigma} = \Sigma_{i\in\{1..n\}}F_i$. Es el tiempo consumido por todos los trabajos.
- *total lateness* $L_{\Sigma} = \Sigma_{i\in\{1..n\}}L_i$. La suma de todos los retrasos de los trabajos.
- *total tardiness* $T_{\Sigma} = \Sigma_{i\in\{1..n\}}T_i$. La suma de todas las tardanzas de los trabajos.
- *total earliness* $E_{\Sigma} = \Sigma_{i\in\{1..n\}}E_i$. La suma de todos los tiempos de terminación previos al tiempo comprometido.
- *maximum lateness* $L_{máx} = máx_{i\in\{1..n\}}L_i$. El retraso del más atrasado de los trabajos.
- *maximum tardiness* $T_{máx} = máx_{i\in\{1..n\}}T_i$. La tardanza del más tardado de los trabajos (Rivera and Eléctrica, 2004).

El trabajo de esta tesis se centra en tres objetivos anteriormente vistos: el *makespan*, el *tardiness* y el *flow time*, dado que estos son los que con mayor frecuencia se desean optimizar en ambientes de manufactura.

El *makespan* es la función objetivo por excelencia, y en general es la más estudiada en la literatura. Se desea obtener una planificación factible tal que el tiempo de fin de todos los trabajos, es decir, el *makespan*, denotado C_{max}, sea mínimo. Este problema entonces se denotaría por $J|S_{ij}|C_{máx}$ según la notación α|β|γ (Graham et al., 1979).

Esta minimización generalmente garantiza una alta utilización de los recursos de producción, una rápida satisfacción de la demanda de los clientes y la reducción de inventarios en curso (Estévez, 2007).

El *tardiness* es otra función objetivo relacionada con las fechas de entrega (*due dates*). A pesar de que el servicio al cliente con respecto a cumplir *due dates*, tiene cada vez una mayor importancia, los trabajos de investigación que tratan sobre la minimización del *tardiness* en problemas *Job Shop* es más bien escasa. En este caso en particular el objetivo es minimizar el *tardiness* total, es decir:

$$\sum_{i=1,\dots,N} T_i$$

Utilizando la notación habitual para los problemas de secuenciación, este problema se denota por $J|S_{ij}|\sum T_i$. Hay poca literatura disponible sobre esta función objetivo, a pesar de que en la vida real se considera más importante que la optimización del *makespan*, sobre todo porque dar un buen servicio al cliente es la prioridad. Evitar la demora en la finalización de ciertos trabajos prioritarios puede ser mucho más importante que conseguir la minimización del tiempo de finalización del último trabajo en ser ejecutado.

El total *flow time* es otra función objetivo que puede tener mucho interés en problemas reales. Por ejemplo, en aplicaciones en donde es muy importante el servicio al cliente. Esta función no considera *due dates*, sino que consiste en minimizar la suma del tiempo de finalización de todos los trabajos. Uno de los problemas encontrados aquí es que existen pocas referencias en la literatura de métodos específicos para resolverla.

El total *flow time* se define como la suma de los tiempos de finalización de todos los trabajos, es decir, se pretende minimizar:

$$\sum_{i=1,\ldots,N} C_i$$

Este problema se denota por $J|S_{ij}|\sum C_i$.

1.2.1 Ejemplos reales de problemas de secuenciación de tareas

En (Ingolotti, 2007), (Pinedo, 2008) o Sierra en (Sierra Sánchez, 2009) se describen varios ejemplos de problemas de esta naturaleza. A continuación se explican brevemente algunos de los más interesantes.

Fábrica de Bolsas de Papel. Considérese una fábrica dedicada a producir bolsas de papel. El material básico para cada una de las operaciones son rollos de papel. El proceso productivo consiste en tres fases: impresión del logo, pegado de los laterales de la bolsa, y la costura del final o de ambos finales de la bolsa. Cada fase consiste en una serie de máquinas no necesariamente idénticas. Las máquinas de una fase se pueden diferenciar en la velocidad a la que operan, el número de colores que pueden imprimir o el tamaño de la bolsa que producen. Cada orden de producción indica la cantidad de bolsas de cada tipo a fabricar y a entregar en una fecha determinada. Los tiempos de procesamiento de las diferentes operaciones son proporcionales al tamaño de la orden, es decir, al número de bolsas en la orden. Un retraso en la entrega implica una sanción en forma de pérdida de confianza. La magnitud de la penalización depende de la importancia de la orden o del cliente y del retraso en la entrega. Uno de los objetivos del sistema de planificación es minimizar la suma de estas sanciones.

Planificación de Tareas en una CPU. Una de las funciones de un sistema operativo multitarea en un ordenador es decidir el tiempo que la CPU dedica a los diferentes programas que han de ser ejecutados. Generalmente, no se conoce de antemano el tiempo de ejecución exacto de cada uno de ellos. Sin embargo, una distribución aproximada de dichos tiempos de ejecución sí puede ser conocida de antemano, teniendo en cuenta sus medias y variaciones. Además, cada tarea por lo general tiene un nivel de prioridad determinado, aunque el sistema operativo permite a los usuarios establecer la prioridad de cada tarea. En este caso, el objetivo es minimizar la suma de los tiempos ponderados de fin de todas las tareas. Para evitar que una tarea breve permanezca mucho tiempo, se divide cada tarea en pequeños trozos. Una vez hecho esto, el sistema operativo rota los trozos en la CPU, de modo que en cualquier intervalo de tiempo la CPU pase cierta cantidad de tiempo con cada tarea. De esta manera, si el tiempo

de procesamiento de una de las tareas es muy corto, la tarea será capaz de dejar el sistema rápidamente. La interrupción de una tarea en ejecución se conoce como expulsión. Es evidente que una política óptima en este entorno hace uso intensivo de expulsiones. En la práctica sucede con frecuencia que la elección de una planificación tiene un impacto significativo en el rendimiento del sistema, por lo que tiene sentido dedicar tiempo y esfuerzo a buscar una buena planificación, en lugar de elegir simplemente una al azar.

1.3 Optimización multi-objetivo

Muchos problemas de secuenciación de tareas en la vida real son multi-objetivo por naturaleza propia. Es decir, existen varios criterios que deben ser tomados en consideración cuando se resuelve un problema de tipo secuenciación de tareas. Tradicionalmente, estos problemas se han abordado como problemas de optimización de un solo objetivo, después de combinar los criterios múltiples en un solo valor escalar. La mayoría de estas técnicas se han probado con gran éxito. Sin embargo, la aplicación de estos métodos para resolver este tipo de problemas es aún poco explorada.

La optimización multi-objetivo difiere de la optimización de un solo objetivo en muchas formas. Para dos o más objetivos en conflicto, a cada objetivo le corresponde una solución óptima diferente, pero ninguna de estas soluciones es óptima para todos los objetivos a la vez. Por lo que la optimización multi-objetivo no trata de encontrar una solución óptima sino que busca un equilibrio entre todas las soluciones. Además de tener varios propósitos, la principal diferencia es que la optimización multi-objetivo tiene dos intenciones. La primera es encontrar un conjunto de soluciones lo más cercano posible a la Frontera Óptima de Pareto. La segunda intención es encontrar un conjunto de soluciones lo más diverso (Garen, 2002).

En los problemas de optimización combinatoria, el objetivo es obtener el arreglo óptimo de un grupo de entidades discretas de tal manera que los requisitos y limitaciones adicionales se han satisfecho. Si el problema de optimización combinatoria es multi-objetivo, significa que existen varios criterios para evaluar la calidad de la solución dada y hay un objetivo (minimización o maximización) unido a cada uno de estos criterios. Por lo general, se cumple que algunos de los criterios entran en conflicto, es decir, una mejora en uno de ellos sólo puede lograrse a expensas del empeoramiento del otro. Por otra parte, para algunos de los

criterios puede pasar que las unidades utilizadas para medir el cumplimiento de cada uno de ellos no sean comparables en absoluto.

1.3.1 Frontera de Pareto

La primera decisión que debe ser tomada cuando tratamos con un problema de optimización multi-objetivo es precisamente cómo combinar la búsqueda y los procesos de toma de decisiones. Es aquí donde aparece la Optimalidad de Pareto.

En este enfoque, un vector que contiene todos los valores objetivos representa la aptitud de la solución y el concepto de dominancia se utiliza para establecer la preferencia entre las soluciones. Una solución x se dice que es no dominada si no hay otra solución que sea mejor en todos los criterios.

Aquí se considera principalmente el problema de minimización porque la mayoría de los problemas de secuenciación son de este tipo, pero la definición anterior cambia de la forma obvia para el caso de problemas de maximización. Es importante tener en cuenta que el tipo de dominancia puede tener un efecto sobre cómo se realiza la búsqueda. Esto es porque si una solución está estrictamente dominada significa que se superó por la otra solución en todos los criterios; mientras que si la solución está dominada vagamente significa que se superó en algunos de los criterios, pero es tan buena como la otra solución en al menos uno de ellos. Entonces, la búsqueda de una nueva solución que domina estrictamente a la actual puede ser más difícil que encontrar una solución que la domine vagamente (Silva and Burke, 2004).

El Frente Óptimo de Pareto es el conjunto de todas las soluciones no dominadas en el espacio multi-objetivo. Sin embargo, cuando las soluciones en el conjunto obtenido no se encuentran en el Frente Óptimo de Pareto, uno debe referirse a ese conjunto como el frente no dominado obtenido o el Frente de Pareto conocido.

La Optimalidad de Pareto se refiere a la búsqueda de este frente o un conjunto que representa una buena aproximación a él y usualmente los métodos de optimización multi-objetivo se refieren a técnicas para la Optimalidad de Pareto.

El atractivo de la Optimalidad de Pareto viene del hecho de que en la mayoría de los problemas de optimización multi-objetivo no hay tal cosa como una única mejor

solución u óptimos globales y también es muy difícil establecer preferencias entre los criterios antes de la búsqueda. Aun cuando esto es posible, puede que estas preferencias cambien.

Dado que en la Optimalidad de Pareto el resultado final debe ser un conjunto de soluciones no dominadas, otro aspecto importante a considerar es la forma de evaluar la calidad del frente no dominado obtenido. Existen varios criterios para evaluar que tan bueno fue el frente obtenido entre los cuales se encuentran:

- El número de soluciones no dominadas obtenidas.
- La cercanía entre el frente obtenido y el Frente Óptimo de Pareto (si se conoce).
- El cubrimiento del frente, es decir, la difusión y distribución de las soluciones no dominadas.(Silva and Burke, 2004)

1.3.2 Métodos para resolver problemas multi-objetivo

La optimización multi-objetivo es un área de investigación muy interesante, no solo porque es una realidad la naturaleza multi-objetivo de muchos problemas sino porque aún queda mucho campo por explorar en este tema. Aun así, varios han sido los métodos usados hasta el momento para resolver problemas multi-objetivo.

Uno de los paradigmas más usados ha sido el de los algoritmos evolutivos, principalmente los algoritmos genéticos, de estos se pueden citar varios ejemplos que han mostrado soluciones satisfactorias como son el *Vector Evaluated Genetic Algorithm* (VEGA) (Schaffer, 1985), el *Multi-objective Genetic Algorithm* (MOGA) (Fonseca and Fleming, 1993), el *Niche Pareto Genetic Algorithm* (NPGA) (Horn et al., 1994, Zitzler and Thiele, 1999), el *Pareto-Archived Evolutionary Strategy* (PAES) (Knowles and Corne, 2000a) y *Pareto converging genetic algorithm* (PCGA) (Kumar and Rockett, 2002).

También se encuentran algunas meta-heurísticas enfocadas a resolver problemas multi-objetivo como son *Multi-objective Tabu Search* (MOTS) (Hansen, 1997), *Pareto Simulated Annealing* (PSA) (Czyżak and Jaszkiewicz, 1997), *Memetic Pareto Archived Evolutionary Strategy* (M-PAES) (Knowles and Corne, 2000b), entre otros (Silva and Burke, 2004).

Generalmente, en los algoritmos evolutivos para resolver problemas multi-objetivo se usan operadores genéticos estándares y la diferencia entre estos algoritmos se concentra en la estrategia usada para la selección, la mutación o el cruzamiento.

Casi todos los algoritmos evolutivos para resolver problemas multi-objetivo usan la dominancia de Pareto para la selección de los individuos durante la búsqueda mientras que muchas de las alternativas meta-heurísticas para estos problemas (las que usan búsquedas locales) emplean un método diferente, que en la mayoría de los casos es una función de agregación ponderada.

En los problemas de secuenciación de tareas pueden considerarse varios objetivos como son *makespan*, *lateness*, *tardinnes*, *earliness*, *flowtime*, etc. La mayoría de las aplicaciones meta-heurísticas en problemas de secuenciación de tareas multi-objetivo que han sido reportadas, han considerado dos o tres objetivos y muchas de ellas se han concentrado en los problemas de secuenciación tipo *Flow Shop*. Dentro de los algoritmos más conocidos se encuentran la meta-heurística basada en Búsqueda Local (Knowles, 2002), los Algoritmos Genéticos Multi-objetivo (Coello, 2006) y extensiones para Algoritmo Genético Multi-objetivo (Coello, 2006).

Sin embargo, pocos enfoques (Horn et al., 1994, Mariano and Morales, 1999a, Mariano and Morales, 2000, Mariano and Morales, 1999b) dan solución al problemas de secuenciación de tareas usando RL para los problemas de tipo JSSP, lo que refleja cuánto hay aún por investigar en este tema.

