UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS

FACULTAD DE MATEMÁTICA, FÍSICA Y COMPUTACIÓN

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

Solución al problema de secuenciación en máquinas paralelas utilizando Aprendizaje Reforzado

Tesis presentada en opción al título de Master en Ciencia de la Computación

Autor: Lic. Juliett M. Suárez Ferreira

Tutor: MSc. Yailen Martínez Jiménez

Santa Clara, 2010

Resumen

Los problemas de Optimización Combinatoria ocupan diversos campos como la economía, el comercio, la ingeniería, la industria o la medicina. Los problemas de secuenciación, como ejemplo de estos, consisten en la localización o asignación de recursos en el tiempo a un conjunto de tareas o actividades; dentro de ellos, aparece el problema de secuenciación en máquinas paralelas, caracterizado como la actividad de asignar un número de trabajos a un conjunto de máquinas con el objetivo de minimizar el costo total de procesamiento.

Recientemente, el aprendizaje reforzado ha sido utilizado para resolver problemas de secuenciación reportando buenos resultados, pero su uso no se ha extendido a problemas de secuenciación complejos donde existen distintos tipos de recursos disponibles. Considerando lo anterior, en este trabajo se presenta una alternativa de solución al problema de secuenciación en máquinas paralelas (JSSP-PM) utilizando el algoritmo *Q-Learning* del Aprendizaje Reforzado. En esta propuesta se realizan adaptaciones al algoritmo original para ajustarlo a las características del problema y se introducen dos enfoques para dar solución al mismo.

Finalmente, el estudio experimental se realiza utilizando instancias del problema que se encuentran disponibles en la librería de investigación de operaciones. Los resultados obtenidos por las variantes propuestas son comparados entre ellos y además con los reportados por otros enfoques.

Abstract

Combinatorial Optimization problems are present in different fields: economy, commerce, engineering, industry or medicine. Scheduling problems, for example, consist in the allocation of resources to jobs or activities; one of these scheduling problems is the Job Shop Scheduling Problem with Parallel Machines (JSSP-PM), which can be characterized as the activity of assigning a number of jobs to a set of machines such that the total processing time is minimized.

Recently, Reinforcement Learning has been applied in the solution of scheduling problems reporting very good results, but it has not been applied to more complex scheduling problems, where different types of resources are available. Considering this, the present work introduces an alternative to solve the Job Shop Scheduling Problem with Parallel Machines using the Q-Learning algorithm from the Reinforcement Learning field. In this proposal, some adaptations to the original algorithm are made in order to adapt it to the characteristics of the problem and two approaches are introduced in order to solve it.

Finally, the experimental study is developed using some instances of the problem available in the Operational Research Library. The results obtained by the alternatives proposed are compared between them and also with the results reported by some other approaches.

Tabla de contenidos

Lista de Figuras

Lista de Tablas

Introducción

La Optimización es la acción y el efecto de buscar la mejor manera de realizar una actividad. Un problema de optimización trata entonces de tomar una decisión óptima para maximizar (ganancias, velocidad, eficiencia) o minimizar (costos, tiempo, riesgo, error) un criterio determinado, teniendo en cuenta ciertas restricciones.

En términos matemáticos, un problema de optimización involucra un conjunto de variables de decisión y una función objetivo que representa o mide la calidad de las decisiones en un cierto dominio X. La Optimización Combinatoria, por su parte, es la rama de la Optimización que afronta los problemas con un dominio X finito, conocidos como problemas combinatorios.

Existen problemas complejos de Optimización Combinatoria en diversos campos como la economía, el comercio, la ingeniería, la industria o la medicina. Un ejemplo de esto son los problemas de secuenciación de tareas (*scheduling*), que consisten en la asignación óptima de recursos en el tiempo a un conjunto de tareas o actividades.

Los problemas de Optimización Combinatoria tienen como característica recurrente que son fáciles de entender y de enunciar, pero generalmente son difíciles de resolver [1].

A diario aparecen nuevos problemas de este tipo, lo que ha dado lugar a que se hayan realizado diversas propuestas de métodos o algoritmos para tratar de solucionarlos. El aprendizaje automático (*Machine Learning*) en una disciplina científica que se encarga del diseño y desarrollo de algoritmos que permiten a las computadoras aprender basadas en datos. Dentro de los algoritmos del aprendizaje automático se encuentran los pertenecientes al Aprendizaje Reforzado (*Reinforcement Learning*) que aprenden cómo actuar dada una situación determinada. Cada acción tomada causará un impacto en el ambiente y el ambiente enviará una señal de refuerzo en forma de recompensa que guiará el aprendizaje del algoritmo.

El Aprendizaje Reforzado significa aprender qué hacer con el objetivo de maximizar una señal de recompensa. Al aprendiz no se le dice qué acción debe tomar, como en casi todas las formas de aprendizaje automático, sino que debe descubrir qué acciones producen la mayor recompensa, probándolas. En los casos más interesantes las acciones afectan no sólo la recompensa inmediata, sino también la próxima situación y a través de esto las subsecuentes recompensas [2].

Dentro de los algoritmos del aprendizaje reforzado pueden encontrarse: TD (λ), SARSA (*State-Action-Reward-State-Action*),) y el *Q-Learning*.

Estos algoritmos asumen que el problema a resolver puede ser tratado como un *Markov Decision Process* (MDP) en el que existen un conjunto de estados $S = \{s_1, s_2, \ldots, s_n\}$. En cada paso un agente percibe el ambiente que lo rodea en el estado s_t y selecciona una acción a_t que produce una transición al estado s_{t+1}. El ambiente emite una recompensa numérica r_{t+1} y el objetivo del agente es maximizar las recompensas que recibe. Existe una probabilidad asociada a cada par estado-acción $p^{a_t}_{s_t s_{t+1}}$ que refleja la probabilidad de ir a s_{t+1} estando en s_t como resultado de tomar la acción a_t [3] [4].

El Aprendizaje Reforzado es utilizado en la solución de una gran variedad de problemas fundamentalmente en la robótica [5] [6] y los juegos [7] [8] [9].

El problema de secuenciación de tareas (*Job Shop Scheduling Problem - JSSP*), ha sido resuelto por el Aprendizaje Reforzado [10]. Este problema involucra un conjunto de trabajos y un conjunto de máquinas con el propósito de encontrar la mejor planificación, es decir, la localización de las operaciones en intervalos de tiempo en las máquinas que tenga la mínima duración para completar todos los trabajos. La suma total de las soluciones posibles para un problema con n trabajos y m máquinas es (n!)m.

En un enfoque propuesto por Gabel y Riedmiller [10] se analiza el uso de las técnicas del Aprendizaje Reforzado para resolver el *JSSP*, en específico el algoritmo *Q-Learning*. Se demuestra que interpretar y resolver las tareas del *JSSP* como un problema de aprendizaje con múltiples agentes es beneficioso para obtener soluciones cercanas a las óptimas y que bien pueden competir con soluciones mostradas por otros enfoques encontrados en la literatura que dan solución al problema [11]. En [12] se muestra una solución al *JSSP* utilizando Q-Learning basada en las ideas anteriores pero con algunas modificaciones que mejoran el comportamiento del algoritmo.

El problema de secuenciación en máquinas paralelas (*Job Shop Scheduling Problem with parallel machines - JSSP-PM*) representa un problema importante en la práctica actual. El mismo consiste en la asignación de las operaciones de cada trabajo a un recurso dentro de un conjunto de máquinas paralelas candidatas (subproblema de asignación); en adición al *JSSP* clásico donde las operaciones deben asignarse a cada recurso existente con el objetivo de minimizar el tiempo total de producción,

denominado *makespan* (subproblema de secuenciación). La existencia de máquinas paralelas candidatas (múltiples tipos de máquinas) permite aumentar el rendimiento y evitar que la producción se detenga cuando una máquina falla o se le hace mantenimiento.

La literatura acerca del *JSSP-PM* no es abundante. En un enfoque propuesto en [13] se propone un método híbrido entre Algoritmos Genéticos (AG) y la metaheurística Optimización Basada en colonias de Hormigas (ACO) como vía de solución del problema. En [14] se soluciona el problema utilizando ACO y en [15] se propone un enfoque utiliza los AG.

El contar con un conjunto de máquinas de un mismo tipo para realizar la misma labor, implica que una vez terminada una operación tiene que decidirse cuál operación va a ser la próxima en realizarse y cuál de las máquinas candidatas la ejecutará, lo que aumenta las posibilidades de elección.

Hasta el momento no se ha encontrado una aplicación del Aprendizaje Reforzado en la solución del *JSSP-PM* sin embargo, las características de este problema (en cada momento se tiene una visión del ambiente y se debe elegir que acción tomar de todas las posibles que permita cambiar de estado con el objetivo de minimizar el tiempo de producción total) pueden resultar adecuadas para que sea resuelto mediante esta técnica. De aquí se deriva el problema científico siguiente: ¿cómo solucionar el problema de secuenciación en máquinas paralelas utilizando el Aprendizaje Reforzado?

Constituye el ***objetivo general*** de esta investigación:

Solucionar el problema de secuenciación en máquinas paralelas (*JSSP-PM*) utilizando el algoritmo *Q-Learning* del Aprendizaje Reforzado, de manera que los resultados que se obtengan sean similares a los reportados por otros enfoques.

Este objetivo general fue desglosado en los ***objetivos específicos*** siguientes:

- Proponer una adaptación del algoritmo *Q-Learning* para resolver el problema *JSSP-PM*.
- Realizar la implementación computacional del algoritmo *Q-Learning* para solucionar el problema planteado.
- Establecer comparaciones estadísticas entre los resultados que brindan las variantes implementadas de *Q-Learning*.

- Evaluar los resultados obtenidos a partir de comparaciones estadísticas con la solución al problema aportada por los algoritmos ACO, AG, y el híbrido AG – ACO que se encontraron

Hipótesis de investigación

Las características del *JSSP-PM* hacen factible la aplicación del aprendizaje reforzado para la solución del mismo. La aplicación del *Q-Learning* en la solución de este problema permite que los resultados obtenidos sean similares o superiores a los reportados en la literatura.

El estudio del *Q-Learning* y la factibilidad de su aplicación para encontrar una solución al problema *JSSP-PM*, adquiere un significado relevante en la vida práctica. La secuenciación de trabajos juega un papel particularmente importante en el contexto de la industria. Además de esta perspectiva industrial, esta caracterización tiene otras interpretaciones; trabajos y máquinas pueden entenderse como programas y computadoras, clases y profesores, misiones militares y soldados, o pacientes y equipamiento de hospital.

El presente informe incluye, además de esta introducción, tres capítulos, conclusiones, recomendaciones, bibliografía y anexos. En el Capítulo I aparecen algunas consideraciones de carácter teórico sobre los problemas de secuenciación de tareas (*scheduling*), en particular el *JSSP-PM.* Se hace referencia al Aprendizaje Reforzado con una forma de solucionar este problema. El Capítulo II dedica su espacio al tratamiento de los aspectos generales relacionados con el *Q-Learning* y la forma de aplicarlo al *JSSP-PM*. En el Capítulo III se ofrecen las valoraciones acerca de los resultados obtenidos por la aplicación del algoritmo *Q-Learning*, así como la comparación con otros métodos que solucionan el *JSSP-PM.*

Capítulo 1: Problemas de secuenciación de tareas y Aprendizaje Reforzado

Optimizar significa mejorar, perfeccionar; sin embargo, en el contexto científico la optimización es el proceso de tratar de encontrar la mejor solución posible para un determinado problema.

El capítulo inicial de este trabajo aborda aspectos esenciales de los problemas de secuenciación de tareas, reconocidos problemas de Optimización Combinatoria que tienen la meta de maximizar o minimizar un objetivo sobre un conjunto finito de soluciones.

Reconociendo que el problema de secuenciación de tareas en máquinas paralelas (*Job Shop Scheduling Problem with Parallel Machines*, en lo adelante *JSSP-PM*) se encuentra dentro de esta categoría, se investiga acerca de su definición y se ofrece un ejemplo general de este problema.

También se profundiza en el Aprendizaje Reforzado como una forma factible para resolver este problema y se propone el algoritmo *Q-Learning* como variante de solución al problema planteado.

1.1 Secuenciación de Tareas

La creciente complejidad de los procesos de manufactura y la competencia en el mercado ha obligado a las empresas a optimizar sus operaciones tanto como les sea posible. En particular, una secuenciación óptima de las órdenes de producción en recursos limitados es una cuestión importante, este tipo de optimización ha fascinado a los investigadores durante décadas [16].

En la mayoría de las publicaciones de este campo, la secuenciación es estudiada como un problema independiente. Sin embargo, en la práctica, está mezclada con las decisiones hechas a otros niveles de la empresa, por ejemplo: planificación de la producción a largo plazo y horario de la fuerza de trabajo.