1.4 Sistemas Multi-Agente

Un Sistema Multi-Agente es un sistema en el cual varios agentes actúan en el mismo ambiente para cumplir con una tarea específica. El creciente interés en el uso de este tipo de sistemas en problemas del mundo real viene dado por la habilidad de resolver problemas muy grandes por un agente centralizado, y también por la habilidad de obtener soluciones donde la experiencia es distribuida, como por ejemplo, problemas de secuenciación en ambientes de manufactura(Sycara, 1998). El campo de los sistemas multi-agente se centra en procesos descentralizados (sistemas distribuidos), ya que cada agente del sistema tiene su propia percepción (pueden estar en ubicaciones distintas y ser responsables de diferentes partes del sistema), control (diferente experiencia) y forma de actuar (diferentes acciones potenciales) (Kaminka, 2004). Todas estas características se resumen en la siguiente definición:

Según (Jennings et al., 1998) un Sistema Multi-Agente 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.

Los Sistemas Multi-Agente se caracterizan porque cada agente tiene un punto de vista limitado, no existe un control global del sistema, los datos pueden ser descentralizados y el cálculo puede ser asíncrono (Martínez, 2012).

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 pueden aprender independientemente o en conjunto. Cuando aprenden en conjunto se asume que las acciones que toman los otros agentes pueden ser observadas. Si se aprende independientemente no se necesita observar las acciones tomadas por los otros agentes.

1.5 Aprendizaje Reforzado

El RL, como se menciona en (Kaelbling et al., 1996), se remonta a los momentos iniciales de la cibernética y los trabajos en estadística, psicología, neurociencia y ciencias de la computación. Durante las últimas décadas también ha atraído el interés de la comunidad de Inteligencia Artificial.

RL es una variante computacional para automatizar el aprendizaje y la toma de decisiones guiados por objetivo. Se distingue de otras alternativas computacionales por su énfasis en que el agente aprenda de la interacción directa con su ambiente, sin depender de una supervisión o de un modelo completo del ambiente.

RL es aprender qué hacer (cómo mapear situaciones con acciones) de forma tal que se maximice una señal numérica de recompensa. Al aprendiz no se le dice qué acciones tomar, sino que debe descubrir qué acciones arrojan la mayor recompensa seleccionándolas. En los casos más interesantes las acciones no solo afectan la recompensa inmediata sino también las situaciones siguientes y por tanto las recompensas futuras. Estas dos características, búsqueda a ‘prueba y error’ y recompensa retardada son de las más importantes que distinguen el RL (Barto, 1998).

En el modelo estándar del RL, un agente está conectado con su ambiente vía percepción y acción, como muestra la **Figura 1.2**, en cada interacción el agente percibe el estado actual s del ambiente y selecciona una acción a para cambiar dicho estado. Esta transición genera una señal r que es recibida por el agente, cuya tarea es aprender una política de selección de

acciones en cada estado para recibir la mayor recompensa acumulada a largo plazo (Zhang, 1996).

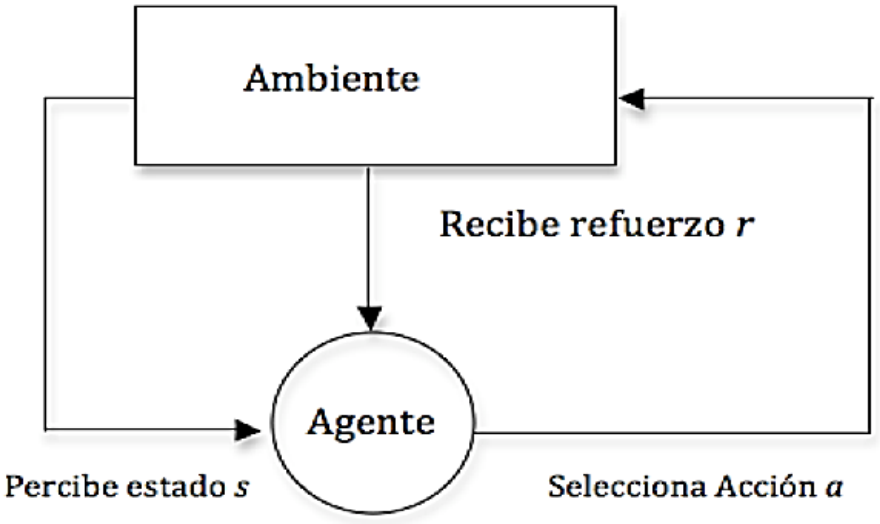


Figura 1.2. Modelo estándar de RL.

Uno de los retos que se plantean en el RL y no en otros tipos de aprendizaje es el equilibrio entre la exploración y la explotación. Para obtener una alta recompensa, un agente con RL 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 la explotación pueden ser perseguidas exclusivamente sin fallar en la tarea. El agente debe tratar una variedad de acciones y progresivamente favorecer aquellas que parecen ser mejores. En una tarea estocástica, cada acción debe ser probada muchas veces para obtener una apreciación fiable de la 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).

De manera formal, el modelo básico de RL 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

Aparte del agente y del entorno, se pueden identificar subelementos de un sistema de RL: la política, la función de recompensa, la función de valor y opcionalmente el modelo del entorno.

En cada instante *t*, el agente percibe su estado $st \in S$ y el conjunto de posibles acciones $A(st)$. Se elige una acción $a \in A(st)$ y recibe del entorno el nuevo estado $st + 1$ y una recompensa $rt + 1$, esto significa que el agente implementa un mapeo de los estados a las probabilidades de selección de cada acción posible.

Esta asignación se denomina política del agente y se denota πt, donde $\pi t(s, a)$ es la probabilidad de que $at = a$ si $st = s$, es decir, es la probabilidad de seleccionar una acción a en el estado s en el tiempo t.

La función de recompensa define la meta en un problema de RL. En términos generales, se asigna a cada par estado-acción del entorno, una recompensa, lo que indica la conveniencia intrínseca de ese estado. El objetivo de un agente de RL es maximizar la recompensa total de lo que recibe a largo plazo. La función de recompensa define qué acontecimientos son buenos y cuales malos para el agente (Barto, 1998).

De la misma forma en que la función de recompensa indica qué es “inmediatamente” bueno, la función de valor especifica qué 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 (Barto, 1998).

Además de RL, los agentes inteligentes pueden ser diseñados por otros paradigmas, en particular la planificación y aprendizaje supervisado, pero existen algunas diferencias entre estos enfoques. Dado que los algoritmos de planificación operan usando un modelo del medio, ellos pueden dar marcha atrás o "deshacer" las transiciones de estado que entran en estados no deseados. Por el contrario, RL está destinado a aplicarse a las situaciones en las que no existe un modelo de acción lo suficientemente manejable. En consecuencia, un agente en el paradigma de RL debe explorar activamente su entorno para observar los efectos de sus

acciones. A diferencia de la planificación, los agentes RL normalmente no pueden deshacer las transiciones de estado. Por supuesto, en algunos casos, puede ser posible construir un modelo de acción a través de la experiencia, lo que permite una mayor planificación usando la experiencia acumulada.

Los agentes también pueden ser entrenados a través del aprendizaje supervisado. El objetivo en el aprendizaje supervisado es inducir una política general de los ejemplos de entrenamiento. Por otra parte, RL no requiere un conocimiento previo de las decisiones correctas e incorrectas. RL puede ser aplicado a situaciones en que las recompensas son escasas, por ejemplo, las recompensas se pueden asociar sólo con ciertos estados. En tales casos, puede ser imposible asociar una etiqueta de correcta o incorrecta sobre una decisión particular sin hacer referencia a las decisiones posteriores del agente, haciendo el aprendizaje supervisado inviable (Martínez, 2012).

En resumen, RL proporciona un enfoque flexible para el diseño de agentes inteligentes en situaciones en las cuales la planificación y el aprendizaje supervisado no son prácticos. RL puede ser aplicado a problemas para los que el conocimiento significativo del dominio no está disponible o es costoso de obtener.

1.5.1 RL para problemas de secuenciación de tareas

Uno de los trabajos existentes sobre la aplicación de enfoques basados en RL para resolver problemas de secuenciación fue presentado en (Zhang, 1995) y extendido un año después (Zhang, 1996). Este trabajo se centró en aplicaciones para la NASA, donde el problema consistía en planificar varias tareas que debían ser ejecutadas para instalar y probar las cargas que son ubicadas en el espacio de carga de los transbordadores. En dicho estudio cada misión se consideraba un trabajo, el cual tiene una fecha de vencimiento y está compuesto por un conjunto de tareas parcialmente ordenadas con restricciones de recursos. El objetivo es planificar dicho conjunto de tareas para satisfacer un conjunto de restricciones temporales y a la misma vez minimizar el tiempo total de duración de la ejecución.

En (Gabel and Riedmiller, 2007) y (Gabel, 2009), los autores sugieren y analizan la aplicación de RL para resolver problemas de secuenciación de tipo *Job Shop*. En estos trabajos se demuestra que interpretar y resolver este tipo de escenarios a través de sistemas multi-agentes

y RL es beneficioso para obtener soluciones cercanas a las óptimas y puede muy bien competir con enfoques de solución alternativos.

El ambiente del problema de secuenciación define el número de agentes y la relación entre ellos. Los agentes pueden no conocer el estado global del sistema y para obtener un mejor desempeño del mismo se comunican entre ellos para determinar sus acciones basadas en información limitada. Claramente, el enfoque centralizado es aplicable a problemas en los cuales la información global está disponible y los agentes son cooperativos.

Los problemas en los que algunos agentes mantienen su información en privado por razones competitivas o de otra índole se asocian a métodos distribuidos que van desde la coordinación entre agentes cooperativos hasta la negociación entre agentes competitivos.

Las principales características que deben ser consideradas cuando se modela un problema son adoptadas de (Gabel, 2009) y se pueden resumir de la siguiente manera:

Conjunto de estados factorizados: Se asume que cada recurso tiene un agente i asociado que observa el estado local en su recurso y controla su comportamiento. Consecuentemente, existen tantos agentes como recursos existan en el problema ($|A_g| = |R| = m$).

Observabilidad local y completa: El estado local s_i del agente i, y por tanto la situación del recurso r_i , es completamente observable. Adicionalmente, la composición de todos los recursos determina el estado global del problema de secuenciación.

Acciones factorizadas: Las acciones corresponden al inicio de las operaciones de los trabajos (expedición de los trabajos). Por tanto, una acción local del agente i refleja la decisión de procesar un trabajo particular (más específicamente, la próxima operación de ese trabajo) entre un conjunto A_i de operaciones actualmente esperando en el recurso r_i .

Conjuntos de acciones cambiantes: Si las acciones denotan la expedición de las operaciones que esperan por ser procesadas, entonces el conjunto de acciones disponibles para un agente varía en el tiempo, ya que el conjunto de operaciones que esperan por un recurso cambia a medida que dichas operaciones se van ejecutando. A_i^r corresponde al conjunto de operaciones $o_{j,k}$ que debe ser procesado en el recurso r_i , es decir $\varrho(o_{j,k})$, donde j es el trabajo al que pertenece la operación y k es el identificador de la operación. Además, el estado local s_i del

agente i está completamente descrito por el conjunto cambiante de acciones esperando por ser procesadas en el recurso r_i , por tanto, $s_i = A_i$.

1.5.2 Q-*Learning*

Q-Learning (Q-L), es una técnica propuesta por Watkins en 1989 (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 *a* en un estado *s* determinado y a partir de ahí seguir una política óptima.

Este algoritmo es muy conocido dentro del RL Se basa en aprender una función *acción-valor* que brinda la utilidad esperada de tomar una acción determinada en un estado específico. El centro del algoritmo es una simple actualización de valores, cada par (s; a) tiene un Q-valor asociado, cuando la acción a es seleccionada mientras el agente está en el estado *s*, el Q-valor para ese par estado-acción se actualiza basado en la recompensa recibida por el agente al tomar la acción. También se tiene en cuenta el mejor Q-valor para el próximo estado s' la regla de actualización completa es la siguiente:

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

Q-L tiene la ventaja de que se ha demostrado que converge a la política óptima. Sin embargo, esto sólo es cierto para los *Markov Decision Process (MDPs).*

El algoritmo puede ser planteado como se muestra en la **Figura 1.3**:

Inicializar $Q(s, a)$ arbitrariamente

Repetir (por cada episodio)

Inicializar s

Repetir (para cada paso del episodio)

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

Tomar acción a, recompensa r, estado futuro s'

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

$s \leftarrow s'$;

Hasta que s sea terminal

Figura 1.3 Algoritmo Q-*Learning*

El algoritmo Q-L 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.

1.6 Conclusiones Parciales

Después de haber hecho una revisión bibliográfica de los principales conceptos y definiciones que se necesitaban para comprender el problema planteado se pueden arribar a las siguientes conclusiones parciales:

- Los problemas de secuenciación tipo *Job Shop* son muy complejos de implementar y para hacerlo comúnmente se tiene en cuenta la optimización de más de un objetivo, aunque la mayoría de la literatura solo optimiza el tiempo total de terminación de todas las tareas.

- De acuerdo a la literatura los objetivos más importantes a optimizar en este tipo de problemas son el tiempo de terminación de todas las tareas, la suma del tiempo de terminación de las tareas y la máxima tardanza.
- Para dar solución a problemas de tipo multi-objetivo no existe una solución única, sino un conjunto de soluciones, denominado Frontera de Pareto.
- El RL resulta un enfoque natural para resolver problemas de secuenciación de tareas tipo Job Shop. Esta técnica se ha aplicado con éxito a problemas de este tipo con enfoques mono-objetivo.

CAPÍTULO 2. ALGORTIMO MULTI-OBJETIVO APLICADO AL PROBLEMA JSSP BASADO EN APRENDIZAJE REFORZADO

En este capítulo se representan las instancias que se van a utilizar para validar el algoritmo. Se describe cómo están distribuidos los agentes en el algoritmo Q-L para los JSSP. Se explican las diferentes funciones heurísticas utilizadas para dar la recompensa y que los agentes fueran capaces de aprender cuáles son las mejores soluciones, además se explica cómo se obtienen las soluciones no dominadas para construir la Frontera de Pareto.

En capítulo además se propone un algoritmo multi-objetivo basado en RL y usando la Optimalidad de Pareto: el MOQL. Este algoritmo dará paso a un frente de soluciones no dominadas cercano al Frente Óptimo de parteo.

2.1 Instancias de los problemas JSSP

Para medir la efectividad del MOQL se usarán algunas de las instancias que aparecen disponibles en OR-Library (Beasley, 1990), la cual es una biblioteca de investigación de operaciones disponible en Internet, las instancias que usaremos son específicamente para los problemas de tipo *Job Shop* y tienen el formato que se describe a continuación.

Cada instancia tiene una primera línea que contiene el número de trabajos y el número de máquinas y luego una línea para cada trabajo que lista el número de la máquina y el tiempo de procesamiento para cada paso del trabajo. Máquinas y trabajos comienzan a enumerarse desde cero. En la **Figura 2.1** se observa el ejemplo de la instancia ft06 que es una de las más estudiadas en la literatura.

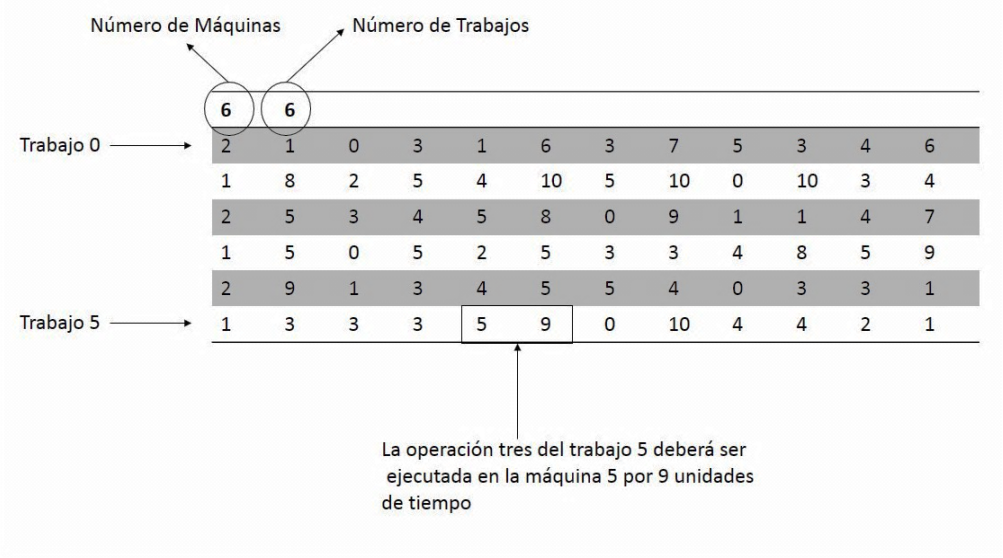


Figura 2.1 Forma de la instancia ft06

En el ejemplo mostrado en la **Figura 2.1** se tienen seis trabajos y seis máquinas, por ejemplo, el trabajo 5 debe ser procesado por la máquina 1 durante tres unidades de tiempo, luego debe ir a la máquina 3 por tres unidades de tiempo y así hasta llegar a la máquina 2, después de esta terminar de procesar la última operación demorándose una unidad de tiempo se dará por concluido el trabajo. Es importante resaltar que el orden de procesamiento descrito en la instancia para cada trabajo no puede ser violado, el orden de procesamiento del trabajo en cuestión será por ejemplo $\{M_1, M_3, M_5, M_0, M_4, M_2\}$ por $\{3, 3, 9, 10, 4, 1\}$ unidades de tiempo respectivamente.

La descripción general de un trabajo se muestra en la **Figura 2.12:**

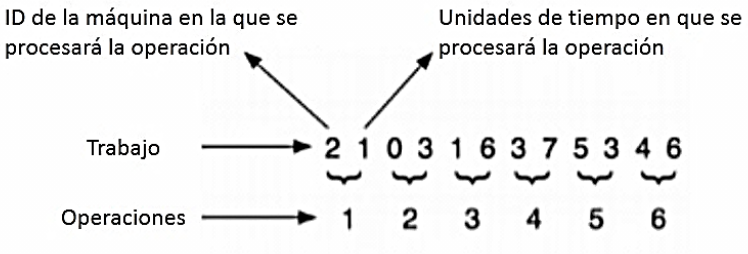


Figura 2.2 Descripción de un Trabajo

La instancia ft06 mostrada en la **Figura 2.3** es una de las más estudiadas, para ella el óptimo reportado es 55. Una solución posible tomada de (Gabel and Riedmiller, 2007) se muestra a continuación:

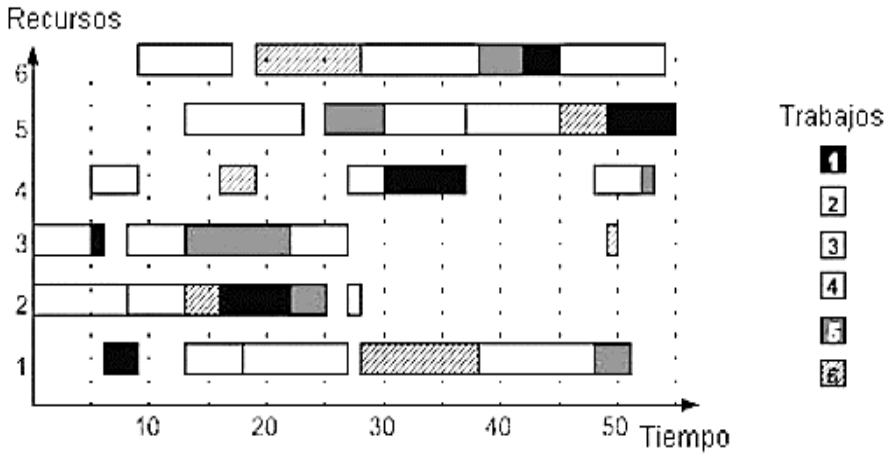


Figura 2.3 Diagrama de Gantt de una solución óptima para la instancia ft06

2.2 Aprendizaje Reforzado

En el paradigma RL, un agente se conecta a su ambiente mediante la percepción y la acción.

El RL provee mecanismos que permiten generar transiciones utilizando un modelo 'débil' sin la necesidad de una descripción analítica completa basada en las probabilidades de transición y permite además la actualización mientras que se realiza el proceso de aprendizaje. (Martínez, 2012)

En los JSSP en cada tiempo t, el agente percibe su estado $s_t \in S$ y el conjunto de acciones posibles $A(s_t)$. Selecciona una acción $a \in A(s_t)$ y recibe del ambiente el nuevo estado s_{t+1} y la recompensa r_{t+1}. El agente selecciona la próxima operación a realizar basado en la probabilidad que tiene de ser seleccionada, lo cual es llamado **política del agente** y es denotado por π, donde $\pi(s,a)$ es la probabilidad de que $a_t = a$ si $s_t = s$, es decir, es la probabilidad de seleccionar la acción a en el estado s en el tiempo t como se muestra en la **Figura 2.14.**

La **función de recompensa** define la meta en un problema de RL. Esta función asigna a cada estado percibido (o cada par estado-acción) del ambiente un número (la recompensa), indicando la deseabilidad de ese estado. El objetivo de un agente en el RL es maximizar la

suma del total de recompensas que él recibe durante la ejecución. La función de recompensa define cuán buenos o malos son los eventos para el agente.

El valor de un estado es la cantidad total de premios que el agente espera acumular en el futuro, a partir del estado dado. La **función de valor (estado)** especifica lo que es bueno para el agente a largo plazo.

La **función de valor (estado-acción)** retorna el valor esperado de las recompensas acumuladas en el futuro, a partir del estado y la acción dados. La estimación de los valores de estas funciones es el núcleo de la actividad en el RL.

En resumen, los métodos del RL brindan la posibilidad de diseñar agentes capaces de resolver problemas en los que el dominio del conocimiento no está disponible o es costoso de obtener.

2.2.1 Q-*Learning*

Como anteriormente se dijo; Q-L es una técnica de RL que trabaja con una función acción-valor, la cual brinda la utilidad esperada de tomar una acción determinada en un estado dado y seguir una política óptima después de esto. Una ventaja de Q-L es que puede comparar la utilidad esperada de las acciones disponibles sin requerir un modelo del ambiente. El centro del algoritmo es la actualización de un valor simple en una iteración, cada par (s, a) tiene un Q-valor asociado (*s* es un estado del conjunto de estados S, y *a* es una acción del conjunto de acciones A). Cuando la acción *a* es seleccionada por el agente que se encuentra en el estado *s* el Q-valor para ese par *estado-acción* es actualizado en base a la recompensa recibida cuando se seleccionó esa acción, y el mejor Q-valor para el subsecuente estado *s*. Vale recordar que la regla de actualización para cada par estado-acción es la siguiente:

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

En esta expresión $\propto \in]\,0,1]$ es la velocidad de aprendizaje y r la recompensa o penalidad resultante de tomar la acción *a* en el estado *s*. La velocidad de aprendizaje determina el grado por el cual el valor anterior es actualizado. Por ejemplo, si la velocidad de aprendizaje es α= 0, entonces nada será actualizado; por otro lado, si $\alpha = 1$ el valor anterior es remplazado por el nuevo estimado. Usualmente se utiliza un valor pequeño para la proporción de aprendizaje, por ejemplo, α= 0.1. El factor de descuento (parámetro γ) tiene el rango de valores desde 0 hasta 1 ($0 \leq \gamma < 1$). Si γ es cercano a cero el agente tiende a considerar solo la recompensa

inmediata. Si γ es cercano a uno el agente considerará la recompensa futura en mayor medida. Q-L tiene la ventaja de que se puede demostrar que siempre converge a una política óptima.

El Q-L es usado por el agente para aprender de la experiencia o entrenamiento. Cada episodio es equivalente a una sesión de entrenamiento. En cada sesión de entrenamiento el agente explora el ambiente y obtiene la recompensa cuando alcanza el estado final o la meta. El propósito de este entrenamiento es reforzar la memoria del agente representado por la matriz de los Q-valores. Un mayor entrenamiento dará como resultado mejores valores que podrán ser utilizados por el agente para moverse de una forma más óptima. Como fue mencionado anteriormente, los agentes necesitan un balance entre explotación y exploración. El método de selección *ε-greedy*, instruye al agente a seguir la política π la mayoría del tiempo, pero en ocasiones, el agente elige una acción de forma aleatoria (con igual probabilidad para cada posible acción en el estado actual). La probabilidad ε determina cuando usar una acción aleatoria, esto provee equilibrio entre explotación y exploración.

2.2.2 Aplicación del Q-L en el JSSP

Para introducir el Q-L al problema de secuenciación JSSP en cuestión, existen elementos importantes que deben ser definidos y que pueden ser descritos como se muestra a continuación.

Estrategia de selección de la acción: Para seleccionar las acciones la estrategia que se usa es *ε-greedy*, pues el uso de la misma proporciona una mayor exploración del espacio de soluciones.

Q-Valores: Se retoma la instancia ft06 descrita en la **Figura 2.2** como ejemplo, en este caso hay seis agentes (uno por recurso) y seis trabajos involucrados, de acuerdo a las restricciones de los JSSP, cada agente ejecutara seis acciones. De acuerdo a (Gabel, 2009) el conjunto de estados para el agente *i* se denota como $S_i = P(A_i^r)$, esto dará un total de $|S_i| = 2^6 = 64$ estados locales para cada agente *i*, por lo que la cantidad de posibles estados del sistema está acotada superiormente por $|S| \leq (2^6)^6 = 2^{36}$. Debido a las restricciones de orden del problema, muchos de estos estados puede que nunca sean alcanzados. Por ello el MOQL solo almacena los estados que pueden ser alcanzados, es decir, la combinación de operaciones que pueden estar a la misma vez en las colas del sistema. Estas combinaciones se van almacenando a medida que aparecen. Por ejemplo, si el algoritmo es ejecutado por solo una iteración,

entonces solo se guardarán seis estados, que son los estados en los que el agente se encontraba cuando fueron seleccionadas las acciones. Otras ejecuciones pueden conllevar a la creación de otros estados.

Recompensa: La meta de este trabajo es lograr optimizar varios objetivos a la vez, el *makespan* (C_{max}) que es la longitud de la planificación, el *flow time* que es la suma de los tiempos que demoraron en procesarse todos y cada uno de los trabajos y el *tardiness* que es la mayor de las tardanzas de los trabajos. La principal diferencia que se tuvo en cuenta entre estos objetivos a la hora de implementar el algoritmo Q-L fue específicamente las funciones de recompensa.

En el caso del *makespan*:

Para lograr alcanzar el *makespan* óptimo se deben tener tan pocos recursos inactivos como sea posible, esto hará que las operaciones en cola en las máquinas disminuyan y por tanto el tiempo de finalización del sistema disminuirá. Para lograr esto el MOQL utiliza dos actualizaciones locales y una global.