La palabra planificación (*planning*) es usada frecuentemente como un término bastante general, incluyendo entre otras cosas los problemas de secuenciación. Pero es bastante común hacer una ligera distinción entre ellas, el objetivo de la planificación es “qué hacer”, mientras que el objetivo de la secuenciación es “cómo hacerlo”. Luego de esta

definición, podemos decir que los problemas de planificación están en un nivel más general y su solución es, usualmente la entrada de uno o más problemas de secuenciación que tiene en cuenta muchos más detalles [17].

Los problemas de secuenciación de tareas han capturado el interés de un gran número de investigadores de múltiples campos de investigación: ingeniería industrial, investigación de operaciones e inteligencia artificial. La secuenciación de tareas en la fabricación es un proceso de optimización que asigna recursos limitados en el tiempo entre actividades de fabricación paralelas o secuenciales. Esta asignación debe obedecer a un conjunto de restricciones que reflejan las relaciones temporales entre las actividades y las limitaciones en capacidad de un conjunto de recursos compartidos. La asignación afecta también la calidad de la secuenciación con respecto a criterios como el costo, la tardanza o el rendimiento.

Los problemas de optimización no son exclusivos de la producción industrial. Situaciones similares ocurren en universidades, hospitales, aeropuertos, compañías de transportación, etc. Estos problemas son típicamente NP-completos [1] y el tiempo computacional de solucionarlos incrementa exponencialmente con el tamaño del problema.

1.1.1 Clasificación de los problemas de secuenciación

La primera distinción a hacer es entre los problemas de secuenciación **deterministas** y **estocásticos**. En la clase determinista, todos los parámetros son conocidos sin incertidumbre, mientras que en los problemas de secuenciación estocásticos los tiempos de procesamiento de las tareas son modelados como variables aleatorias.

Otra clasificación son los problemas de secuenciación **estáticos** y **dinámicos**. En los problemas de secuenciación estáticos, todos los trabajos (actividades) que serán procesados son conocidos. En los problemas de secuenciación dinámicos, nuevos trabajos pueden adicionarse durante el tiempo de procesamiento. Es importante resaltar que un problema dinámico es estocástico por naturaleza, pero que un problema sea estático no implica que sea determinista.

1.1.2 Tipos de soluciones

Acorde a las propiedades de la secuenciación, una solución a un problema de secuenciación puede ser categorizada en cuatro tipos:

- Inadmisible
- Semi-activa
- Activa
- Sin retraso (*Non-delay*)

Está claro que existen infinitas soluciones para un problema de secuenciación, tiempos ociosos pueden ser insertados en una secuencia de trabajos de infinitas formas y el resultado sigue siendo una secuencia válida de trabajos, es decir, un arreglo de operaciones que satisfacen todas las condiciones del proceso de secuenciación.

Considere el conjunto infinito de todas las posibles secuenciaciones que posee exactamente el mismo orden de las operaciones en cada máquina. Estas secuenciaciones difieren solamente en la cantidad de tiempo ocioso que es insertado entre las operaciones. Dentro de este conjunto existe una secuenciación única nombrada secuenciación semi-activa que domina todas las otras secuenciaciones en el conjunto para cualquier medida de desempeño. Todas las otras secuenciaciones en el conjunto son llamadas inadmisibles (el número de planificaciones inadmisibles es infinito y la mayoría de ellos contiene tiempos ociosos excesivos).

La secuenciación semi-activa puede ser obtenida desde cualquier secuenciación mediante ejecuciones repetidas de una operación llamada “corrimiento a la izquierda limitado” (*limited-left-shift*). Esto es mover cada operación de un trabajo a la izquierda tanto como sea posible, es decir, hasta que sea bloqueada por la operación precedente en la máquina o la operación precedente del trabajo, pero el orden de las operaciones no puede ser alterado. En una secuenciación semi-activa ninguna operación puede ser movida por la operación “*limited-left-shift*”.

Por otro lado, las secuenciaciones activas son aquellas en las que no es posible ejecutar un “corrimiento a la izquierda” (*left-shift*) en ninguna operación. Un “*left-shift*” es cualquier decremento en el tiempo en el que la operación comienza que no requiere un incremento en el tiempo de inicio de cualquier otra operación [18]. A diferencia del corrimiento a la izquierda limitado, este corrimiento permite a una operación “saltar” sobre otra hacia un intervalo de tiempo ocioso, si ese intervalo de tiempo ocioso es suficiente para acomodar la operación que va a ser movida.

La Figura 1, basada en los datos de la Tabla 1, muestra las diferencias entre los distintos tipos de soluciones. En dicha figura encontramos el Trabajo 1 en blanco y el Trabajo 2 en negro.

Tabla 1: Ejemplo con dos trabajos y dos máquinas

Trabajos	Operación 1	Operación 2
1	M_2 / 2	M_1 / 1
2	M_1 / 2	M_2 / 1

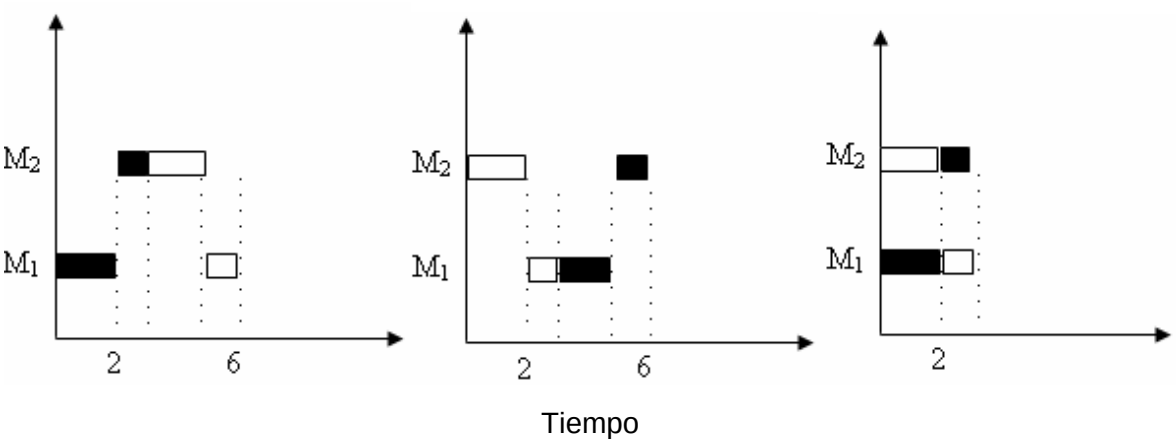

Figura 1: Tres planificaciones semi-activas

Las secuenciaciones en la Figura 1 son todas semi-activas, ya que, en ninguno de los casos las operaciones pueden ser desplazadas a la izquierda a través de un corrimiento limitado. No obstante, la secuenciación 1 (a la izquierda) no es activa debido a que la primera operación del trabajo 1 puede moverse al tiempo ocioso inicial de la máquina 2 sin postergar el comienzo de la segunda operación del Trabajo 2 (aplicando *left-shift*). Una vez hecho esto, la segunda operación del Trabajo 1 puede desplazarse a la izquierda (*limited-left-shift*) obteniéndose como resultado la tercera secuenciación mostrada (a la derecha de la Figura 1).

Las secuenciaciones sin retardo (*non-delay*) son aquellas en las que una máquina nunca puede estar ociosa si alguna operación está habilitada para ser ejecutada. Este tipo de secuenciación es por definición una secuenciación activa. Es importante destacar que la mejor secuenciación no es necesariamente una sin retraso. Es más fácil

de generar una secuenciación sin retraso que una activa. Existe evidencia de que las secuenciaciones sin retraso llevan a soluciones cuya calidad media es superior a aquellas producidas por planificaciones activas. No obstante, los algoritmos de secuenciación típicos indagan en el espacio de todas las planificaciones activas con el objetivo de asegurar que el óptimo sea tomado en consideración [11].

Figura 2: Jerarquía de las planificaciones

1.1.3 Métodos Constructivos vs. Métodos de reparación

Las metodologías de secuenciación pueden ser clasificadas además acorde a si las soluciones son encontradas constructivamente o reparando iterativamente una solución completa. Los métodos constructivos extienden una solución parcial hasta que cada tarea queda reflejada en la secuencia construida. Los métodos de reparación modifican iterativamente una solución completa para eliminar conflictos y optimizar la solución inicial [19].

La mayoría de los métodos de optimización y los métodos de búsqueda orientados o dirigidos por restricciones siguen el enfoque constructivo. Estos métodos han sido probados con éxito para una gran variedad de clases de problemas de secuenciación.

1.2 Problema de secuenciación de tareas

Uno de los problemas de secuenciación más conocidos es el de secuenciación de tareas en inglés *Job Shop Sheduling Problem* (*JSSP*), que involucra un conjunto de trabajos y un conjunto de máquinas con el propósito de encontrar la mejor secuenciación que es una asignación de las operaciones en intervalos de tiempo en las máquinas de forma que se terminen todos los trabajos en el menor tiempo posible. La

suma de todas las posibles soluciones para un problema con n trabajos y m máquinas es (n!)m. En este caso, los métodos de optimización fallan al no ofrecer soluciones rápidas. Es por eso que se gira la atención en encontrar métodos que puedan producir soluciones satisfactorias aunque no necesariamente óptimas.

El problema de secuenciación de tareas constituye un desafío. Si bien es fácil de declarar, y de visualizar lo que se requiere, es extremadamente difícil progresar hacia la solución, de cualquier forma el problema sigue atrayendo a los investigadores.

Enfoques clásicos como Ramas y Cotas [20] han sido utilizados para solucionar el problema de secuenciación. Además, existen varios procedimientos de búsqueda local que han resuelto este tipo de problemas, por ejemplo, Recocido Simulado[21], o Búsqueda Tabú[22].

La terminología de secuenciación de tareas surge en la industria de la manufactura. En la teoría fundamental de la secuenciación [23] [24] [25] [26] los problemas se definen con la siguiente formulación matemática: dado un conjunto M de m máquinas que pueden procesar un solo trabajo a la vez, y dado un conjunto T de n trabajos que deben ser procesados en cada una de las máquinas anteriores con un orden predefinido, es necesario encontrar una secuencia que minimice el tiempo total de procesamiento. Cada trabajo t_i consiste de m operaciones encadenadas $o_{i1}, o_{i2}, \ldots, o_{im}$ que tienen que ser realizadas en un orden preestablecido (restricciones de orden) [19].

Hay un total de $N = n * m$ operaciones, donde o_{ik} es la operación correspondiente al trabajo t_i que tiene que ser ejecutada en la máquina m_k durante un intervalo de tiempo ininterrumpido p_{ik}. El flujo de las operaciones de cada trabajo a través de las máquinas es independiente del de otros trabajos. Cada máquina es capaz de procesar solo un trabajo a la vez, y un trabajo solo puede estar siendo procesado por solo una máquina simultáneamente (restricciones de capacidad). El parámetro $C_{\max}$ (*makespan*)[17] señala la medida del comportamiento que debe ser minimizada (el tiempo para completar todos los trabajos).

El objetivo es determinar los tiempos de comienzo ($p_{ik} \geq 0$) de cada operación de forma tal que se minimice el tiempo total de trabajo y se satisfagan todas las restricciones:

$$C^{*}_{\max} = \min\{ C_{\max} \} = \min_{soluciones\ factibles} \{\max\{ t_{ik} + p_{ik} \} : \forall \mathrm{J}_{\mathrm{i}} \in \mathrm{J}, \forall \mathrm{M}_{\mathrm{k}} \in M \}$$

El *Job Shop Scheduling* es un problema de optimización combinatoria que cuenta con las siguientes restricciones:

- No se pueden procesar dos operaciones del mismo trabajo simultáneamente.
- Ningún trabajo se procesa dos veces en la misma máquina.
- No está permitido el intercambio en el orden de las operaciones.
- Cada trabajo debe ser procesado hasta terminarse.
- Los trabajos deben esperar que la próxima máquina en su orden de procesamiento esté disponible.
- Los trabajos pueden comenzar a procesarse y terminar su procesamiento en cualquier instante de tiempo.
- Ninguna máquina puede procesar más de una operación a la vez.
- Existe solo una máquina de cada tipo.
- Las máquinas pueden estar inactivas durante el proceso de secuenciación.
- El orden de procesamiento de cada trabajo es conocido de antemano y es inviolable.

Para los investigadores es un hecho que el *JSSP* es uno de los problemas NP-completos más complejos y posee consecuentemente un constante reto intelectual: 40 años de esfuerzo, resultados publicados en cientos de artículos de revistas, disertaciones, y aún los algoritmos del estado del arte fallan frecuentemente en encontrar soluciones óptimas a instancias relativamente pequeñas del problema [17].

1.3 Problema de Secuenciación en Máquinas Paralelas (JSSP-PM)

El *JSSP-PM* es un problema de optimización combinatoria que se puede describir de la siguiente forma:

Dado m tipos de máquinas, cada uno de los cuales está representado por k máquinas que pueden procesar sólo un trabajo a la vez (nótese que el caso en que $k = 1$ representa el *JSSP* clásico), y dado un conjunto de n trabajos que deben ser procesados en cada una de estas máquinas en un orden determinado, encontrar una secuencia factible que minimice el tiempo de procesamiento, es decir, el tiempo necesario para completar todos los trabajos.

Se listan a continuación las restricciones del *JSSP-PM*:

- Cada trabajo esta formado por un conjunto de operaciones con un orden específico.
- Las operaciones que forman al trabajo están compuestas por el tipo de máquina en la que se van a procesar y el tiempo que demora su procesamiento.
- El orden de procesamiento de las operaciones de cada trabajo es conocido de antemano y es inviolable.
- La ejecución de una operación en una máquina no puede ser interrumpida.
- No se pueden procesar dos operaciones del mismo trabajo simultáneamente.
- Una vez que un trabajo culmine la ejecución de una operación, debe esperar a que exista al menos una máquina disponible del tipo correspondiente a la próxima operación que debe ejecutar.
- Ningún trabajo se procesa dos veces en el mismo tipo de máquina ni en la misma máquina.
- Ninguna máquina puede realizar más de una operación a la vez.
- Las máquinas se demoran en el procesamiento de los trabajos el tiempo que especifica la operación que realizan.
- Las máquinas pueden estar inactivas durante el proceso de secuenciación.

- Cada trabajo debe ser procesado hasta terminarse.
- Un trabajo se considera terminado cuando ha realizado todas las operaciones.

Como se puede apreciar, las tareas juegan el rol fundamental, se comportan como entidades inteligentes que conocen el tiempo necesario para su ejecución y la forma en la que se ejecutan, dejando a los recursos sólo como el medio que les facilita su terminación.

1.3.1 Definición matemática.

La descripción del problema de secuenciación en máquinas paralelas presentada en el epígrafe anterior puede formalizarse matemáticamente como sigue:

Se tiene un conjunto $M = \{M_{l,j}\}_{l=1\ j=1}^{k,m}$ de máquinas con $l = 1,\ldots,k$ y $j = 1,\ldots,m$ donde m representa los tipos de máquinas que intervienen en el problema y k es la cantidad de máquinas que existen de cada tipo. En la Figura 3 se observa una distribución de m tipos de máquinas, con k máquinas de cada tipo.

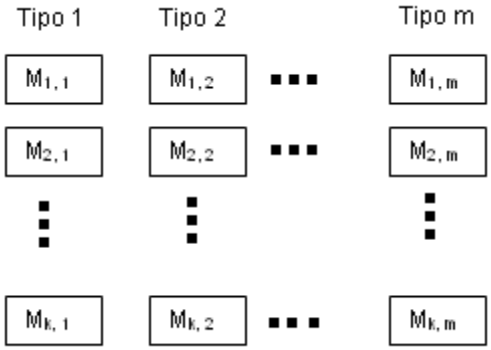

Figura 3: Ejemplo de distribución de *m* tipos de máquinas, *k* de cada tipo

Se define un conjunto de trabajos $J = \{J_i\}_{i=1}^{n}$ con $i = 1,\ldots,n$ y un conjunto de operaciones $O = \{O_{i,j}\}_{i=1,\ j=1}^{n,m}$ donde $O_{i,j}$ es la operación $j-ésima$ del trabajo i, destacando que el número de operaciones que tiene que realizar un trabajo es igual a la cantidad de tipos de máquinas que involucra el problema, en este caso m.

Cada trabajo J_i tiene una secuencia de operaciones $O_{i,j}$ encadenadas que tienen que ser realizadas en el orden especificado y que están formadas por un par

$(tipo\ de\ máquina\ ,\ tiempo\ )$, que también se conoce de antemano, donde $tipo\ de\ máquina$ especifica el tipo de la máquina que debe realizar la operación y $tiempo$ es el tiempo que demora en realizarse la misma. Cada operación puede ser ejecutada en cualquiera de las k máquinas pertenecientes al tipo de máquina especificado. Denominemos $(m_{i,j}, p_{i,j})$ al par anteriormente descrito.

El objetivo es encontrar un tiempo de inicio $t_{i,j}$ para cada operación de forma tal que se minimice el tiempo de procesamiento de los trabajos ($C_{\max}$) denominado *makespan.*

$$\min\{\ C_{\max}\ \} = \min\ \{\max\{\ t_{ij} + p_{ij}\} : \forall J_i \in J\ \forall M_{lj} \in M\ \}$$

Este makespan es factible si se cumplen las siguientes restricciones:

i. $t_{i,j} \geq 0$ para todo $O_{i,j} \in O$

ii. $t_{i,h} \geq t_{i,j} + p_{i,j}$ para todo $O_{i,j}, O_{i,h} \in O \quad con \quad O_{i,j} \prec O_{i,h}$

iii. $t_{i,j} \geq t_{c,h} + p_{c,h}\ o\ t_{c,h} \geq t_{i,j} + p_{j,j}$ para todo $O_{i,j}, O_{c,h} \in O \quad con \quad m_{i,j} = m_{c,h}$

La restricción i) implica que ninguna máquina está disponible antes del instante 0.

La restricción ii) da cuenta de la relación de precedencia $\prec$. Esta relación refleja el hecho de que un trabajo se compone de operaciones que tienen que efectuarse en un orden preciso y no en otro. $\prec$ es una relación de precedencia fijada por el problema, entre operaciones que pertenezcan al mismo trabajo, tal que, si $O_{i,j} \prec O_{i,h}$ entonces $O_{i,h}$ no puede comenzar antes que $O_{i,j}$ finalice.

Finalmente la restricción iii) implica que ninguna máquina puede procesar dos operaciones al mismo tiempo, es decir, si dos operaciones utilizan la misma máquina, una debe esperar que la otra termine antes de poder empezar a ejecutarse. Para eso, es necesario establecer un orden entre las operaciones que se efectúen en la misma máquina, quedando determinada la secuencia de operaciones en cada una de las máquinas.

1.3.2 Ejemplo

Dada una instancia con tres tipos de máquinas, dos máquinas de cada tipo y seis trabajos donde el objetivo es minimizar el makespan, como se ilustra en la Figura 4.

Las operaciones de los trabajos necesitan ser procesadas en un orden numérico. Tipos de máquinas y tiempos de procesamiento son especificados para cada operación. De la tabla podemos inferir que el $Trabajo_1$ debe ser procesado primero en una máquina de Tipo 1 con tiempo 1, luego debe ser procesado por una máquina de Tipo 2 por 3 unidades de tiempo y luego debe procesarlo una máquina de Tipo 3 durante 5 unidades de tiempo para completar un procesamiento exitoso.

Una forma de ilustrar la solución de un problema de este tipo es la confección de un diagrama de Gantt[1] que también podemos ver en la Figura 4, donde se observa una posible distribución de los seis trabajos en las seis máquinas de las que dispone el ejemplo, dos de cada tipo, obteniéndose 12 como tiempo final de producción.

	Op.1	Op.2	Op.3
Trabajo 1	$M_1/1$	$M_2/3$	$M_3/5$
Trabajo 2	$M_2/3$	$M_1/2$	$M_3/1$
Trabajo 3	$M_1/2$	$M_3/1$	$M_2/4$
Trabajo 4	$M_2/4$	$M_3/2$	$M_1/2$
Trabajo 5	$M_3/2$	$M_2/4$	$M_1/2$
Trabajo 6	$M_3/3$	$M_1/2$	$M_2/1$

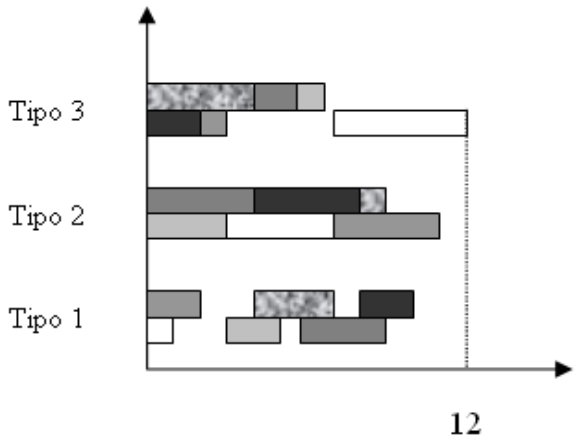

Figura 4: Ejemplo de un problema de secuenciación en máquinas paralelas y su solución

1.4 Instancias del JSSP-PM

Para medir el comportamiento del algoritmo implementado se usarán algunas de las instancias que se proponen en [27], las cuales se basan en las instancias clásicas del

[1] Herramienta gráfica cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

Job Shop Scheduling que se encuentran disponibles en la Librería de Investigación de Operaciones (*OR-Library*) [21].

Las instancias de la *OR-Library* tienen el siguiente formato:

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 numerarse desde cero. En la Figura 5 se observa el ejemplo de la instancia ft06.

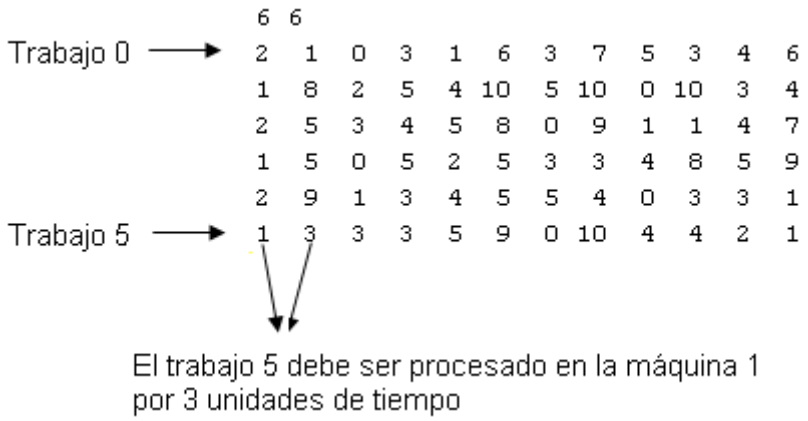

Figura 5: Instancia ft06

En este ejemplo el Trabajo$_0$ debe ser procesado primero en la Máquina$_2$ por 1 unidad de tiempo, luego va a la Máquina$_0$ para ser procesado por 3 unidades de tiempo y así sucesivamente. Es importante destacar que el orden en el que los trabajos deben ser procesados por las máquinas no se puede violar. Por ejemplo, el Trabajo 5 debe ser procesado en el orden: $\{M_1, M_3, M_5, M_0, M_4, M_2\}$ por $\{3,3,9,10,4,1\}$ unidades de tiempo respectivamente.

La Máquina$_2$ debe decidir cuál trabajo va a procesar primero ya que en el primer paso puede elegir entre los trabajos 0, 2 o 4. Esto significa que al seleccionar por ejemplo el Trabajo$_2$, los trabajos 0 y 4 quedarán esperando por algunas unidades de tiempo a que la máquina termine dicho procesamiento, ya que de acuerdo a las restricciones del problema ningún recurso puede procesar más de un trabajo a la vez y además los trabajos no pueden violar las restricciones de orden, por lo que no pueden ser procesados primero en otras máquinas aunque estén disponibles.

La instancia ft06.txt mostrada en la Figura 5 es una de las más estudiadas, para ella el óptimo reportado es $C_{\max} = 55$. Una solución posible tomada de [10] se muestra a continuación:

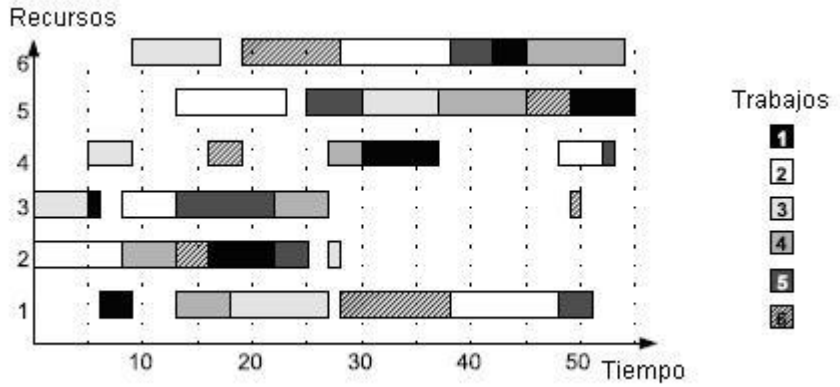

Figura 6 Representación en un diagrama de Gantt de una solución óptima para la instancia ft06

De las instancias anteriores se obtienen nuevas repitiendo k veces cada uno de los trabajos donde k es el número de máquinas asociada a cada uno de los tipos de máquinas que se tendrán en el JSSP-PM. La instancia anterior quedaría como muestra la Figura 7 para un problema con $k = 2$ máquinas de cada tipo.