Las actualizaciones locales se basan en los tiempos de procesamientos de las operaciones y dan un estimado del beneficio de seleccionar una acción específica. En el primer caso la recompensa es *1/tiempo de procesamiento* de la operación (nótese que a mayor tiempo de procesamiento menor recompensa se le dará a esa acción), lo que dará mayor prioridad en el sistema a las operaciones que más rápido se procesan. Cada vez que una operación es seleccionada, se actualiza el estado local de la máquina en que se ejecutó dando *1/tiempo de procesamiento* de la operación como recompensa. En el segundo caso se utiliza como recompensa la suma total de los tiempos de ejecución de las operaciones en cola de la máquina menos el tiempo de procesamiento de la operación seleccionada y en este caso, también a mayor tiempo de procesamiento menor recompensa se le dará a esa acción.

Por otra parte, la actualización global se basa en dar una recompensa a partir del resultado final de un episodio, por lo que se modificará el Q-valor (calculado a partir de las actualizaciones locales explicadas anteriormente) de las acciones tomadas en este, teniendo en cuenta el beneficio real al que ellas conllevaron. Nótese que el hecho de que una acción en particular obtenga una buena señal de recompensa no es definitivo, ya que al finalizar el episodio el conjunto de acciones tomadas por los agentes pueden haber resultado en una buena

o mala secuencia, por tanto, se van a estimular o penalizar respectivamente las acciones involucradas. La forma de dar esta recompensa se define como recompensa $1/makespan$, obsérvese 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.

En el caso del *flow time*:

La actualización local en este caso se basa al igual que en el *makespan* en los tiempos de procesamientos de las operaciones y dan un estimado del beneficio de seleccionar una acción específica. La recompensa será *1/tiempo de procesamiento* de la operación (a mayor tiempo de procesamiento menor recompensa se le dará a esa acción), lo que dará mayor prioridad en el sistema a las operaciones que más rápido se procesan. Cada vez que una operación es seleccionada se actualiza el estado local de la máquina en que se ejecutó dando *1/tiempo de procesamiento* de la operación como recompensa.

Después de obtenido el *flow time* de un episodio dado, se hace una actualización a todos los estados por los que el sistema pasó durante este proceso, estos estados son almacenados en una lista y la recompensa dada a cada uno de ellos será *1/flow time* la cual da una buena medida y mejora las soluciones obtenidas ya que es proporcional a la calidad de la solución encontrada.

En el caso del *maximum tardiness*:

La actualización local en este caso se basa en las fechas de inicio y entrega del trabajo. Si el trabajo seleccionado j_i tiene una tardanza $max\{0, L_j\} > 0$ siendo $L_j = tiempo\_terminacion\_j_i - due\ date j_i$, la recompensa obtenida es ($max\{0, L_j\}$), de forma que mientras más alto sea la tardanza del trabajo mayor prioridad se le da a este para que termine de procesarse lo más rápido posible. En el caso de que el trabajo está en tiempo, la recompensa será $1/max\{0, |L_j|\}$; es decir, el adelanto del trabajo; a mayor adelanto del trabajo menor recompensa obtendrá, dándole más prioridad a los trabajos retrasados. Esta recompensa fue usada en una actualización local del estado actual (cada vez que una operación es seleccionada se actualiza el estado local de la máquina en que se ejecutó).

Después de obtenido el *tardiness* de un episodio dado se hace una actualización a todos los estados por los que el sistema pasó durante este proceso, estos estados son almacenados en una

lista y la recompensa dada a cada uno de ellos es *1/tardiness* la cual da una buena medida y mejora las soluciones obtenidas ya que es proporcional a la calidad de la solución encontrada.

2.3 Configuración de los Agentes Inteligentes

Comenzaremos definiendo los principales conceptos a tener en cuenta a la hora de resolver un problema de secuenciación de tareas usando RL aplicado al MOQL:

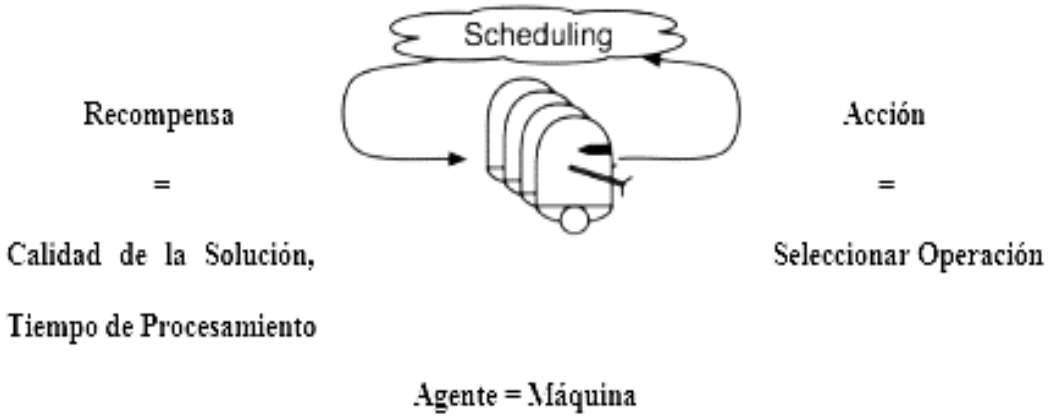


Figura 2.4 Esquema general del problema de secuenciación JSSP usando RL

Cuando se aplica RL para resolver el JSSP, un agente es asociado a cada uno de los *m* recursos (máquinas). Este agente decide localmente las acciones elementales.

En la implementación que se realiza del MOQL el primer paso es obtener los datos necesarios de la instancia a resolver. Una vez almacenada la información, los agentes asociados a los diferentes recursos comienzan a seleccionar sus próximas acciones. Un agente no puede tomar una decisión en cada instante de tiempo sino cuando el recurso al que está asociado ha terminado una operación, ya que cada recurso solo puede procesar un trabajo a la vez, y además un trabajo solo puede estar procesándose en una máquina en un instante de tiempo determinado.

Las instancias del problema, obtenidas de una librería pública en (Beasley, 1990), brindan la secuencia de recursos que deben procesar un trabajo y el tiempo asociado a cada procesamiento como se explicó anteriormente. Cuando un trabajo termina de ser procesado por un recurso se sabe cuál es el próximo al que debe ser enviado, lo que se debe decidir es cuándo ese trabajo va a ser procesado por la próxima máquina. Quizás una operación que llegue de última a un recurso debe ser seleccionada para procesarse primero. En cada paso,

para seleccionar una acción, cada agente toma en cuenta sólo las operaciones que pueden ser procesados en ese momento de acuerdo a las restricciones del problema. Básicamente se mantienen dos tipos de conjuntos, uno que contiene los recursos que están disponibles para procesar algún trabajo (agentes habilitados para tomar una acción) y otro que contiene las operaciones que cada agente puede seleccionar como su próxima acción. Ambos conjuntos son continuamente actualizados ya que una vez que un recurso finaliza el procesamiento de un trabajo cambia el estado del sistema. Para el uso del algoritmo Q-L 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. El ambiente del problema de secuenciación define el número de agentes y la relación entre ellos.

Estados y acciones: Existe un agente asociado a cada recurso, y este agente tomará decisiones sobre futuras acciones. Para un agente tomar un acción significa que debe decidir cuál va a ser la próxima operación a procesar del conjunto de operaciones posibles, estas son los que están esperando en el correspondiente recurso como en la **Figura 2.15**. Cada agente tiene una visión local, solo tiene acceso a la información asociada a su recurso, y los trabajos que están esperando por él. Es decir, para un agente el estado del sistema será el conjunto de operaciones en cola en el recurso al que este fue asignado, mientras que una acción será seleccionar la próxima operación a ejecutar dentro de las que son posibles en ese estado.

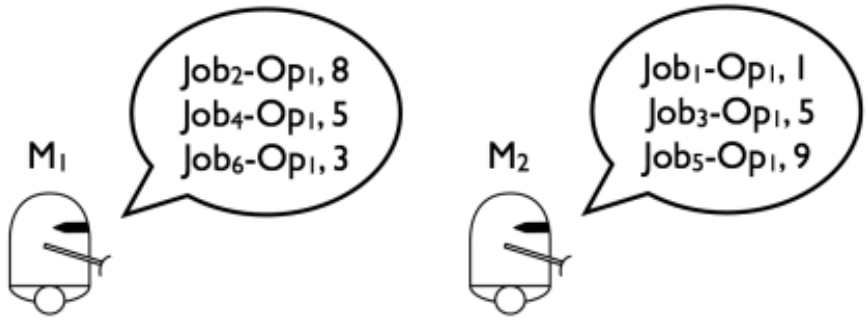


Figura 2.5 Conjunto de acciones posibles a ejecutar por un agente en un estado

Estrategia de selección

Uno de los retos que se plantean en el RL y no en otros tipos de aprendizaje es el equilibrio entre la exploración y la explotación. 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 expondremos brevemente dos estrategias de selección muy comunes, *greedy*, ε-*greedy*.

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

ε-*Greedy*: Es una alternativa al comportamiento avaricioso de la estrategia *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 RL, 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).

2.4 Frontera de Pareto

El conjunto de soluciones de Pareto es llamado la Frontera de Pareto. Por lo tanto resolver un problema multi-objetivo de secuenciación constituye un problema de optimización de Pareto. Generar el conjunto óptimo de Pareto para un problema de secuenciación puede ser costoso computacionalmente, y es a menudo inalcanzable, debido a la complejidad de los problemas de secuenciación (Ishibuchi et al., 2003). A menudo cuando se usan meta-heurísticas no existe garantía de que el conjunto de Pareto para un problema dado de optimización multi-objetivo como son los problemas de secuenciación multi-objetivo pueda ser generado. Sin embargo, se puede generar un conjunto de soluciones no dominadas cercano al conjunto óptimo de Pareto (Bagchi, 1999, Ishibuchi et al., 2003).

Uno de los conceptos más importantes a tener en cuenta cuando se habla de optimización de Pareto, es el de dominancia de Pareto.

Dominancia de Pareto: En este enfoque, un vector que contiene todos los valores objetivos representa la aptitud de la solución y el concepto de dominancia se utiliza para establecer la

preferencia entre las soluciones. Una solución *V* se dice que es no dominada si no hay otra solución que sea mejor en todos los criterios. Formalmente, la relación de dominancia enfocada a la implementación del MOQL se describe de la siguiente forma:

Supóngase que dos vectores distintos $V = (v_1, v_2, v_3)$ y $U = (u_1, u_2, u_3)$ contienen los valores objetivos de dos soluciones para este problema, luego:

- V domina estrictamente a U si $v_i < u_i$ para $i = 1,2,3$.
- V domina vagamente a U si $v_i \leq u_i$ para $i = 1,2,3$ y $v_i \leq u_i$, para al menos una i.
- V y U son incomparables si V no domina (estrictamente o vagamente) a U ni U domina (estrictamente o vagamente) a V.

En el enfoque propuesto en esta tesis, para obtener el Frente de Pareto, por episodio para la secuencia encontrada se calculan los tres objetivos; el *makespan*, el *flow time* y el *tardiness* formando un vector de tres dimensiones que se tratará de incluir en el Frente de Pareto.

Para incluir el vector en dicho frente se tendrá en cuenta el concepto de dominancia explicado anteriormente. Si este vector no es dominado por ninguno de los que conformen el Frente en ese momento, se incluirá la posible solución y se eliminarán del Frente de Pareto todas aquellas soluciones que sean dominadas por la nueva solución. Así sucesivamente hasta obtener el Frente de Pareto que represente la solución al problema multi-objetivo en cuestión.

La inclusión de una solución en el frente se describe en el siguiente algoritmo:

Repetir (para cada solución *i* en el frente de Pareto):

Si *i* domina estricta o vagamente a *solución actual*

Salir del episodio

Si *solución actual* domina a *i*

Eliminar *i* del Frente de Pareto

Adicionar *i* al Frente de Pareto

2.5 Propuesta de solución

El MOQL puede resumirse de la siguiente forma:

Para cada objetivo:

Inicializar:

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

Para cada episodio:

$s = \{\} \quad acciones\_tomadas = \{\}$

Mientras existan trabajos sin procesar:

Por cada Agente (máquina)

Inicializar s como el conjunto de operaciones en cola.

Inicializar $acciones$ como el conjunto de posibles operaciones a ejecutar.

Seleccionar a dentro de $acciones$ usando una política ε-*greedy*.

Agregar a a $acciones\_tomadas$.

Tomar acción a, observar s' y r como la función de recompensa temporal de cada objetivo (ver acápite 2.2.1)

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

$$s \leftarrow s'$$

Para cada acción a en $acciones\_tomadas$:

Actualizar todos los estados en que aparezca a tomando como r la recompensa global correspondiente a cada objetivo.

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

Para la secuencia obtenida se calculan los objetivos restantes y de ser posible se incluye la solución en el Frente de Pareto

2.5.1 Ejemplo de utilización

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 MOQL serán los siguientes: episodios 2, ritmo de aprendizaje $(\alpha) = 0.1$, factor de descuento $(\gamma) = 0.8$ y $\varepsilon = 0.2$. Como en este caso hay dos máquinas, se tienen dos agentes (A y B); que representarán a la máquina uno y a la dos respectivamente. El MOQL ejecuta el *makespan*, luego el *flow time* y por último el *tardiness*. Se muestra solo el ejemplo para la optimización del *makespan*, en el caso de los otros dos objetivos sería el mismo procedimiento cambiando solamente el mecanismo de recompensa del agente.

trabajo	Máquina(tiempo de procesamiento)	
0	0(3)	1(3)
1	1(2)	0(3)
2	1(3)	0(2)

Figura 2.6 Ejemplo de problema JSSP

Optimizando el *makespan*:

Primero se inicializan los Q-valores para cada agente:

Estado	Acción	Q-valor

Episodio 1:

Inicializar el conjunto de acciones tomadas en este episodio:

$$actualización\_temporal = \{\}$$

Iteración 1 Agente A:

- El agente observa el estado del ambiente $s = \{J0: 0\}$.

- Evaluar el conjunto de acciones posibles, en este caso hay solo una opción $acciones = \{J0: 0\}$.
- Supongamos que generamos un valor aleatorio menor que ε y por tanto escogemos una acción aleatoriamente, en este caso solo existe la opción $a = J0: 0$.
- Incluimos a en $actualización\_temporal$;

  $acciones\_tomadas = \{J0: 0\}$
- Realizar la acción seleccionada:

  $s' = \{\}$

  $r = 1/\,\text{tiempoProcesamiento}\,J0: 0 = 1/3 = 0.33$
- Actualizamos el Q-valor correspondiente:

  $Q(s, a) = 0 + 0.2[0.33 + 0.7 * 0 - 0] = 0.066$

Estado	Acción	Q-valor
$\{J0: 0\}$	$J0: 0$	0.066

Iteración 1 Agente B:

- El agente observa el estado del ambiente $s = \{J0: 1, J1: 0, J2: 0\}$.
- Evaluar el conjunto de acciones posibles, en este caso hay tres opciones $acciones = \{J0: 1, J1: 0, J2: 0\}$.
- Supóngase que se genera un valor aleatorio menor que ε y por tanto se escoge una acción aleatoria, en este caso la opción $a = J1: 0$.
- Se incluye a en $acciones\_tomadas$;

  $acciones\_tomadas = \{J0: 0, J1: 0\}$
- Realizar la acción seleccionada:

  $s' = \{J0: 1, J2: 0\}$

  $r = 1/\,\text{tiempoProcesamiento}\,J1: 0 = 1/2 = 0.5$
- Se actualiza el Q-valor correspondiente:

$$Q(s,a) = 0 + 0.2[0.5 + 0.7 * 0 - 0] = 0.1$$

Estado	Acción	Q-valor
$\{J0:1, J1:0, J2:0\}$	$J1:0$	0.1

Iteración 2 Agente A:

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

  $s' = \{\}$

  $r = 1/\,\text{tiempoProcesamiento}\,J1:1 = 1/3 = 0.33$
- Se actualiza el Q-valor correspondiente:

  $$Q(s,a) = 0 + 0.2[0.33 + 0.7 * 0 - 0] = 0.066$$

Estado	Acción	Q-valor
$\{J0:0\}$	$J0:0$	0.066
$\{J1:1\}$	$J1:1$	0.066

Iteración 2 Agente B:

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

$s' = \{J2: 0\}$

$r = 1/\,\text{tiempoProcesamiento}\,J0: 1 = 1/3 = 0.33$

- Se actualiza el Q-valor correspondiente:

$Q(s, a) = 0 + 0.2[0.33 + 0.7 * 0 - 0] = 0.066$

Estado	Acción	Q-valor
$\{J0: 1, J1: 0, J2: 0\}$	$J1: 0$	0.1
$\{J0: 1, J2: 0\}$	$J0: 1$	0.066

Iteración 3 Agente A:

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

$s' = \{\}$

$r = 1/\,\text{tiempoProcesamiento}\,J2: 1 = 1/2 = 0.5$

- Se actualiza el Q-valor correspondiente:

$Q(s, a) = 0 + 0.2[0.5 + 0.7 * 0 - 0] = 0.1$

Estado	Acción	Q-valor
$\{J0: 0\}$	$J0: 0$	0.066
$\{J1: 1\}$	$J1: 1$	0.066
$\{J2: 1\}$	$J2: 1$	0.1

Iteración 3 Agente B:

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

  $s' = \{\}$

  $r = 1/\,\text{tiempoProcesamiento}\,J2{:}\,0 = 1/3 = 0.33$

- Se actualiza el Q-valor correspondiente:

  $Q(s,a) = 0 + 0.2[0.33 + 0.7 * 0 - 0] = 0.066$

Estado	Acción	Q-valor
$\{J0{:}\,1, J1{:}\,0, J2{:}\,0\}$	$J1{:}\,0$	0.1
$\{J0{:}\,1, J2{:}\,0\}$	$J0{:}\,1$	0.066
$\{J2{:}\,0\}$	$J2{:}\,0$	0.1

Al finalizar la iteración 3 todos los trabajos han sido procesados obteniendo la siguiente secuencia: $\{J0{:}\,0, J1{:}\,0, J1{:}\,1, J0{:}\,1, J2{:}\,1, J2{:}\,0\}$ con un *makespan* de 11 unidades de tiempo como se muestra en la **Figura 2.17**.

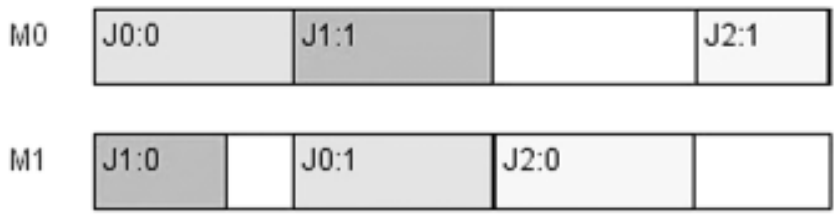


Figura 2.7 Secuencia encontrada para el ejemplo de la **Figura 2.6** después del primer episodio

Para todos las acciones en $acciones\_tomadas$ se les actualizan los Q-valores tomando como recompensa $1/makespan$ de la secuencia encontrada.

Al terminar el episodio se calculan los otros dos objetivos para esta secuencia obtenida y se trata de incluir la terna (*makespan, flowtime, tardiness*) de esta secuencia en el Frente de Pareto como se vio en la sección anterior (2.4).

Episodio 2:

$acciones\_tomadas = \{\}$

En este episodio se mantendrán los Q-valores para cada agente

Iteración 1 Agente A:

Se mantiene:

Estado	Acción	Q-valor
$\{J0:0\}$	$J0:0$	0.066
$\{J1:1\}$	$J1:1$	0.066
$\{J2:1\}$	$J2:1$	0.1

- $s = \{J0:0\}, \quad acciones = \{J0:0\}.$
- $a = J0:0.$
- $acciones\_tomadas = \{J0:0\}$
- $s' = \{\}, \qquad r = 1/\,\text{tiempoProcesamiento}\,J0:0 = 1/3 = 0.33$
- $Q(s,a) = 0.066 + 0.2[0.33 + 0.7 * 0 - 0.066] = 0.1188$

Estado	Acción	Q-valor
$\{J0:0\}$	$J0:0$	0.1188
$\{J1:1\}$	$J1:1$	0.066
$\{J2:1\}$	$J2:1$	0.1

Iteración 1 Agente B:

Se mantiene:

Estado	Acción	Q-valor

$\{J0:1, J1:0, J2:0\}$	$J1:0$	0.1
$\{J0:1, J2:0\}$	$J0:1$	0.066
$\{J2:0\}$	$J2:0$	0.1

- $s = \{J0:1, J1:0, J2:0\}$, $acciones = \{J0:1, J1:0, J2:0\}$.
- Se obtiene un número aleatorio mayor que ε y por tanto se escoge la acción que mayor Q-valor tiene $a = J1:0$
- $acciones\_tomadas = \{J0:0,\ J1:0\}$
- $s' = \{J0:1, J2:0\}$, $r = 1/\text{tiempoProcesamiento}\, J1:0 = 1/2 = 0.5$
- $Q(s,a) = 0.1 + 0.2[0.5 + 0.7 * 0.066 - 0.1] = 0.18924$

Estado	Acción	Q-valor
$\{J0:1, J1:0, J2:0\}$	$J1:0$	0.18924
$\{J0:1, J2:0\}$	$J0:1$	0.066
$\{J2:0\}$	$J2:0$	0.1

Iteración 2 Agente A:

- $s = \{J1:1\}$, $acciones = \{J1:1\}$
- $a = J1:1$
- $acciones\_tomadas = \{J0:0,\ J1:0, J1:1\}$
- $s' = \{\}$, $r = 1/\text{tiempoProcesamiento}\, J1:1 = 1/3 = 0.33$
- $Q(s,a) = 0.066 + 0.2[0.33 + 0.7 * 0 - 0.066] = 0.1188$

Estado	Acción	Q-valor
$\{J0:0\}$	$J0:0$	0.1188
$\{J1:1\}$	$J1:1$	0.1188

$\{J2:1\}$	$J2:1$	0.18

Iteración 2 Agente B:

- $s = \{J2:0, J0:1\}, \quad acciones = \{J2:0, J0:1\}.$
- $a = J2:0$
- $acciones\_tomadas = \{J0:0,\ J1:0, J1:1, J2:0\}$
- $s' = \{J0:1\},\ \ r = 1/\text{tiempoProcesamiento}\, J2:0 = 1/3 = 0.33$
- $Q(s,a) = 0 + 0.2[0.33 + 0.7 * 0 - 0] = 0.066$

Estado	Acción	Q-valor
$\{J0:1, J1:0, J2:0\}$	$J1:0$	0.1
$\{J0:1, J2:0\}$	$J0:1$	0.066
$\{J2:0\}$	$J2:0$	0.1
$\{J0:1, J2:0\}$	$J2:0$	0.066

Iteración 3 Agente A:

- $s = \{J2:1\}, \quad acciones = \{J2:1\}$
- $a = J2:1$
- $acciones\_tomadas = \{J0:0,\ J1:0, J1:1, J2:0, J2:1\}$
- $s' = \{\}, \quad r = 1/\text{tiempoProcesamiento}\, J2:1 = 1/2 = 0.5$
- $Q(s,a) = 0.18 + 0.2[0.5 + 0.7 * 0 - 0.18] = 0.244$

Estado	Acción	Q-valor
$\{J0:0\}$	$J0:0$	0.1188
$\{J1:1\}$	$J1:1$	0.1188
$\{J2:1\}$	$J2:1$	0.244

Iteración 3 Agente B:

- $s = \{J0:1\}, \quad acciones = \{J0:1\}$.
- $a = J0:1$
- $acciones\_tomadas = \{J0:0,\ J1:0, J1:1, J2:0, J2:1, J0:1\}$
- $s' = \{\}, \qquad r = 1/\text{tiempoProcesamiento}\, J0:1 = 1/3 = 0.33$
- $Q(s,a) = 0 + 0.2[0.33 + 0.7 * 0 - 0] = 0.066$

Estado	Acción	Q-valor
$\{J0:1, J1:0, J2:0\}$	$J1:0$	0.1
$\{J0:1, J2:0\}$	$J0:1$	0.066
$\{J2:0\}$	$J2:0$	0.1
$\{J0:1, J2:0\}$	$J2:0$	0.066
$\{J0:1\}$	$J0:1$	0.066

Al finalizar la iteración 3 todos los trabajos han sido procesados obteniéndose la siguiente secuencia: $\{J0:0,\ J1:0, J1:1, J2:0, J2:1, J0:1\}$ con un *makespan* de ocho unidades de tiempo como se muestra es la **Figura 2.18**.

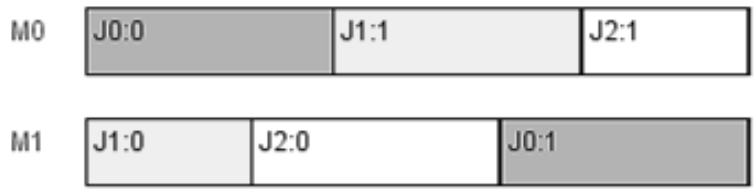


Figura 2.8 Secuencia encontrada para el ejemplo de la **Figura 2.6** después del segundo episodio

Para todos las acciones en $acciones\_tomadas$ les actualizamos los Q-valores tomando como recompensa $1/makespan$ de la secuencia encontrada.

Al terminar el episodio se calculan los otros dos objetivos para la secuencia obtenida y se trata de incluir la terna (*makespan, flowtime, tardiness*) de esta secuencia en el Frente de Pareto como se vio en la sección anterior (2.4).

Después de realizado todo el proceso con los tres objetivos se obtiene el siguiente Frente de Pareto: {(8,21,8)} la cual representa la secuencia representada en la **Figura 2.19**:

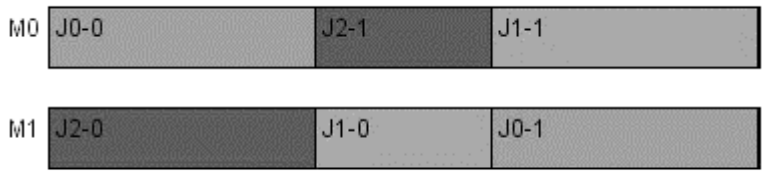


Figura 2.9 Secuencia que representa la terna obtenida en el Frente de Pareto

En esta, se obtuvo un *makespan* = 8, *flowtime* = 21 y *tardiness* = 8

2.6 Propuesta de Aplicación

Con el objetivo de facilitar el trabajo se creó una aplicación que le brinda al usuario una forma más intuitiva y cómoda de lidiar con el problema en cuestión. Al ejecutar el programa se muestra la interfaz de configuración del MOQL la cual se muestra en la **Figura 2.10**.