Número de trabajos Tipos de máquinas

12 6

Trabajo 0 →
2 1 0 3 1 6 3 7 5 3 4 6
1 8 2 5 4 10 5 10 0 10 3 4
2 5 3 4 5 8 0 9 1 1 4 7
1 5 0 5 2 5 3 3 4 8 5 9
2 9 1 3 4 5 5 4 0 3 3 1
1 3 3 3 5 9 0 10 4 4 2 1

Trabajo 6 →
2 1 0 3 1 6 3 7 5 3 4 6
1 8 2 5 4 10 5 10 0 10 3 4
2 5 3 4 5 8 0 9 1 1 4 7
1 5 0 5 2 5 3 3 4 8 5 9
2 9 1 3 4 5 5 4 0 3 3 1
1 3 3 3 5 9 0 10 4 4 2 1

Figura 7: Instancia ft06 duplicada (ft06')

Como se observa en la Figura 7, ahora la cantidad de trabajos es 12 y el 6 representa, en lugar de la cantidad de máquinas, los tipos de máquinas que van a existir,

exactamente $k = 2$ de cada tipo para este ejemplo por lo que en total existirán 12 máquinas.

1.5 Métodos para solucionar problemas de optimización combinatoria.

Existen dos formas fundamentales de encontrar soluciones para los problemas de Optimización Combinatoria: los métodos exactos y los métodos aproximados.

Los métodos exactos son capaces de encontrar la solución óptima a los problemas. Entre estos se encuentran los algoritmos de vuelta atrás (*backtracking*) [28], ramificación y poda (*branch and bound*) [20] y programación dinámica [29].

La principal desventaja de los métodos exactos es que al explorar todos los caminos posibles, el costo computacional en tiempo y espacio es elevado.

Los algoritmos aproximados son la única forma factible de obtener soluciones cercanas a las óptimas con un costo computacional relativamente bajo [30]. Existen métodos aproximados de naturaleza muy diferente, por lo que es complicado dar una clasificación completa. Además, muchos de ellos han sido diseñados para un problema específico sin posibilidad de generalización o aplicación a otros problemas similares.

El Aprendizaje Reforzado es una rama del Aprendizaje Automatizado que ha sido utilizado en la solución de problemas de Optimización Combinatoria, obteniendo buenos resultados específicamente en el JSSP. En un enfoque propuesto por Gabel y Riedmiller [10] se analiza el uso de las técnicas del aprendizaje reforzado para resolver este problema y se demuestra que interpretar y resolver las tareas del *JSSP* como un problema de aprendizaje con múltiples agentes es beneficioso para obtener soluciones cercanas a las óptimas.

1.6 Aprendizaje Reforzado

El Aprendizaje Reforzado (*Reinforcement Learning*, RL) es un enfoque de la Inteligencia Artificial en el que los agentes aprenden a partir de su interacción con el ambiente. Es aprender qué acción tomar dada una situación determinada con el objetivo de maximizar una señal numérica de recompensa que da la medida de cuán buena fue la acción elegida por el agente [31].

En el paradigma del aprendizaje reforzado un agente se conecta a su ambiente mediante la percepción y acción, como lo descrito en la Figura 8. En cada paso de la interacción, el agente percibe el estado actual “s” de su ambiente y entonces selecciona una acción “a” para cambiar este estado. Esta transición genera una señal de refuerzo “r”, que es recibida por el agente. La tarea del agente es aprender una política de elección de acciones en cada estado, que le posibilite recibir un número máximo de recompensas acumuladas. Los métodos del aprendizaje reforzado exploran el ambiente todo el tiempo para obtener una política deseada [19].

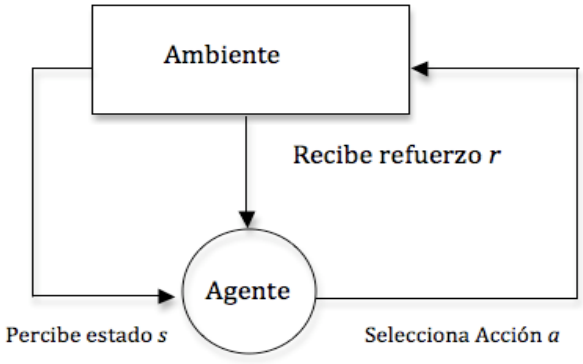

Figura 8 Paradigma del Aprendizaje Reforzado

Un ambiente común posee las características de un *MDP*. En este tipo de ambiente lo que sucederá en el futuro depende únicamente del estado actual del ambiente y la acción que se tomará. Los investigadores del Aprendizaje Reforzado se han enfocado en el aprendizaje en este tipo de ambiente, desarrollando métodos como el *Q-Learning* [32].

Uno de los retos del Aprendizaje Reforzado es establecer un balance entre la exploración y la explotación. Para obtener una mayor recompensa, un agente del aprendizaje reforzado debe preferir acciones que se hayan explorado en el pasado y que sean efectivas en cuanto a la recompensa obtenida; pero para descubrir tales acciones, tiene que probar acciones que no han sido seleccionadas con anterioridad. El agente tiene que explotar lo que ya conoce en el orden de obtener recompensas; pero tiene que explorar también para hacer una mejor selección de acciones en el futuro. El dilema es que ni explotación ni exploración pueden seguirse exclusivamente. El agente debe probar una variedad de acciones y progresivamente favorecer aquellas que parecen mejor. Cada acción debe ser probada varias veces para ganar una estimación

fiable de su recompensa esperada. Un control apropiado del balance entre explotación y exploración es importante para construir métodos eficientes de aprendizaje [2].

1.6.1 Modelo del Aprendizaje Reforzado

En el Aprendizaje Reforzado un **agente** aprende y toma decisiones para tratar de alcanzar una meta determinada a partir de su interacción con el **ambiente**, que es aquello que no es controlado por el agente, genera **estados** y recibe **acciones**. Los estados son la percepción que el agente tiene del ambiente y la acción es la conducta que utiliza el agente para modificar el ambiente.

Formalmente el modelo básico de Aprendizaje Reforzado consiste en:

- Un conjunto de estados del ambiente S ;
- Un conjunto de acciones A ;
- Un conjunto numérico de "recompensas" en $\mathbb{R}$.

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_t , donde $\pi_t(s,a)$ 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.

La **función de recompensa** define la meta en un problema de aprendizaje reforzado. 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 del aprendizaje reforzado 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 del Aprendizaje Reforzado.

Los agentes del Aprendizaje Reforzado pueden trabajar en un ambiente donde no exista un modelo de acciones preestablecido y normalmente no deshacen las acciones que ya tomaron sino que exploran activamente el ambiente para observar el efecto de sus decisiones, por lo que tampoco necesitan conocer con anterioridad cuál de ellas será correcta o incorrecta. En resumen, los métodos del Aprendizaje Reforzado 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.

Una de las áreas donde el Aprendizaje Reforzado ha sido ampliamente utilizado es en el diseño de robots autónomos, siendo el objetivo lograr que el robot tome decisiones efectivas a medida que explora el ambiente en el que se encuentra [33].

La Figura 9 muestra un problema sencillo donde el agente tiene que moverse de casilla en casilla (en este caso los estados), seleccionando entre dos acciones: derecha (D) y abajo (A). El sensor del agente devolverá el valor de cada estado. El agente debe comenzar en el estado a_1 y terminar en el e_5 y recibirá una recompensa en base al valor de cada estado visitado.

El objetivo es aprender una política que permita al agente acumular el mayor número de recompensas. Por ejemplo la secuencia de acciones D-A-D-A-A-D-D-A comenzando por el estado a_1 da como resultado la puntuación óptima: 17.

	a	b	c	d	e
1	0	2	1	-1	1
2	1	1	2	0	2
3	3	-5	4	3	1
4	1	-2	4	1	2
5	1	1	2	1	1

Figura 9 Ejemplo de problema de decisión.

Los problemas de aprendizaje reforzado pueden dividirse en dos tipos fundamentales: Problema de Predicción y Problema de Control [34]. El primero trata de determinar la **función de valor (estado)** máxima para una política dada. Este problema se resuelve utilizando un método de Monte Carlo [35], una combinación de este con Programación Dinámica [29] o la técnica de Diferencias Temporales $TD(0)$.

El problema de control trata de determinar la política óptima por convergencia a la **función de valor (estado-acción)**. Este problema se resuelve utilizando los algoritmos SARSA y Q-Learning [36].

1.6.2 Algoritmos del Aprendizaje Reforzado

Los problemas descritos anteriormente pudieran resolverse utilizando técnicas de Programación Dinámica, pero para lograrlo se necesita un modelo explícito y completo del proceso que se desea controlar basado en la probabilidad de las transiciones.

Otra forma de solucionar estos problemas es a través de los métodos de Monte Carlo los cuales permiten encontrar soluciones aproximadas a una variedad de problemas matemáticos a través de la realización de experimentos de muestreo estadístico. Estos métodos utilizan un modelo 'débil' pero requieren una corrida completa antes que puedan realizarse actualizaciones [37].

El Aprendizaje Reforzado 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.

1.6.2.1 TD

El método de las Diferencias Temporales fue propuesto por Sutton en 1988 [38] basado en el método de iteración de políticas de la Programación Dinámica y provee la facilidad de actualizar el sistema inmediatamente después de haber visitado nuevos estados. Está centrado en las diferencias en un instante de tiempo entre el costo esperado y el costo que se predice al tomar una acción.

Una formulación más general se logra teniendo en cuenta la influencia de las diferencias temporales en los estados futuros del problema. De esta forma surge el algoritmo $TD(\lambda)$ con $0 < \lambda < 1$. El caso $TD(0)$ es equivalente al método *Value Iteration*

de la Programación Dinámica. El caso $TD(1)$ actualiza el valor de la aproximación basado solamente en el valor del estado final.

Esta probado que este método es un mecanismo robusto para evaluar la política de un agente mediante el cálculo de los costos esperados.

1.6.2.2 Q-Learning

Q-Learning es una técnica propuesta por Watkins en 1989 [32] que tiene su origen en el método *Value Iteration* y está basada en la utilización de un valor *q* que representa el costo esperado de seleccionar una acción en un estado determinado y a partir de ahí seguir una política óptima. Una descripción detallada del algoritmo será dada en el Capítulo 2.

Este método es uno de los más populares del Aprendizaje Reforzado debido a que fue el primero en surgir para resolver los problemas de control, existe una prueba de la convergencia del método en [39] y en [40] y además es la extensión por excelencia del concepto de Control Óptimo[2], en el sentido que es una técnica simple que calcula una política óptima sin la evaluación del costo intermedio y sin usar un modelo preestablecido. Otra de las razones de la popularidad del *Q-Learning* es que existe evidencia de que se comporta mejor que otros métodos del Aprendizaje Reforzado [41].

1.6.2.3 Alternativas del algoritmo Q-Learning

El algoritmo *SARSA* propuesto por Sutton en 1996 [42] se diferencia del *Q-Learning* en que selecciona aleatoriamente las acciones y funciona mejor cuando el conjunto de acciones a tomar es muy grande.

Otras variantes que combinan *Q-Learning* y $TD(\lambda)$ son $Q(\lambda)$ propuesto por Watkins en 1989 [32] y el algoritmo de Peng y William que data de 1996 [43].

Otro enfoque es el algoritmo Dyna-Q propuesto por Sutton en 1990 [44] que se caracteriza por aprender directamente del modelo de transición probabilística y de las

---

[2] Técnica matemática usada para resolver problemas de optimización en sistemas que evolucionan en el tiempo y que son susceptibles de ser influenciados por fuerzas externas.

recompensas. Existe una variante de este algoritmo propuesta por Moore y Atkeson en 1993 que realiza la selección de los estados de una forma más eficiente [45].

1.7 Aprendizaje Reforzado aplicado al JSSP

Thomas Gabel y Martin Riedmiller sugieren en [10] la aplicación de las técnicas del Aprendizaje Reforzado para resolver el *JSSP*. Ellos demuestran que interpretar y resolver este problema con un enfoque de aprendizaje con múltiples agentes es beneficioso para obtener soluciones cercanas a las óptimas y que bien pueden competir con otros enfoques de solución reportados en la literatura.

En este trabajo los autores seleccionan el *Q-Learning* para resolver el *JSSP* y lo modelan como un *MDP*, donde un agente k es asociado a cada uno de los m recursos, este agente decide localmente las acciones a tomar. Para un agente tomar una acción significa decidir cuál trabajo va a ser procesado del conjunto de trabajos que están esperando en la cola asociada al recurso correspondiente.

En este enfoque no hay intercambio de información entre los agentes, cada decisión se toma de forma independiente (los agentes no tienen conocimiento de las decisiones tomadas por otros agentes), se asume que todos los agentes se comportan de forma óptima.

En [12] se siguen las ideas propuestas anteriormente al aplicar *RL* al *JSSP* y se implementa un algoritmo *Q-Learning* introduciendo algunas modificaciones con el objetivo de mejorar el comportamiento del aprendizaje.