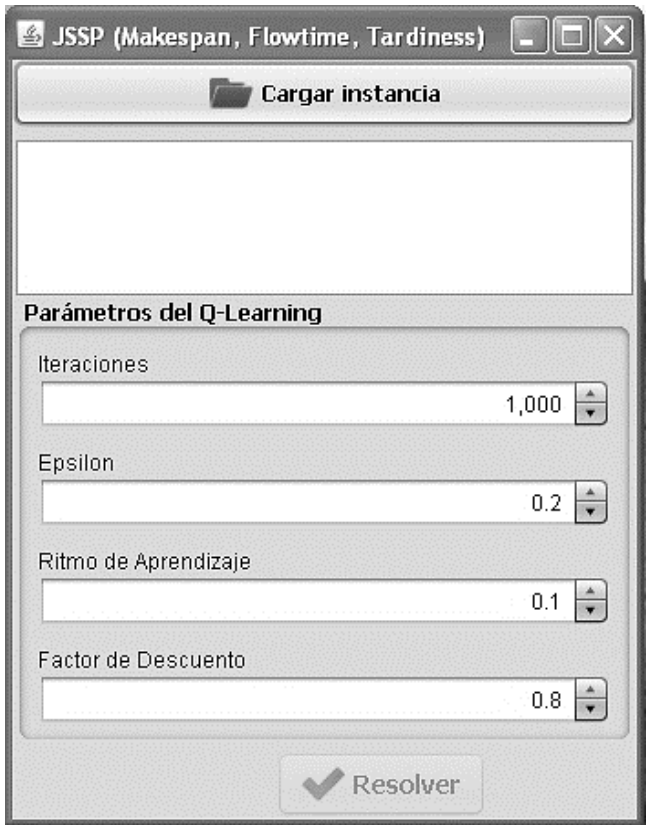


Figura 2.10 Interfaz de la aplicación propuesta

Las utilidades de esta interfaz están enfocadas a la modificación de los parámetros de entrada del MOQL. Luego de cargar la instancia a resolver y especificar los valores del Ritmo de Aprendizaje, el Factor de Descuento, épsilon y los episodios se resuelve el problema.

Las soluciones se mostrarán en un pequeño panel que brinda al usuario las opciones de visualizar y guardar las secuencias correspondientes a cada solución del Frente de Pareto obtenido.

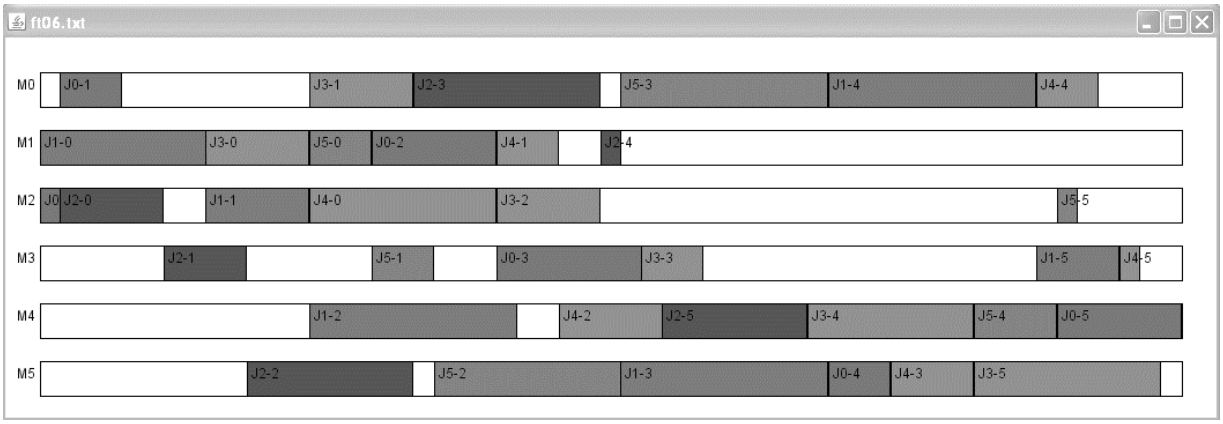


Figura 2.11 Visualización de una de las secuencias encontradas por MOQL para la instancia ft06

La aplicación salva los resultados de la corrida en un archivo (txt) y una vez salvados se le pueden efectuar todas las operaciones deseadas, así como visualizarlos de una manera más sencilla.

Se implementó una pequeña aplicación, con una interfaz gráfica amigable y de fácil utilización, que le brinda la posibilidad de configurar todos los parámetros de entrada del MOQL, así como visualizar los resultados obtenidos además de poder exportarlos a archivos de textos.

2.7 Conclusiones Parciales

Para dar solución al problema planteado se asoció un agente a cada recurso, este agente se encargará de interactuar con el ambiente, obtener el estado en que se encuentra y seleccionar una de las posibles acciones existentes siempre y cuando el recurso al que fue asignado esté disponible.

El uso de agentes inteligentes junto RL evidenció ser un enfoque natural en la solución de los problemas JSSP. Para recompensar la elección de los agentes existen varias funciones de recompensa basadas principalmente en el tiempo de procesamiento de las operaciones, el tiempo de finalización de los trabajos y las fechas de entrega. Las funciones de recompensa se usaron tanto locales como globales, basándose estas últimas en las soluciones reales de los objetivos por episodio; para los tres objetivos que se optimizan. Como estrategia de selección de acciones se implementó la ε-*greedy* pues proporciona una mayor exploración del espacio de soluciones. La forma de dar solución al problema multi-objetivo planteado está basada en

la Optimalidad de Pareto para lo cual se definió una relación de dominancia que es utilizada para generar el frente de Pareto que resulta como solución del MOQL.

Para una mejor vinculación entre el secuenciador y el algoritmo se implementó una pequeña aplicación, con una interfaz gráfica amigable y de fácil utilización, que le brinda la posibilidad de configurar todos los parámetros de entrada del MOQL, así como visualizar los resultados obtenidos además de poder exportarlos a archivos de textos.

CAPÍTULO 3. PRUEBA EXPERIMENTAL Y ANÁLISIS DE LOS RESULTADOS

En el siguiente capítulo se realizan diferentes experimentos en con el fin de validar el MOQL. Además se proponen nuevos casos de estudio para futuras comparaciones.

3.1 Evaluación de las Fronteras de Pareto

La solución de un problema multi-objetivo resulta en un conjunto de soluciones no dominadas conocido como Frontera de Pareto. Estas soluciones no son más que ternas que contienen en cada posición la cota superior (UB por sus siglas en inglés) encontrada por el algoritmo en una iteración para cada objetivo, $(UB\ makespan, UB\ flow\ time, UB\ tardiness)$.

Para evaluar el comportamiento del MOQL varias métricas se proponen en la literatura (Knowles and Corne, 2002). La mayoría de estas se aplican al conjunto de soluciones no dominadas obtenidas. Debido a esto, se ha de tener en cuenta cuántas se deben utilizar para medir el rendimiento del MOQL, sobre esto (Zitzler et al., 2000) planteó que para un problema multi-objetivo con *n* objetivos al menos *n* métricas debían ser evaluadas.

En (Zitzler et al., 2000) se sugieren tres rasgos dentro de la búsqueda multi-objetivo de Pareto que pueden ser identificados y medidos.

- La distancia entre el conjunto de soluciones no dominadas obtenidas y el frente óptimo de Pareto debe minimizarse (esto si se conoce el Frente Global de Pareto).
- Maximizar la diversidad de la solución encontrada y así obtener una distribución de vectores tan nivelada y uniforme como sea posible.
- El tamaño del frente no dominado debe ser maximizado (para cada objetivo un amplio rango de valores debe ser cubierto por las soluciones obtenidas).

La mayoría de las métricas reportadas en la literatura revisada asumen que el Frente Óptimo de Pareto es conocido o se puede generar. Cuando ese es el caso se puede medir la efectividad del MOQL comparando el Frente de Pareto que este generó con el Frente de Pareto Global y luego calcular ciertas medidas de error que pueden ayudar a determinar la efectividad del MOQL. En ambientes no controlados no se conoce el Frente Global de Pareto por lo que sería imposible medir las soluciones del algoritmo utilizado. Generar el Frente Global de Pareto

requiere un alto esfuerzo computacional, especialmente para el JSSP en estudio. La complejidad computacional se vuelve mayor a medida que crece el tamaño del problema. Es por ello que otras métricas se han desarrollado para medir la diversidad y el cubrimiento de las soluciones generadas por algunos algoritmos multi-objetivos.

El MOQL trata de optimizar, como ya se conoce, tres objetivos, de los cuales uno ha sido utilizado muy poco en enfoques multi-objetivos. Es por ello que para medir el comportamiento de este algoritmo se aplicarán las métricas a solo dos de los tres objetivos a optimizar en pos de poder establecer una comparación entre MOQL y los reportados en la literatura.

Los objetivos que se usan para formar los Frentes de Pareto a los que luego se les aplicarán los experimentos de comparación son el *makespan* y el *flow time* ya que no se encontraron reportados en la literatura casos en los cuales se trate de resolver el mismo problema multi-objetivo que aquí se plantea. Por tanto, se establecen comparaciones que evalúen el funcionamiento de MOQL para dos objetivos, las que permiten medir qué tan bueno es el desempeño del algoritmo comparado con otros de la literatura.

En el caso del *tardiness* se propone en el acápite siguiente un caso de estudio con el fin de poder establecer una base para futuras comparaciones.

Es importante resaltar que el algoritmo fue implementado en el lenguaje java JDK 1.7, como entorno de desarrollo (IDE) se utilizó *NetBeans* 7.4 y los experimentos fueron ejecutados en un procesador *Intel Pentium* IV a 3.40 GHz, con 3 GB de RAM.

En el funcionamiento del algoritmo Q-L intervienen parámetros que pueden tomar diferentes valores (Martínez, 2012):

- α es la proporción de aprendizaje, es un valor entre cero y uno y determina el grado en que se va a actualizar el q-valor si $\alpha = 0$, no habrá actualización; si $\alpha = 1$ el valor antiguo es remplazado por el nuevo estimado. Usualmente es utilizado un valor pequeño para la proporción de aprendizaje, por ejemplo $\alpha = 0.1$.
- γ es el factor de descuento que tiene el rango de valores desde 0 hasta 1. Si γ es cercano a cero el agente tiende a considerar solamente la recompensa inmediata. Si γ es cercano a uno, el agente considerará la recompensa futura en mayor medida. Se reporta $\gamma = 0.8$ como un valor frecuentemente utilizado en el algoritmo.

- ε es un umbral que permite el balance entre la explotación y la exploración. Todas las acciones a realizar tienen asociadas una probabilidad generada aleatoriamente, si esta probabilidad está por debajo de ε se selecciona una acción aleatoria, de lo contrario se selecciona una acción de acuerdo con la política del agente. El valor 0.2 es usualmente utilizado para este parámetro en la literatura.
- El número de ciclos es el parámetro de parada del algoritmo.

Como los problemas muti-objetivos encuentran un conjunto de soluciones no dominadas respecto a múltiples objetivos, la comparación entre diferentes algoritmos multi-objetivo no es tarea sencilla.

3.1.1 Métricas usadas para la evaluación. Experimentos y resultados.

Dentro de las métricas usadas para comparar y evaluar algoritmos multi-objetivo reportadas en la literatura se usan dos para hacer la validación de la solución propuesta. Los frentes obtenidos por el MOQL se comparan con los obtenidos por otros algoritmos multi-objetivo.

En las siguientes tablas **Tabla 3.41**, **Tabla 3.5** y **Tabla 3.53** se plantean los frentes obtenidos a partir de los cuales se realizan las respectivas validaciones.

FT06 (n=6, m=6)	
Makespan	*Flow time*
58	279
57	297
64	265
55	301

Tabla 3.1 Frente de Pareto obtenido por el MOQL para el *makespan* y el *flow time* en la instancia ft06.

LA01 (n=10, m=5)		**LA02 (n=10, m=5)**		**LA03 (n=10, m=5)**		**LA04 (n=10, m=5)**		**LA05 (n=10, m=5)**	
Makespan	*Flow time*	*Makespan*	*Flow time*	*Makespan*	*Flow time*	*Makespan*	*Flow time*	*Makespan*	*Flow time*
704	4837	700	4480	607	4773	644	4263	648	4084
710	4832	686	5121	665	4253	628	4371	600	4118
667	5084	690	4489	684	4181	626	4381	643	4090
670	4894	687	4563	631	4451	611	4460	593	4349
666	5162	750	4468	632	4442	640	4345	650	4072
680	4889	692	4485	636	4425				
		689	4562	684	4178				
				677	4215				
				688	4175				

Tabla 3.2 Frente de Pareto obtenido por el MOQL para el *makespan* y el *flow time* en las instancias la01 a la la05.

LA06 (n=15, m=5)		**LA07 (n=15, m=5)**		**LA08 (n=15, m=5)**		**LA09 (n=15, m=5)**		**LA10 (n=15, m=5)**	
Makespan	*Flow time*	*Makespan*	*Flow time*	*Makespan*	*Flow time*	*Makespan*	*Flow time*	*Makespan*	*Flow time*
1013	8736	890	8473	863	8574	951	9931	1140	9025
926	8865	891	8415	935	8008	956	9784	958	9125
960	8838	895	8390	868	8529	1115	9235	965	9105
984	8778	904	8350	869	8477	966	9260	984	9054
989	8765	1106	8342	870	8450	968	9258		
1012	8737			871	8445				
				887	8242				
				908	8160				
				909	8150				
				915	8140				

Tabla 3.3 Frente de Pareto obtenido por el MOQL para el *makespan* y el *flow time* en las instancias la01-05

3.1.2 Por ciento de incremento medio relativo (MRPI)

Una de las medidas que se usan para evaluar el algoritmo es el por ciento de incremento medio relativo (MRPI por sus siglas en inglés). Esta mide la efectividad del frente de Pareto encontrado considerando las soluciones extremas (es decir el mejor *makespan* y el mejor *flow time* que se logró en el frente encontrado) como referencia. Una medida absoluta llamada MRPI de las soluciones extremas del frente no dominado obtenido con respecto a las mejores cotas superiores reportadas en la literatura.

Para calcular MRPI se obtiene el error medio relativo como la diferencia entre lo alcanzado por el MOQL y la mejor cota superior (UB por sus siglas en inglés) reportada en la literatura, esta diferencia se divide por UB y todo esto se multiplica por 100. Para calcular MRPI se agrupan las instancias por su tamaño, se suman los errores relativos medios y se divide por la cantidad de instancias en el grupo.

Las instancias utilizadas para realizar los experimentos son la ft06 y la la01 a la la015 que se encuentran en la OR-Library.

En las siguientes tablas **Tabla 3.4** y **Tabla 3.5** se muestra el MRPI calculado para las instancias antes mencionadas tomando los valores extremos obtenidos por objetivo en los frentes resultantes.

Instancia	**N**	**M**	***Makespan* (UB)**	**Mejor *makespan* obtenido**	**% Desviación**	**MRPI *makespan***
ft06	6	6	55	55	0,00	0,00
la01	10	5	666	666	0,00	
la02	10	5	655	686	4,73	
la03	10	5	597	607	1,68	1,994
la04	10	5	590	611	3,56	

la05	10	5	593	593	0,00	
la06	15	5	926	926	0,00	
la07	15	5	890	890	0,00	
la08	15	5	863	863	0,00	0,00
la09	15	5	951	951	0,00	
la10	15	5	958	958	0,00	
la11	20	5	1222	1222	0,00	
la12	20	5	1039	1039	0,00	
la13	20	5	1150	1150	0,00	0,48
la14	20	5	1292	1292	0,00	
la15	20	5	1207	1236	2,40	

Tabla 3.4 Comportamiento del MOQL con respecto a las mejores cotas superiores propuestas en la literatura para la instancia ft06 de (Fisher and Thompson, 1963) y las instancias de (Lawrence, 1984), medido en términos de MRPI en el *makespan*

Instancia	**N**	**M**	**flow time (UB)**	**Mejor flow time obtenido**	**% Desviación**	**MRPI Flow time**
ft06	6	6	265	265	0,00	0,00
la01	10	5	4832	4832	0,00	
la02	10	5	4459	4468	0,20	

la03	10	5	4151	4175	0,58	0.108
la04	10	5	4259	4263	0,09	
la05	10	5	4072	4072	0,00	
la06	15	5	8725	8736	0,13	
la07	15	5	8251	8342	1,10	
la08	15	5	7948	8008	0,75	0.668
la09	15	5	9235	9348	1,22	
la10	15	5	9012	9025	0,14	
la11	20	5	14590	14600	0,07	
la12	20	5	12134	12134	0,00	
la13	20	5	13617	13833	1,59	0.58
la14	20	5	14964	15108	0,96	
la15	20	5	14621	14662	0,28	

Tabla 3.5 Comportamiento del MOQL con respecto a las mejores cotas superiores propuestas en la literatura para la instancia ft06 de (Fisher and Thompson, 1963) y las instancias de (Lawrence, 1984), medido en términos de MRPI en el *flow time*.

Para validar los resultados del método propuesto se compara con una meta-heurísticas basada en la técnica de Recocido Simulado llamada Pareto *archived simulated annealing for Job Shop scheduling with multiple objectives* (PASA) propuesta por (Suresh and Mohanasundaram, 2006). Este genera una solución inicial de forma aleatoria, luego usa un nuevo mecanismo de perturbación llamado *Segment-Random Insertion* (SRI) para generar un

conjunto de soluciones ubicadas en la vecindad de la solución actual. El PASA realiza una búsqueda del conjunto de soluciones no dominadas basándose en la dominancia de Pareto o a través de la implementación de una simple función de probabilidad. En el artículo antes mencionado se minimizan el *makespan* y el *mean flow time* (media del *flow time*), para realizar las comparaciones se utilizaron los resultados publicados en dicho artículo. La **Tabla 3.56** muestra los MRPI resultantes para el *makespan* y el *flow time* de las instancias en estudio.

Instancias	**MRPI *Makespan***		**MRPI *Flow time***	
	MOQL	PASA	MOQL	PASA
ft06	0.00	0.00	0.00	0.00
la01-05	1.994	0.00	0.108	0.16
la06-10	0.00	0.00	0.668	0.17
la11-15	0.48	0.00	0.58	0.51

Tabla 3.6 MRPI obtenido de aplicar el MOQL y PASA a las instancias ft06 y la01 a la la15

Para esta comparación se utiliza el test de *Wilcoxon* con un nivel de significación de 0.05, los resultados son mostrados en la **Tabla 3.7** y **Tabla 3.8**.

Algoritmos	R^+	R^-	**Significación**
PASA- MOQL	3	0	**0.180**

Tabla 3.7 Resultados del test de *Wilcoxon* para la comparación entre MOQL y el PASA para el *makespan*

Algoritmos	R^+	R^-	**Significación**
PASA- MOQL	5	1	**0.285**

Tabla 3.8 Resultados del test de *Wilcoxon* para la comparación entre MOQL y el PASA para el *flow time*

Como puede observarse, no existen diferencias significativas entre los dos algoritmos en cuanto a los resultados obtenidos por lo que se puede considerar que el MOQL tiene un desempeño tan bueno como el PASA.

3.1.3 Net front contribution ratio (NFCR)

Cuando se comparan dos algoritmos, se cuenta con dos conjuntos de soluciones no dominadas. Entre esos conjuntos pueden existir soluciones que estén en un conjunto que dominen a las soluciones que estén en el otro o viceversa.

Una de las medidas que se usan en esta investigación para comparar se basa en encontrar el net frente no dominado. Para evaluar cada algoritmo es usada la métrica *net front contribution ratio* (NFCR). Suponga que se quieren comparar dos algoritmos *A1* y *A2*, se tiene un frente *a* resultado del algoritmo *A1* y un frente *b* resultado del algoritmo *A2*. Se forma entonces el frente *c* que está compuesto por la combinación de las soluciones de los frentes a y *b* que no se dominan entre ellas. Por tanto, se tiene *n1* y *n2* que son la cantidad de soluciones no dominada del frente *a* y del frente *b* que pasaron a formar parte del frente *c*. El NFCR de cada algoritmo se calcula entonces como sigue:

$$NFCR_{A1} = n1/n$$

$$NFCR_{A2} = n2/n$$

Donde *n* es la cantidad de soluciones no dominadas que forman el frente *c*.

En las **Tabla 3.9**, **Tabla 3.10** y **Tabla 3.11**, se muestran los Frentes de Pareto obtenidos por PASA para las instancias ft06 y la01-la15, los cuales fueron usados para crear los frentes *net* que se plantean a continuación de las tablas.

Como se puede observar en (Suresh and Mohanasundaram, 2006), se obtuvieron los Frentes de Pareto para minimizar el *makespan* y el *mean flow time* que no es más que el clásico *flow time* entre la cantidad de trabajos de la instancia al que se le calcula. Con el objetivo de realizar la comparación con el enfoque propuesto se calculó el *flow time* multiplicando el *mean flow time* obtenido por el PASA por la cantidad de trabajos de la instancia.

LA01 (n=10, m=5)		**LA02 (n=10, m=5)**		**LA03 (n=10, m=5)**		**LA04 (n=10, m=5)**		**LA05 (n=10, m=5)**	
Makespan	*Mean flow time*	*Makespan*	*Mean flow time*	*Makespan*	*Mean flow time*	*Makespan*	*Mean flow time*	*Makespan*	*Mean flow time*
937	4832	655	491.000	597	528.600	590	487.100	593	422.700
666	4953	660	498.100	598	526.500	594	484.700	600	416.500
677	4840	663	472.100	603	484.300	598	465.000	605	416.200
766	4833	669	459.500	606	478.500	605	456.300	606	412.100
		687	458.000	614	477.300	608	455.600	643	409.100
		694	457.700	618	472.300	610	448.900	648	408.300
		699	457.600	619	459.800	616	446.200	656	407.200
		709	457.600	627	459.500	630	445.100		
		614	453.300	628	450.200	636	440.000		
		629	448.300	631	445.200	648	434.800		
		747	447.900	632	444.700	652	432.300		
		867	446.800	636	442.900	655	426.000		
				638	440.800	686	425.900		
				640	427.500				

665	425.300
672	422.700
677	421.100
684	418.100
689	417.500

Tabla 3.9 Frente de Pareto obtenido por PASA para la instancia la01-05

LA06 (n=15, m=5)		**LA07 (n=15, m=5)**		**LA08 (n=15, m=5)**		**LA09 (n=15, m=5)**		**LA10 (n=15, m=5)**	
Makespan	*Mean flow time*	*Makespan*	*Mean flow time*	*Makespan*	*Mean flow time*	*Makespan*	*Mean flow time*	*Makespan*	*Mean flow time*
926	591.200	890	564.930	863	571.800	951	620.07	958	608.4
961	588.730	891	561.730	865	570.730	966	617.47	965	607.13
984	585.070	892	561.730	866	569.600	968	617.27	984	604.67
989	584.530	895	559.600	868	568.730	972	612.67	985	598.13
1067	583.270	904	557.270	869	565.800	975	609.87	1042	592.2
1086	582.400	905	555.870	870	564.200			1044	591.47
		906	552.600	871	563.270			1050	588.8
		968	550.330	878	562.270			1053	588.07
		4038	546.470	883	552.930				

1106	542.800	887	549.470
		894	547.330
		900	544.470
		908	543.400
		909	543.200
		910	542.730
		913	538.730
		940	533.870

Tabla 3.10 Frente de Pareto obtenido por PASA para la instancia la06 a la la10

	FT06 (n=6, m=6)				
Makespan	55	57	58	60	64
Flow time	50.170	49.500	46.670	45.00	44.170

Tabla 3.11 Frente de Pareto obtenido por PASA para la instancia ft06

Los frentes Net obtenidos de la aplicación del algoritmo MOQL a las instancias ft06 y la01 a la la10 se muestran a continuación en las tablas **Tabla 3.5**, **Tabla 3.5** y **Tabla 3.5**:

LA01 (n=10, m=5)		**LA02 (n=10, m=5)**		**LA03 (n=10, m=5)**		**LA04 (n=10, m=5)**		**LA05 (n=10, m=5)**	
Makespan	*Flow time*	*Makespan*	*Flow time*	*Makespan*	*Flow time*	*Makespan*	*Flow time*	*Makespan*	*Flow time*
704	4837	700	4480	607	4773	644	4263	600	4118
710	4832	750	4468	665	4253	628	4371	643	4090
670	4894	614	4533	631	4451	626	4381	650	4072
680	4889	629	4483	632	4442	611	4460	593	4227
666	4953	747	4479	636	4425	640	4345	648	4083
679	4893			684	4178	590	4871		
684	4881			688	4175	594	4847		
691	4858			597	5286	598	4650		
				598	5265	605	4563		
				603	4843	608	4556		
				606	4785	610	4489		
				618	4723	655	4260		
				619	4598	686	4259		
				627	4595				
				638	4408				

				640	4275				
				672	4227				
				677	4211				

Tabla 3.12 Frentes Net obtenidos de aplicar el MOQL a las instancias la01 a la la05

LA06 (n=15, m=5)		**LA07 (n=15, m=5)**		**LA08 638(n=15, m=5)**		**LA09 (n=15, m=5)**		**LA10 (n=15, m=5)**	
Makespan	*Flow time*	*Makespan*	*Flow time*	*Makespan*	*Flow time*	*Makespan*	*Flow time*	*Makespan*	*Flow time*
1013	8736	890	8473	863	8574	966	9260	958	9125
926	8865	891	8415	935	8008	968	9258	965	9105
960	8838	895	8390	868	8529	951	9301	984	9054
989	8765	904	8350	869	8477	972	9190	985	8971
1012	8737	905	8338	870	8450	975	9148	1042	8883
961	8831	906	8289	871	8445			1044	8872
984	8776	968	8255	887	8242			1050	8832
				865	8561			1053	8821
				866	8544				
				878	8434				
				883	8294				

894	8210
900	8167
908	8151
909	8148
910	8140
913	8081

Tabla 3.13 Frentes Net obtenidos de aplicar el MOQL a las instancias la06 a la la10

	FT06 (n=6, m=6)				
Makespan	58	57	64	55	60
Flow time	279	297	265	301	270

Tabla 3.14 Frente Net obtenido de aplicar el MOQL a la instancia ft06

Después de haber calculado los frentes Net en la **Tabla 3.5** se exponen los NFCR para cada algoritmo en las instancias usadas.

ALGORITMOS	**INSTANCIAS**										
	ft06	***la01***	***la02***	***la03***	***la04***	***la05***	***la06***	***la07***	***la08***	***la09***	***la10***
MOQL	0.2	0.5	0.6	0.39	0.62	0.4	0.71	0.56	0.41	0.4	0.38
PASA	0.2	0.5	0.4	0.61	0.38	0.6	0.29	0.44	0.59	0.6	0.62

Tabla 3.15 NFCR calculado a al MOQL y al PASA para las instancias ft06 y la01 a la la10

Luego de obtener los frentes Net y calcular el NFCR del MOQL y el PASA para cada instancia se utiliza el test de *Wilcoxon* con un nivel de significación de 0.05 para realizar pruebas estadísticas de las soluciones, los resultados son mostrados en la **Tabla 3.16**.