Ambos enfoques obtienes buenos resultados comparados con los óptimos reportados en la literatura. Dado que el *JSSP-PM* es una generalización del *JSSP* se ha elegido al *Q-Learning* para resolver este problema de secuenciación.

1.8 Consideraciones finales

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

La aplicación del Aprendizaje Reforzado ha reportado buenos resultados en la solución de problemas de secuenciación de tareas.

El *JSSP-PM*, como problema de secuenciación no queda exento de los comentarios anteriores, por lo que el planteamiento de una posible forma de solucionar este problema utilizando el algoritmo *Q-Learning*, perteneciente al Aprendizaje Reforzado, constituye el punto de partida en el capítulo que sigue.

Capítulo 2: Aplicación del algoritmo Q-Learning al problema de secuenciación en máquinas paralelas.

Q-Learning es un algoritmo del Aprendizaje Reforzado que trabaja basado en la capacidad de aprendizaje de los agentes asociados al método. En este capítulo se presentan sus principales características.

Se trata además, la solución al problema de secuenciación en máquinas paralelas a partir de dicho algoritmo, proponiendo dos formas de abordar el problema basadas en la colocación de los agentes.

2.1 Q-Learning

El algoritmo *Q-Learning* trabaja aprendiendo con una función de acción-valor, que da la utilidad esperada de tomar una acción dada en un estado determinado. El núcleo del algoritmo es la actualización de un *Q-valor* en cada iteración. Cada par estado-acción (s,a) (donde $s \in S$ y $a \in A$) tiene un *Q-valor* asociado. 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 subsiguiente estado s'. La *regla de actualización* para cada par estado-acción es la siguiente:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \qquad (1)$$

En esta expresión $\alpha \in]0,1]$ es la *proporción de aprendizaje* y r la *recompensa* o penalidad resultante de tomar una acción a en el estado s. La proporción de aprendizaje determina el grado por el cual el viejo valor es actualizado. Por ejemplo, si la proporción de aprendizaje es $\alpha = 0$, entonces no habrá actualización. Por otro lado, si $\alpha = 1$ el viejo valor es remplazado por el nuevo estimado. Usualmente es utilizado un valor pequeño para la proporción de aprendizaje, por ejemplo $\alpha = 0.1$.

El *factor de descuento* (parámetro γ) tiene el rango de valores desde 0 hasta 1 ($0 \le \gamma < 1$). 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.

El algoritmo puede resumirse como sigue:

```
Inicializar Q(s, a) arbitrariamente
Repetir (para cada episodio):
  Inicializar s
  Repetir (para cada paso del episodio):
      Escoger a desde s usando una política ϵ-greedy
      Tomar acción a, observar r,  s′
      Q(s,a) ← Q(s,a) + α[r + γ max_a' Q(s',a') − Q(s,a)]
      s ← s′
  Hasta que s sea terminal
```

El algoritmo anterior 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. La acción $\varepsilon - greedy$ del método de selección, instruye al agente a seguir la política π la mayoría del tiempo, pero en ocasiones, a elegir una acción de forma aleatoria (con igual probabilidad para cada posible acción en el estado actual s). La probabilidad ε determina cuando usar una acción aleatoria, esto provee algún equilibrio entre explotación y exploración.

2.1.1 Ejemplo

Suponga que se tienen 5 habitaciones conectadas por puertas como muestra la Figura 10. Cada habitación esta nombrada con las letras desde la A hasta la E. Los exteriores

de la edificación pueden considerarse una habitación grande que rodea la construcción y será nombrada F. Note que existen dos puertas a través de las cuales se puede entrar a la edificación, éstas se encuentran en las habitaciones B y E [46].

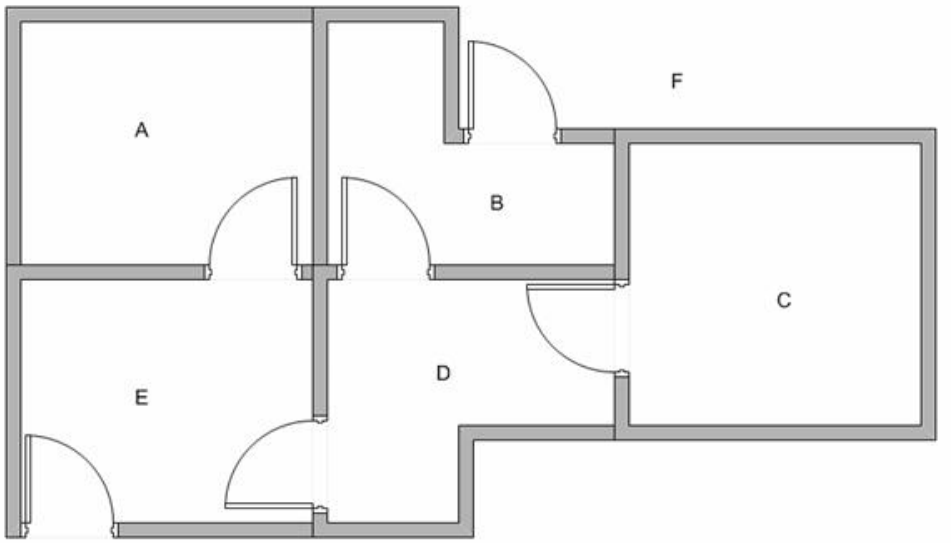

Figura 10: Ejemplo de una construcción de 5 habitaciones

Las habitaciones pueden ser representadas por un grafo donde cada habitación es un nodo y cada puerta es un camino (o enlace) (Figura 11). Si un agente se encuentra en una habitación su objetivo es salir de la construcción, es decir, su objetivo es llegar al nodo F (que representa los exteriores del edificio). Las puertas que llevan inmediatamente al objetivo proveen una recompensa de 100, el resto de las puertas que no tienen conexión directa con el exterior dan una recompensa de 0.

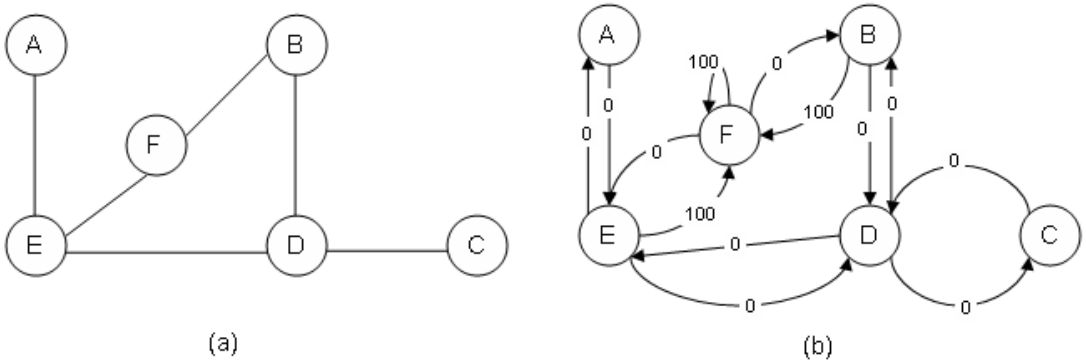

Figura 11: Representación del problema usando un grafo

Imagine que agente es un robot que puede aprender a través de la experiencia. El agente puede pasar de una habitación a la otra pero no tiene conocimiento del ambiente, no conoce cuál es la secuencia de puertas que debe seguir para lograr salir de la construcción. Suponga que se desea modelar una especie de evacuación de un agente desde cualquier habitación. Suponga que existe por ejemplo un agente en la habitación C y el objetivo es que este aprenda cómo salir de la casa (llegar a F). ¿Cómo hacer que el agente aprenda a través de la experiencia?

Cada habitación (incluyendo el exterior) es un estado. Los movimientos de los agentes de una habitación a la otra son considerados las acciones. Suponga ahora que el agente está en el estado C. Del estado C el agente puede ir al estado D porque existe una puerta (conexión) hasta allí. El agente no puede ir al estado B estando en C de forma directa ya que no existe ninguna puerta que conecte dichas habitaciones.

Acorde a las condiciones explicadas anteriormente, las recompensas del ambiente pueden resumirse como sigue:

	A	*B*	*C*	*D*	*E*	*F*
A	–	–	–	–	0	–
B	–	–	–	0	–	100
C	–	–	–	0	–	–
D	–	0	0	–	0	–
E	0	–	–	0	–	100
F	–	0	–	–	0	100

Figura 12: Recompensas del ambiente (Matriz R)

Una matriz similar nombrada Q es insertada en el cerebro del agente, esta matriz representará la memoria de lo que el agente va aprendiendo a través de múltiples experiencias. Las filas de la matriz Q representan el estado actual del agente, la columna de la matriz Q apunta a la acción a tomar para ir al próximo estado. Al comienzo, se dice que el agente no conoce nada, es por eso que Q es una matriz de ceros en un inicio Figura 13.

	A	*B*	*C*	*D*	*E*	*F*
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Figura 13: Q-Matriz inicial

Seleccionando como estado inicial la habitación B y echando un vistazo a la segunda fila (estado B) de la matriz R (Figura 12), existen dos acciones posibles para el estado B, éstas son ir al estado D o al estado F. Suponga que la acción seleccionada es ir a la habitación F.

Una vez en el estado F, el q-valor correspondiente a la fila B (estado) y la columna F (acción seleccionada) es actualizado de acuerdo a la expresión (1). Luego de tomar esa acción, el estado F se convierte en el estado actual y el proceso se repite. Debido a que F es el estado objetivo, el episodio se termina y comienza otro manteniéndose la Q-matriz resultante.

En este caso el único valor que sufre modificaciones es el correspondiente a la fila B y la columna F, de acuerdo con la regla de actualización, se convierte en 100 para el próximo episodio.

En el segundo episodio se va a tener a D como el estado inicial. Mirando la cuarta fila de la matriz R (Figura 12), se tienen tres acciones posibles, ir al estado B, al C o al E. Suponga que se selecciona ir al estado B como acción, ahora imagine que se encuentra en el estado B. Mirando la segunda fila de la matriz de recompensas R (estado B). Estando en B se tienen dos posibles acciones, ir a D o a F. Se calcula el Q-valor y el resultado se muestra en la Figura 14.

	A	*B*	*C*	*D*	*E*	*F*
A	0	0	0	0	0	0
B	0	0	0	0	0	100
C	0	0	0	0	0	0
D	0	80	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Figura 14: Q-Valores luego de dos episodios

El ciclo se repite porque B no es el estado objetivo. Para el nuevo ciclo, el estado actual es el B. De las dos posibles acciones, la acción seleccionada e ir al estado F. Luego se ha obtenido el estado objetivo y el episodio se concluye.

Si los agentes adquieren mayor experiencia a medida que aumentan los episodios, finalmente encontrarán los Q-valores de convergencia siguientes:

	A	*B*	*C*	*D*	*E*	*F*
A	–	–	–	–	400	–
B	–	–	–	320	–	500
C	–	–	–	320	–	–
D	–	400	256	–	400	–
E	320	–	–	320	–	500
F	–	400	–	–	400	500

Figura 15: Q-Valores de convergencia

2.2 Aplicación del algoritmo Q-Learning al problema de secuenciación en máquinas paralelas

Los resultados obtenidos por los enfoques abordados en [10] y [12] para el *JSSP* son los óptimos o cercanos a éstos y pueden competir con los resultados de otros enfoques que tratan el mismo problema encontrados en la literatura. De lo que se concluye que el Aprendizaje Reforzado, específicamente el algoritmo *Q-Learning* es una buena vía de solucionar el *JSSP*.

El *JSSP-PM* en adición al problema de secuenciación de los trabajos del JSSP, tiene además el problema de asignación de los trabajos a una de las máquinas del conjunto

de máquinas de cada tipo. Tratar este problema como *MDP* y solucionarlo utilizando un algoritmo *Q-Learning* puede reportar buenos resultados.

El trabajo actual se basa en la idea de Gabel y Riedmiller en la que se usan costos en lugar de recompensas lo que significa que *Q-valores* pequeños corresponden a buenos pares estado-acción. Por lo que es necesario hacer una modificación a la regla de actualización de los q-valores del algoritmo *Q-Learning.*

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(c(s,a,s') + \gamma \min_{b \in A(s')} Q(s',b)) \quad (2)$$

Teniendo en cuenta que el *makespan* es minimizado siempre que se logre tener tan pocos trabajos esperando a ser procesados en los recursos como sea posible, Gabel y Riedmiller definen como función de costo la expresión siguiente:

$$c(s,a,t) = \sum_{j=1}^{m} i \ \ | i \ en \ \ cola \ \ en \ \ j \quad (3)$$

Se incurre en altos costos cuando existen muchos trabajos esperando a ser procesados en el sistema, por lo que la utilización de los recursos es pobre.

Analizadas las funciones propuestas se decidió implementar un algoritmo *Q-Learning* usando la misma función de costo, pero además se toman en cuenta las modificaciones hechas en [12], con el objetivo de mejorar el comportamiento del aprendizaje.

Como segunda alternativa para la función de costo, se tomó en cuenta en lugar del número de trabajos en cola, el costo de seleccionar una acción, es decir el tiempo de procesamiento de la acción tomada. Esto significa hacer la selección basada en el costo de procesamiento de los trabajos. En este caso la mejor acción es considerada el trabajo con el menor tiempo de procesamiento.

$$c(s,a,t) = p_{ij} \quad (4)$$

La tercera alternativa para la función de costo está basada en la idea de Gabel y Riedmiller, pero se calcula la suma de los tiempos de procesamiento de los trabajos que se encuentran esperando en la cola, en lugar de calcular el número de trabajos en cola. Esto puede dar una mejor idea del tiempo ya que puede existir la misma cantidad de

trabajos en cola, pero la suma de los tiempos de procesamiento puede que sea diferente.

$$c(s,a,t) = \sum p_{ij} \mid i \ en \ cola \ \ en \ recurso \ \ \ j \tag{5}$$

Las instancias del problema, descritas anteriormente brindan el orden en el que los trabajos deben ser procesados y el tiempo asociado a cada una de sus operaciones. Se toma como ejemplo la instancia ft06 duplicada (Figura 16).

```
12  6
2   1   0   3   1   6   3   7   5   3   4   6
1   8   2   5   4  10   5  10   0  10   3   4
2   5   3   4   5   8   0   9   1   1   4   7
1   5   0   5   2   5   3   3   4   8   5   9
2   9   1   3   4   5   5   4   0   3   3   1
1   3   3   3   5   9   0  10   4   4   2   1
2   1   0   3   1   6   3   7   5   3   4   6
1   8   2   5   4  10   5  10   0  10   3   4
2   5   3   4   5   8   0   9   1   1   4   7
1   5   0   5   2   5   3   3   4   8   5   9
2   9   1   3   4   5   5   4   0   3   3   1
1   3   3   3   5   9   0  10   4   4   2   1
```

Figura 16: Instancia ft06'

Recordemos que en el problema de secuenciación de tareas los trabajos son representados por una lista de operaciones y que cada operación está formada por el tipo de máquina donde se va a ejecutar y el tiempo que demorará su procesamiento. En la Figura 17 se puede observar la descripción de un trabajo tal y como aparecen en las instancias.

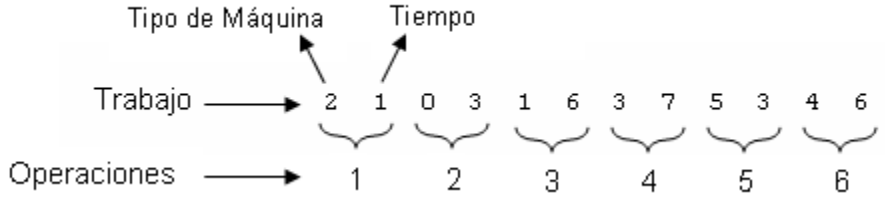

Figura 17: Descripción de un trabajo

En la implementación que se realiza del algoritmo, el primer paso es obtener los datos necesarios de la instancia del problema a resolver y crear dos estructuras de datos

principales, una que almacena el orden en el que los trabajos necesitan ser procesados y la otra con los correspondientes tiempos de procesamiento (Figura 18).

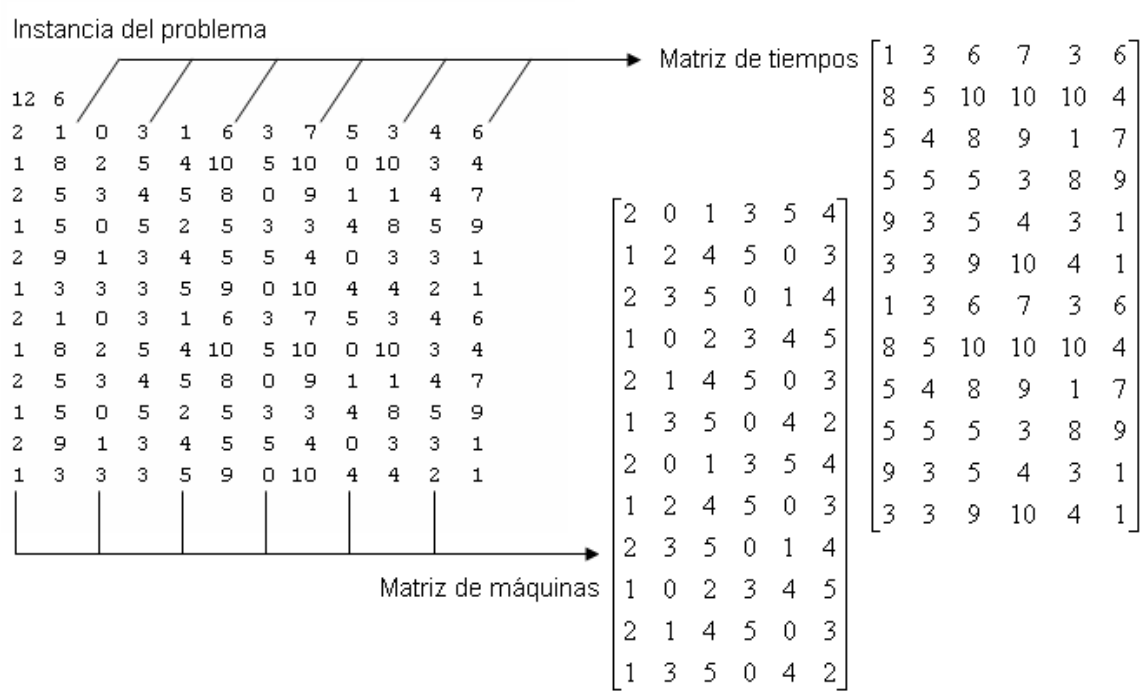

Figura 18: Matrices obtenidas a partir de la instancia ft06'

Una vez almacenada la información, los agentes del *Q-Learning* comenzarán a seleccionar sus próximas acciones.

Cuando un trabajo termina de ser procesado por una máquina se conoce cuál es el próximo tipo de recurso al que debe ser enviado, lo que se debe decidir es cuándo ese trabajo va a ser procesado por el próximo recurso y en cuál de las máquinas asociadas a él se va a procesar. Quizás un trabajo que llegue de último a un recurso debe ser seleccionado para procesarse primero.

En cada paso, para seleccionar una acción, cada agente toma en cuenta sólo los trabajos que pueden ser procesados en ese momento de acuerdo a las restricciones del problema.

Cuando un agente selecciona un trabajo de un conjunto de candidatos, puede seleccionar el mejor, tomando en consideración el Q-valor asociado a este (explotación) o puede seleccionar un trabajo de forma aleatoria (exploración). La acción es ejecutada de acuerdo a la política ϵ-*greedy* seguida por el agente.

Inicialmente, en la instancia ft06' solo las máquinas de los tipos 1 y 2 pueden comenzar la producción ya que las operaciones iniciales de todos los trabajos de esa instancia indican que el procesamiento debe empezar por dichas máquinas. En la Figura 19 se muestra la primera columna correspondiente a la instancia mencionada que muestra los tipos de máquinas encargadas de realizar la primera operación de los trabajos.

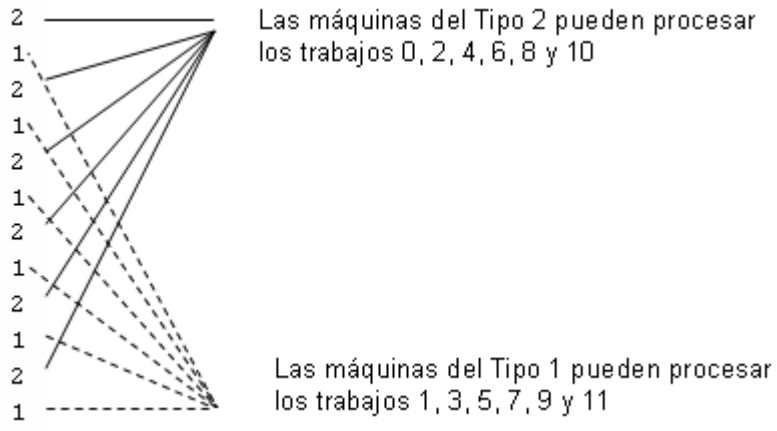

Figura 19: Tipos de máquina de las primeras operaciones

Existen exactamente 2 máquinas de cada tipo disponibles para ejecutar los trabajos. Llamemos $M_{1,1}$ y $M_{1,2}$ a las dos máquinas del tipo 1 y $M_{2,1}$ y $M_{2,2}$ a las 2 máquinas de tipo 2. Los agentes deben decidir qué trabajos de los posibles van a ser procesados y cuáles de las máquinas de cada tipo los van a ejecutar.

La recompensa numérica que agente recibe cuando termina de procesar un trabajo es el costo de realizar dicho trabajo, por tanto, el menor costo determina la mejor acción. Entiéndase por costos los tiempos de procesamiento de los trabajos en los recursos.

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

En este trabajo se proponen dos formas de aplicar *Q-Learning* al *JSSP-PM* basadas en la forma de distribuir los agentes en el medio.

2.2.1 QL con agentes por tipo de recurso

Con esta variante un agente es asociado a cada uno de los m tipos de recursos (tipos de máquinas). Este agente decide localmente las acciones elementales. Desde el punto

de vista del agente tomar una acción significa decidir cuál trabajo va a ser procesado, del conjunto de trabajos que se encuentran esperando por el tipo de recurso correspondiente y decidir cuál de las k máquinas de este tipo es la que va a procesarlo.

Cada agente tiene una cola con los trabajos que están esperando para ser procesados por alguna de las máquinas del tipo que él representa (Figura 20).

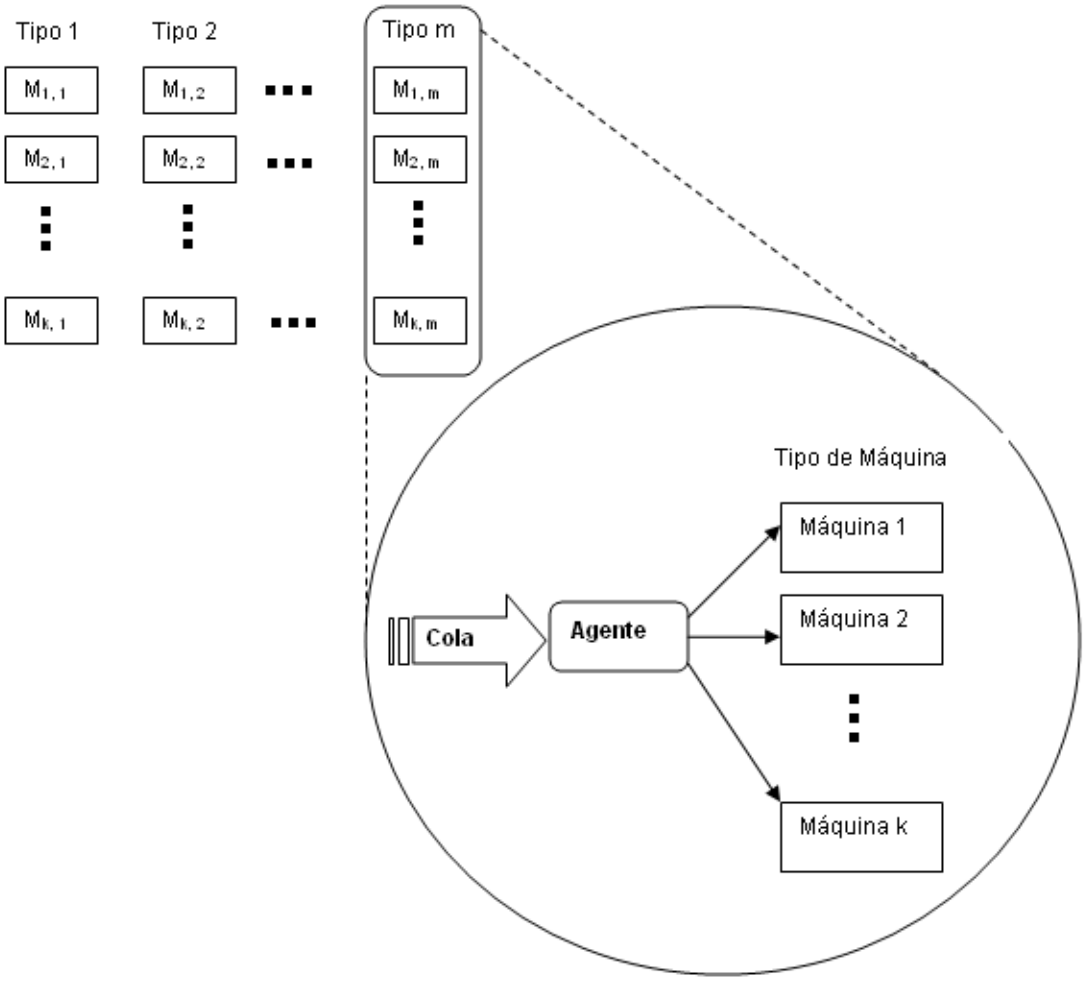

Figura 20: Agente por tipo de recurso

Un agente no puede tomar una decisión en cada instante de tiempo sino cuando el tipo de recurso al que está asociado le queden máquinas libres, o en caso de que estén todas realizando operaciones cuando alguna de ellas haya terminado una operación, ya que cada recurso sólo puede procesar un trabajo a la vez, y además un trabajo sólo puede estar procesándose en una máquina en un instante de tiempo determinado.