Algoritmos	R^+	R^-	**Significación**
PASA- MOQL	4.70	23.50	**0.905**

Tabla 3.16 Resultados del test de *Wilcoxon* para la comparación entre MOQL y el PASA para el NFCR

Como puede observarse, no existen diferencias significativas entre los dos algoritmos en cuanto a los resultados obtenidos por lo que se puede considerar que el MOQL tiene un desempeño tan bueno como el PASA.

3.2 Caso de estudio. Tardiness

En el caso del *tardiness* se plantean los mejores resultados obtenidos para dos instancias seleccionadas, la ft06 y la la05 a la cuales se les ha calculado el *due date* y el *release date*. En la literatura revisada se plantea que los tiempos de procesamiento de estas instancias fueron generados de forma aleatoria usando una distribución uniforme y en rangos entre 1 y 100, por tanto las fechas con las cuales se harán los experimentos será un número aleatorio que tenga como cota inferior la suma de los tiempos de procesamiento de las operaciones en cada trabajo y superior ese valor por dos.

El problema con las *due date* y *release date* necesarias para poder minimizar el *tardiness*, es que cada investigador, según se pudo comprobar lo largo de la revisión realizada, las genera de forma diferente y en la mayoría de los casos no son publicadas por lo que se hace tan difícil realizar una comparación cuando se trata de optimizar el *tardiness*. Por lo antes planteado, en el caso de este objetivo no se reportan óptimos conocidos, por ello se expondrán las fechas con el fin de futuras comparaciones.

En las tablas **Tabla 3.5** y **Tabla 3.5** se muestra como quedan generadas las fechas para las instancias ft06 y la05:

6	**6**												
0	**48**	2	1	0	3	1	6	3	7	5	3	4	6
0	**84**	1	8	2	5	4	10	5	10	0	10	3	4
0	**70**	2	5	3	4	5	8	0	9	1	1	4	7
0	**38**	1	5	0	5	2	5	3	3	4	8	5	9
0	**37**	2	9	1	3	4	5	5	4	0	3	3	1
0	**50**	1	3	3	3	5	9	0	10	4	4	2	1

Tabla 3.17 Instancia ft06 con *due date* y *release date*

10	**5**										
0	**384**	1	72	0	87	4	95	2	66	3	60
0	**195**	4	5	3	35	0	48	2	39	1	54
0	**240**	1	46	3	20	2	21	0	97	4	55
0	**247**	0	59	3	19	4	46	1	34	2	37
0	**181**	4	23	2	73	3	25	1	24	0	28
0	**256**	3	28	0	45	4	5	1	78	2	83
0	**297**	0	53	3	71	1	37	4	29	2	12
0	254	4	12	2	87	3	33	1	55	0	38
0	254	2	49	3	83	1	40	0	48	4	7
0	260	2	65	3	17	0	90	4	27	1	23

Tabla 3.18 Instancia la05 con *due date* y *release date*

En la **Tabla 3.5** se muestran las Fronteras de Pareto obtenidas de aplicar el MOQL a las instancias ft06 y la05 después de generado el *due date* y el *release date*

ft06			*la05*		
C_{max}	F	T	C_{max}	F	T
55	301	16	593	4682	348
67	265	29	608	4573	348
59	292	17	604	4680	339
59	284	20	593	4690	339
60	279	22	657	4072	417
60	311	11	605	4408	345
67	297	10	610	4483	370
65	291	12	593	4554	374
68	286	3	610	4439	391
64	306	13	647	4448	387
58	285	19	600	4580	355
66	303	4			
70	290	2			

Tabla 3.19 Fronteras de Pareto resultantes de aplicar MOQL para la optimización de los tres objetivos estudiados

3.3 Conclusiones Parciales

Después de haber realizado varios experimentos para validar el MOQL se concluye:

- Dada la imposibilidad de acceder a los libros y artículos que presentan otras vías para solucionar el problema en cuestión, se decidió evaluar el desempeño del algoritmo multi-objetivo para el *makespan* y el *flow time*.
- Con el fin de realizar una comparación entre el MOQL y el PASA se realizaron varias corridas del algoritmo, los frentes de Pareto obtenidos fueron evaluados y comparados con el PASA usando dos de las métricas reportadas en la literatura.
- Los resultados arrojados por la comparación evidenció que el MOQL obtiene resultados tan buenos como los reportados por el PASA.
- El tercer objetivo fue evaluado mediante dos casos de estudio propuestos basados en las instancias ***ft06*** y ***la05*** a los cuales se les generó el *due date* y el *release date* utilizando una distribución uniforme. Se publicaron estas fechas y los Frentes de Pareto obtenidos para comparaciones en futuras investigaciones.

CONCLUSIONES

Después de una extensa revisión bibliográfica se definieron como objetivos a optimizar por el MOQL: el *makespan*, el *flow time* y el *tardiness* por ser considerados los más importantes. Se diseñó e implementó un algoritmo multi-objetivo basado en RL y usando la Optimalidad de Pareto para resolver problemas de secuenciación de tareas tipo *Job Shop* que optimiza los tres objetivos escogidos. Para una mejor vinculación entre el secuenciador y el algoritmo se implementó una pequeña aplicación, con una interfaz gráfica amigable y de fácil utilización, que le brinda la posibilidad de configurar todos los parámetros de entrada del MOQL, así como visualizar los resultados obtenidos además de poder exportarlos a archivos de textos.

Se evaluaron mediante la aplicación de las métricas reportadas en la literatura los Frentes de Pareto obtenidos de la optimización del *makespan* y el *flow time*. Estos resultados arrojaron la conclusión de que el comportamiento del MOQL es tan bueno como el de otros algoritmos reportados en la bibliografía ya que se comparó con el que mejor comportamiento mostraba según la literatura consultada. Se proponen y evalúan dos casos de estudio para el *tardiness* con el objetivo de realizar comparaciones en futuras investigaciones.

RECOMENDACIONES

Como recomendaciones para extender los resultados obtenidos en este trabajo se proponen:

- Incluir al MOQL la optimización de otros objetivos que puedan ser de interés.
- Integrar el MOQL a la herramienta utilizada para los problemas de secuenciación y basada en RL desarrollada por el grupo de investigación.
- Realizar estudios para comprobar la efectividad del algoritmo en instancias de las que se conozcan los óptimos del *tardiness*.

BIBLIOGRAFÍA

BAGCHI, T. P. 1999. *Multiobjective scheduling by genetic algorithms*, Springer.

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,* 41**,** 1069-1072.

BRUCKER, P., HURINK, J., JURISCH, B. & WGSTMANN, B. 1997. A branch & bound algorithm for the open-shop problem *Elsevier Science. Discrete Applied Mathematics* 76**,** 43-59.

COELLO, C. 2006. Algoritmos Evolutivos Multiobjetivo: Resultados Recientes y Problemas Abiertos. *Evolutionary Computation Group (EVOCINV), departamento de Ingeniería Eléctrica. México, DF,* 7360.

CZYŻAK, P. & JASZKIEWICZ, A. 1997. Pareto simulated annealing. *Multiple Criteria Decision Making.* Springer.

DOWSLAND, K. A. & ADENSO-DÍAZ, B. 2003. Diseño de Heurísticas y Fundamentos del Recocido Simulado. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial,* 7**,** 93-102.

ESTÉVEZ, P. 2007. Optimización mediante Algoritmos Genéticos.

FISHER, H. & THOMPSON, G. L. 1963. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial scheduling***,** 225-251.

FONSECA, C. M. & FLEMING, P. J. Genetic Algorithms for Multiobjective Optimization: FormulationDiscussion and Generalization. ICGA, 1993. 416-423.

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

GABEL, T. & RIEDMILLER, M. On a Successful Application of Multi-Agent Reinforcement Learning to Operations Research Benchmarks. IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007.

GAREN, J. Multiobjective Job-Shop Scheduling With Genetic Algorithms Using a New Representation and Standard Uniform Crossover. MOMH Workshop, Paris, 2002.

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

GARRIDO, A., SALIDO, M. A., BARBER, F. & LÓPEZ, M. Heuristic methods for solving job-shop scheduling problems. ECAI-2000 Workshop on New Results in Planning, Scheduling and Design, Berlín, 2000. 36-43.

GONZÁLEZ, M. 2011. *Soluciones Metaheurísticas al Job-Shop Scheduling Problem with Sequence-Dependent Setup Times*. Universidad de Oviedo.

GRAHAM, R., LAWLER, E. L., LENSTRA, J. K. & KAN, A. H. G. R. 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics,* 5**,** 287-326.

HANSEN, M. P. Tabu search for multiobjective optimization: MOTS. Proceedings of the 13th International Conference on Multiple Criteria Decision Making, 1997. Citeseer, 574-586.

HORN, J., NAFPLIOTIS, N. & GOLDBERG, D. E. A niched Pareto genetic algorithm for multiobjective optimization. Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on, 1994. Ieee, 82-87.

INGOLOTTI, L. 2007. *Modelos y métodos para la optimización y eficiencia de la programación de horarios ferroviarios.* PhD thesis, Universidad Politécnica de Valencia.

ISHIBUCHI, H., YOSHIDA, T. & MURATA, T. 2003. Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *Evolutionary Computation, IEEE Transactions on,* 7**,** 204-223.

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

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

KAMINKA, G. A. 2004. Multi-Agent Systems. *Encyclopedia of Human-Computer Interaction.* Berkshire Publishing.

KNOWLES, J. & CORNE, D. On metrics for comparing nondominated sets. Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on, 2002. IEEE, 711-716.

KNOWLES, J. D. 2002. *Local-search and hybrid evolutionary algorithms for Pareto optimization.* Department of Computer Science Local-Search and Hybrid Evolutionary Algorithms for Pareto OptimiZation Joshua D. Knowles Submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science. Department of Computer Science, University of Reading.

KNOWLES, J. D. & CORNE, D. W. 2000a. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary computation,* 8**,** 149-172.

KNOWLES, J. D. & CORNE, D. W. M-PAES: A memetic algorithm for multiobjective optimization. Evolutionary Computation, 2000. Proceedings of the 2000 Congress on, 2000b. IEEE, 325-332.

KUMAR, R. & ROCKETT, P. 2002. Improved sampling of the Pareto-front in multiobjective genetic optimizations by steady-state evolution: a Pareto converging genetic algorithm. *Evolutionary computation,* 10**,** 283-314.

LAGUNA, M. 1994. A guide to implementing tabu search. *Investigación Operativa,* 4**,** 5-25.

LAWRENCE, S. 1984. Supplement to resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. *Pittsburgh, PA: GSIA, Carnegie Mellon University*.

MARIANO, C. & MORALES, E. 1999a. MOAQ a Distributed Reinforcement Learning Algorithm for the Solution of Multiple Objectives Optimization Problems. *Memorias del Segundo Encuentro Nacional de Computaci on ENC99, paper***,** 12-15.

MARIANO, C. & MORALES, E. 2000. A new distributed reinforcement learning algorithm for multiple objective optimization problems. *Advances in Artificial Intelligence.* Springer.

MARIANO, C. E. & MORALES, E. MOAQ: An Ant-Q algorithm for multiple objective optimization problems. Proceedings of the Genetic and Evolutionary Computation Conference, 1999b. 894-901.

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

MENDEZ, C., CERDA, J., GROSSMANN, I., HARJUNKOSKI, I. & FAHL, M. 2006. State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers & Chemical Engineering,* 30**,** 913-946.

PINEDO, M. 2008. *Scheduling: theory, algorithms, and systems, Third Edition*, Springer Verlag.

RIVERA, D. C. & ELÉCTRICA, I. 2004. Un Sistema Inmune Artificial para resolver el problema del Job Shop Scheduling. *Centro de investigación y estudios avanzados del instituto politécnico nacional. Departamento de ingeniería eléctrica sección de computación*.

RUIZ, R., ŞERIFOĞLU, F. S. & URLINGS, T. 2008. Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research,* 35**,** 1151-1175.

SCHAFFER, J. D. Multiple objective optimization with vector evaluated genetic algorithms. Proceedings of the 1st international Conference on Genetic Algorithms, 1985. L. Erlbaum Associates Inc., 93-100.

SHEN, W. 2002. Distributed Manufacturing Scheduling Using Intelligent Agents. *IEEE Intelligent Systems,* 17**,** 88-94.

SIERRA SÁNCHEZ, M. R. 2009. Mejora de algoritmos de búsqueda heurística mediante poda por dominancia. Aplicación a problemas de scheduling.

SILVA, J. L. & BURKE, E. 2004. A tutorial on multiobjective metaheuristics for scheduling and timetabling. *Multiple Objective Meta-Heuristics*.

SURESH, R. & MOHANASUNDARAM, K. 2006. Pareto archived simulated annealing for job shop scheduling with multiple objectives. *The International Journal of Advanced Manufacturing Technology,* 29**,** 184-196.

SYCARA, K. P. 1998. Multiagent Systems. *AI Magazine***,** 79-92.

VAN VELDHUIZEN, D. A. & LAMONT, G. B. Evolutionary computation and convergence to a pareto front. Late breaking papers at the genetic programming 1998 conference, 1998. Citeseer, 221-228.

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

ZHANG, H. 1995. Applications of Neural Networks in Manufacturing: A state-of-the-art Survey. *International Journal of Production Research,* 33**,** 705-728.

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

ZITZLER, E., DEB, K. & THIELE, L. 2000. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation,* 8**,** 173-195.

ZITZLER, E. & THIELE, L. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *Evolutionary Computation, IEEE Transactions on,* 3**,** 257-271.