Estados y acciones: Existe un agente asociado a cada tipo de recurso, y este agente tomará decisiones sobre futuras acciones. Para un agente tomar un acción significa que

debe decidir cuál va a ser el próximo trabajo a procesar del conjunto de trabajos posibles (estos son los que están esperando en la cola del agente del correspondiente tipo de recurso), y decidir cuál de las k máquinas de este tipo es la que va procesarlo. Cada agente tiene una visión local, sólo tiene acceso a la información asociada a su recurso (cantidad y disponibilidad de las máquinas del tipo en cuestión), y los trabajos que están esperando por él.

El agente selecciona la próxima operación va procesar utilizando la política π y decide cuál máquina va a hacerlo eligiendo la máquina con menor tiempo de producción.

Q-Valores: Para cada uno de los m tipos de recursos hay n posibles trabajos a procesar, entonces para almacenar los q-valores es construida una matriz con n filas y m columnas (el número de filas es el número de trabajos y el número de columnas es el número de agentes).

2.2.2 QL con agentes por recurso

Esta variante propone asociar un agente a cada máquina involucrada en el procesamiento, es decir, si se tiene m tipos de recursos (tipos de máquinas) y k máquinas por cada uno de ellos, se tendría un total de $m * k$ agentes. Para un agente tomar una acción, significa decidir cuál trabajo va a ser procesado, del conjunto de trabajos que se encuentran esperando por la máquina correspondiente y decidir cuál de las siguientes k máquinas va a procesarlo luego que acabe de ser procesado por la máquina en cuestión.

Cada agente tiene una cola con los trabajos que están esperando para ser procesados por la máquina a la que él representa (Figura 21).

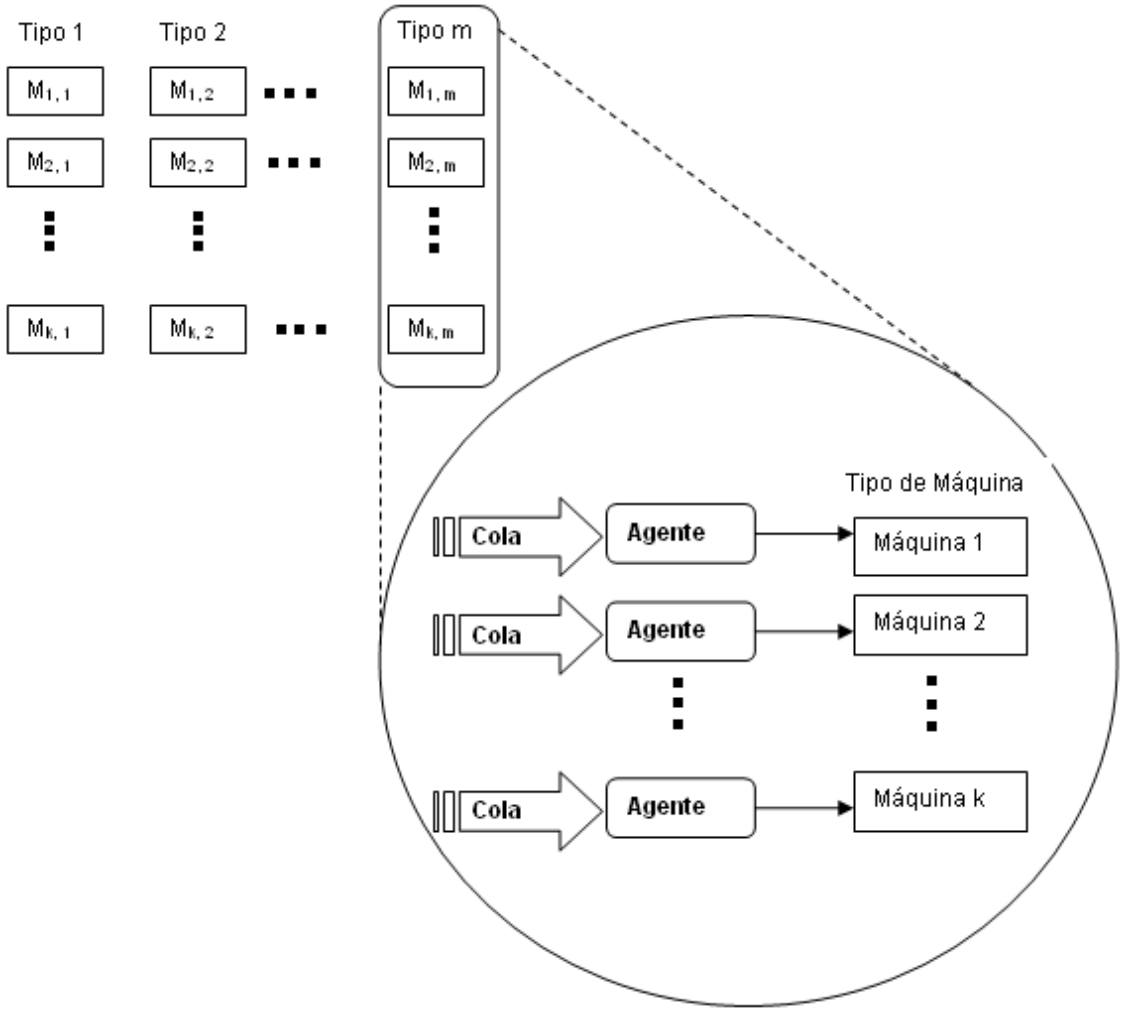

Figura 21: Agentes por recurso

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 sólo puede procesar un trabajo la vez, y además un trabajo sólo puede estar procesándose en una máquina en un instante de tiempo determinado.

Estados y acciones: Existe un agente asociado a cada recurso, y este agente tomará decisiones sobre futuras acciones. Para un agente el tomar una acción significa que debe decidir cuál va a ser el próximo trabajo a procesar del conjunto de trabajos posibles (estos son los que están esperando en la cola asociada al recurso correspondiente), y decidir cuál de las k máquinas del tipo de recurso correspondiente a la próxima operación que debe realizar es la que va a procesarlo. El agente además de tener una visión local con la información asociada a su recurso como los trabajos

que están esperando por él, tiene conocimiento acerca de las colas de espera de los agentes que representan las máquinas candidatas a ser la próxima en procesar el trabajo que actualmente procesa el recurso al que representan.

El agente selecciona la próxima operación va procesar utilizando la política π y decide a cuál máquina va a enviar el trabajo cuya operación ha sido procesada eligiendo la máquina con menor tiempo de producción.

Q-Valores: k matrices de n filas y m columnas son construidas para almacenar los q-valores ya que para cada uno de los m tipos de recursos hay k máquinas de cada uno y n posibles trabajos a procesar. El número de filas es el número de trabajos y el número de columnas es el número de agentes que representa cada máquina en las k matrices.

2.3 Complejidad computacional.

La complejidad computacional del Q-*Learning* está determinada por las acciones que ejecutan los agentes [47].

En la variante donde los agentes se encuentran asociados a los tipos de recursos, se tienen m agentes, uno por cada tipo de recurso, que debe seleccionar cuál operación va a procesar de todas las posibles, la máxima cantidad de operaciones es igual al número de trabajos (n) y en qué máquina va a ejecutarla de las k posibles. Esta selección de acciones se realiza un número fijo de ciclos denominado c. De forma general la complejidad del método queda como un $O(c \cdot m \cdot n \cdot k)$.

En la variante donde los agentes se encuentran asociados a los recursos, se tienen $m \cdot k$ agentes (uno por cada recurso), que debe seleccionar cuál operación va a procesar de todas las posibles (como máximo igual al número de trabajos) y a cuál máquina de las posibles debe enviar el trabajo para que sea procesado (como máximo k). Esta selección de acciones se realiza un número fijo de ciclos denominado c. De forma general la complejidad del método queda como un $O(c \cdot m \cdot n \cdot k^2)$.

En resumen la complejidad del *Q-Learning* es un $O(Acciones \ \ de \ los \ \ agentes)$.

2.4 Conclusiones parciales.

En este capítulo se presentaron las modificaciones necesarias que deben hacerse al algoritmo *Q-Learning* para aplicarlo al problema de secuenciación en máquinas paralelas.

Se proponen dos enfoques de solución al *JSSP-PM* con la utilización del Q-Learning: el primero coloca un agente por cada tipo de recurso y el segundo coloca un agente por recurso.

Capítulo 3: Resultados Experimentales

En este capítulo se muestran los experimentos realizados con las variantes del algoritmo *Q-Learning* propuestas para las diferentes funciones de costo utilizadas.

Las instancias utilizadas en la experimentación proceden de la *OR-Library* [27] con las modificaciones propuestas en [13].

Se realiza una validación estadística de las propuestas desarrolladas en comparación con los algoritmos reportados en la literatura [13-15] que solucionan el *JSSP-PM*.

3.1 Técnicas estadísticas utilizadas para la validación de los resultados.

El comportamiento no determinístico de los algoritmos sobre múltiples conjuntos de datos es una razón por la cual no existe un procedimiento establecido para poder compararlos[48, 49].

Distintas técnicas estadísticas son utilizadas para tratar de determinar si las diferencias encontradas entre dos algoritmos son significativas. En este trabajo se aplican las pruebas no paramétricas [50], debido a que los resultados obtenidos por los algoritmos implementados no cumplen las condiciones requeridas para poder usar de forma correcta comparaciones paramétricas [51-53].

En este trabajo se aplicará la prueba de de Iman-Davenport [54] para detectar diferencias entre un conjunto de algoritmos, si no se detectan diferencias, se puede concluir que los algoritmos involucrados obtienen resultados que no difieren significativamente unos de otros. Si se detectan diferencias significativas en el grupo de algoritmos entonces se deben utilizar pruebas pareadas como: el test de Holm [55] con y el de Shaffer [56], con el objetivo de detectar dónde se encuentran las diferencias.

En caso de que sea necesario establecer una comparación entre dos algoritmos, se realiza un prueba de Wilcoxon [57].

3.2 Estudio Experimental

Todos los algoritmos fueron implementados en el lenguaje java JDK 1.5, como entorno de desarrollo (IDE) se utilizó Eclipse 3.1 y fueron ejecutados en un procesador Intel Pentium IV a 3.0 GHz, con 1 GB de RAM.

En el funcionamiento del algoritmo *Q-Learning* intervienen parámetros que pueden tomar diferentes valores:

- 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 0.1 es usualmente utilizado para este parámetro en la literatura.
- El *número de ciclos* es el parámetro de parada del algoritmo propuesto.

Para las corridas de los algoritmos implementados se utilizaron los juegos de datos de la *OR-Library* [27], en específico las 15 primeras instancias Lawrence con las modificaciones explicadas en el capítulo 1.

De las instancias originales se obtienen las denominadas la01’ hasta la15’, duplicando los trabajos para k = 2 y las la01” hasta la15” triplicando los trabajos para k = 3. Esto significa que para k = 2 se agrega una copia de cada trabajo y para k = 3 dos copias de cada trabajo, con el objetivo de encontrar el óptimo reportado para cada instancia cuando k = 2 y k = 3 máquinas por cada tipo de recurso.

3.2.1 Resultados del Q-Learning con agentes por tipo de recurso.

El algoritmo *Q-Learning* propuesto que trabaja colocando un agente por cada tipo de recurso fue probado utilizando las tres funciones de costo propuestas. El Anexo 1

muestra los resultados obtenidos en la corrida de los algoritmos para las instancias involucradas en el estudio.

- **QL_ATR1**: es el algoritmo *Q-Learning* que utiliza el número de operaciones en cola como función de costo.
- **QL_ATR2**: es el algoritmo *Q-Learning* que utiliza el tiempo de procesamiento de la operación como función de costo.
- **QL_ATR3**: es el algoritmo *Q-Learning* que utiliza la suma de los tiempos de procesamiento de las operaciones en cola como función de costo.

En el análisis estadístico primeramente se aplica un test de *Iman-Davenport* a los valores obtenidos por los algoritmos anteriores, obteniéndose 0.05104686868359735 valor del test por lo que no se detectan diferencias significativas entre las variantes del algoritmo *Q-Learning* con agentes por tipo de recurso probadas con las diferentes funciones de costo. El valor de α utilizado es 0.05.

En la Tabla 2 se muestran los resultados del test de *Wilcoxon* con el objetivo de comparar dos a dos las variantes utilizadas.

Tabla 2 Resultados del test de Wilcoxon

Algoritmo	R+	R-	Valor p	Hipótesis
QL_ATR2 - QL_ATR1	**11**	15	0.71	A
QL_ATR3 - QL_ATR1	18	**8**	0.041	R
QL_ATR3 - QL_ATR2	19	**8**	0.10	A

Entre las alternativas QL_ATR2 y QL_ATR1 no se encuentran diferencias significativas aunque según los rangos positivos y negativos QL_ATR1 obtiene ligeramente mejores resultados que QL_ATR2.

Entre las alternativas QL_ATR3 - QL_ATR2 no se encuentran diferencias significativas aunque según los rangos positivos y negativos QL_ATR2 obtiene mejores resultados que QL_ATR3.

En la comparación entre QL_ATR3 - QL_ATR1 la hipótesis es rechazada por lo que los resultados de QL_ATR1 son significativamente mejores que los de QL_ATR3. Es decir

la variante que utiliza el número de operaciones en cola es significativamente superior a la que utiliza la suma del tiempo de procesamiento de las operaciones en cola.

De este análisis puede derivarse la conclusión de que la función de costo que obtiene los resultados más bajos es la que utiliza la suma de los tiempos de procesamiento de los trabajos en cola, las otras dos funciones utilizadas tienen un comportamiento similar.

3.2.2 Resultados del Q-Learning con agentes por recurso

El algoritmo *Q-Learning* propuesto que trabaja colocando un agente por cada recurso fue probado utilizando las tres funciones de costo propuestas. El Anexo 2 muestra los resultados obtenidos en la corrida de los algoritmos para las instancias involucradas en el estudio.

- **QL_AR1**: es el algoritmo *Q-Learning* que utiliza el número de operaciones en cola como función de costo.
- **QL_AR2**: es el algoritmo *Q-Learning* que utiliza el tiempo de procesamiento de la operación como función de costo.
- **QL_AR3**: es el algoritmo *Q-Learning* que utiliza la suma de los tiempos de procesamiento de las operaciones en cola como función de costo.

El valor del test de *Iman-Davenport* aplicado para comparar las variantes anteriores es de 0.4528886741384791 por lo que no se detectan diferencias significativas entre las propuestas anteriores.

3.2.3 Estudio comparativo entre las variantes del algoritmo *Q-Learning*

En este epígrafe se presenta un estudio comparativo entre cada una de las alternativas diseñadas, el *Q-Learning* con agentes por tipo de recurso (ATR) y el *Q-Learning* con agentes por recurso (AR).

En la Tabla 3 se pueden apreciar los resultados obtenidos para ambos algoritmos comparados con los óptimos reportados en la literatura.

Tabla 3 Resultados obtenidos por AR y ATR

Instancia	Óptimo	ATR	AR
la01'	**666**	**666**	**666**
la02'	**655**	752	721
la03'	**597**	598	636
la04'	**590**	606	647
la05'	**593**	**593**	**593**
la06'	**926**	**926**	**926**
la07'	**890**	**890**	**938**
la08'	**863**	**863**	894
la09'	**951**	**951**	963
la10'	**958**	**958**	**958**
la11'	**1222**	**1222**	**1222**
la12'	**1039**	**1039**	**1039**
la13'	**1150**	**1150**	**1150**
la14'	**1292**	**1292**	**1292**
la15'	**1207**	1304	1244
la01"	**666**	**666**	718
la02"	**655**	745	747
la03"	**597**	589	646
la04"	**590**	614	685
la05"	**593**	**593**	**593**
la06"	**926**	927	942
la07"	**890**	925	947
la08"	**863**	883	893
la09"	**951**	951	992
la10"	**958**	**958**	**958**
la11"	**1222**	**1222**	1228
la12"	**1039**	**1039**	1054
la13"	**1150**	**1150**	1153
la14"	**1292**	**1292**	**1292**
la15"	**1207**	1281	1344

Fueron seleccionadas para hacer las pruebas los mejores resultados obtenidos de cada alternativa para cada instancia. Para esta comparación se utiliza el test de Wilcoxon con un valor de significación de 0.5, los resultados son mostrados en la Tabla 4.

Tabla 4 Resultados del test de Wilcoxon en la comparación entre AR y ATR

Algoritmo	R+	R-	Valor p	Hipótesis
AR - ATR	16	**3**	0.038	R

Se detectan diferencias significativas entre los algoritmos obteniéndose mejores resultados con la variante del Q-Learning que utiliza los agentes asociados a los tipos de recursos.

3.3 Estudio comparativo entre el Q-Learning y las variantes ACO, AG y AG-ACO

En la literatura se encontraron tres variantes que solucionan el *JSSP-PM*, en este epígrafe se realiza una comparación de dichas soluciones con la propuesta de solución que utiliza *Q-Learning*.

AG-ACO: es una heurística híbrida entre *ACO* (Ant Colony Optimization / Optimización basada en Colonias de Hormigas) y *AG* (Genetic Algorithm / Algoritmos Genéticos) para dar solución al JSSP–PM [13].

El algoritmo trabaja basado en un mecanismo de cooperación entre las dos heurísticas centrado en la actividad individual de ambos algoritmos en la que cada uno de ellos ejecuta un proceso estocástico y cooperan compartiendo y modificando el mismo ambiente. El intercambio está basado en la regla de actualización de la feromona y en el operador de selección fundamentalmente.

AG: es el enfoque basado en Algoritmos Genéticos que soluciona el *JSSP-PM* propuesto en [15].

ACO: es una forma de solución al *JSSP-PM* que utiliza la Optimización basada en Colonias de Hormigas. En [14] se presentan los resultados de aplicar ACO al *JSSP-PM*, en el algoritmo la hormiga utiliza el grafo que representa el problema a solucionar para encontrar los caminos óptimos y así solucionar el problema.

QL: es el algoritmo *Q-Learning* propuesto, en la comparación se utilizan los resultados que brinda el enfoque que asocia los agentes a los tipos de recursos que fue la variante que mejores resultados obtuvo de las propuestas.

En la Tabla 5 se muestran los resultados obtenidos por los diferentes enfoques comparados con los óptimos reportados en la literatura.

Tabla 5 Resultados del algoritmo *Q-Learning* y los reportados en la literatura

Instancia	Óptimo	AG-ACO	AG	ACO	QL
la01'	**666**	**666**	675	669	**666**
la02'	**655**	688	712	693	721
la03'	**597**	626	644	642	598
la04'	**590**	611	628	625	606
la05'	**593**	**593**	**593**	**593**	**593**
la06'	**926**	**926**	**926**	**926**	**926**
la07'	**890**	**890**	939	908	**890**
la08'	**863**	**863**	872	865	**863**
la09'	**951**	**951**	**951**	**951**	**951**
la10'	**958**	**958**	**958**	**958**	**958**
la11'	**1222**	**1222**	**1222**	**1222**	**1222**
la12'	**1039**	**1039**	1051	1041	**1039**
la13'	**1150**	**1150**	1158	1150	**1150**
la14'	**1292**	**1292**	**1292**	**1292**	**1292**
la15'	**1207**	1246	1284	1249	1244
la01"	**666**	677	722	689	**666**
la02"	**655**	712	728	707	745
la03"	**597**	673	703	659	589
la04"	**590**	629	654	635	614
la05"	**593**	**593**	599	594	**593**
la06"	**926**	936	961	932	927
la07"	**890**	922	960	908	925
la08"	**863**	871	875	872	883
la09"	**951**	952	977	958	**951**
la10"	**958**	**958**	970	961	**958**
la11"	**1222**	1239	1238	1224	**1222**
la12"	**1039**	1049	1057	1069	**1039**
la13"	**1150**	1163	1167	1156	**1150**
la14"	**1292**	**1292**	1300	1293	**1292**
la15"	**1207**	1283	1311	1269	**1281**

Con el objetivo de determinar diferencias significativas entre los resultados del algoritmo QL propuesto y los reportados en la literatura se aplicó un test de *Iman-Davenport*. El test encuentra diferencias significativas en el grupo de algoritmos (valor del test 4.4386274100837233E-8.), para un valor de $\alpha = 0.10$. La Tabla 6 muestra los ranking medios asociados a los algoritmos que están siendo evaluados.

Tabla 6: Ranking medios

Algoritmo	Ranking
AG-ACO	2.133
AG	3.55
AG	2.516
QL	1.80

Posteriormente, para detectar específicamente entre qué algoritmos existen diferencias significativas se aplica un test de comparaciones múltiples de Holm, donde se utiliza como muestra de control el algoritmo QL. Como resultado de este test se encontraron diferencias significativas con respecto al algoritmo AG, para un valor de $\alpha = 0.10$. La Tabla 7 muestra estos resultados.

Tabla 7: Resultados del test de Holm para un valor de confianza de 0.10

Algoritmo	z=(R0-Ri)/SE	p	a/i	hipótesis
AG	5.249999999999997	1.520992103444E-7	0.33	R
ACO	2.15	0.03155521478218	0.05	R
AG-ACO	0.999999999999998	0.317310507862914	0.05	A

Como se puede apreciar en la Tabla 7 la hipótesis de igualdad es rechazada (R) en las comparaciones del algoritmo de control QL contra el algoritmo AG y el algoritmo ACO. De lo anterior se deduce que el algoritmo de control (QL) obtiene resultados significativamente superiores al algoritmo AG y al ACO. No siendo así con respecto al algoritmo AG-ACO donde se acepta la hipótesis de igualdad (A).

Para complementar lo anterior se realizaron comparaciones múltiples de todos los pares de algoritmos utilizando las pruebas de Holm y Shaffer. La Tabla 8 muestra los resultados obtenidos. Ambas pruebas coinciden en las diferencias existentes entre el algoritmo QL con AG y ACO. La tabla muestra además que el algoritmo AG tiene diferencias significativas con respecto a los algoritmos ACO y AG-ACO.

Tabla 8: Test de Holm y test de Shaffer para un valor de confianza de 0.10

Algoritmos	$z=(R_0-R_i)/SE$	p	Holm	Shaffer	Hipótesis
AG vs QL	5.24999999999	1.520992103E-7	0.016	0.016	R
AG-ACO vs AG	4.24999999999	2.137705154E-5	0.02	0.033	R
AG vs ACO	3.09999999999	0.00193520642	0.025	0.033	R
ACO vs QL	2.15	0.03155521478	0.033	0.033	R
AG-ACO vs ACO	1.15000000000	0.25014387127	0.05	0.05	A
AG-ACO vs QL	0.99999999999	0.31731050786	0.01	0.01	A

Adicionalmente se realizó la prueba de Wilcoxon que obtiene iguales resultados a los anteriormente mencionados, pero de ella se pueden obtener conclusiones asociadas a los rangos obtenidos.

Tabla 9: Resultados del test de Wilcoxon para $\alpha = 0.05$

Algoritmo	R+	R-	Valor p	Hipótesis
QL - AG_ACO	**4**	13	0.102	A
QL - ACO	**5**	19	0.042	R
QL - AG	**3**	22	0.00	R
AG - AG_ACO	23	**1**	0.00	R
ACO - AG_ACO	16	**7**	0.33	A
ACO – AG	**1**	23	0.00	R

Entre las alternativas QL y AG-ACO no se encuentran diferencias significativas aunque según los rangos positivos y negativos QL obtiene ligeramente mejores resultados que AG.

Entre las alternativas QL y ACO se encontraron diferencias significativas que muestran que los resultados obtenidos por el QL son superiores a los que reporta el algoritmo ACO.

En la comparación entre el QL y AG la hipótesis es rechazada por lo que los resultados de QL son significativamente superiores a los de AG.

Es válido destacar que además los resultados obtenidos por ACO y AG-ACO son significativamente superiores a los de AG y que entre ACO y AG-ACO no existen diferencias significativas pero el valor de los *rankings* obtenidos favorece a AG-ACO.

3.4 Conclusiones Parciales

Del análisis estadístico realizado a los resultados obtenidos por los algoritmos *Q-Learning* aplicados al JSSP-PM se concluye que:

En la variante con agente por tipo de recurso aunque no se encuentran diferencias significativas entre las ejecuciones con las tres funciones de costo utilizadas, las funciones que utilizan el número de operaciones en espera y el tiempo de procesamiento de la operación muestran resultados ligeramente superiores a la función que utiliza la suma de los tiempos de procesamiento de las operaciones en cola.

En la variante que utiliza los agentes por recurso no se obtienen diferencias significativas con el uso de las tres funciones costo.

Los resultados obtenidos por el *Q-Learning* con agentes por tipo de recurso son significativamente superiores a los obtenidos aplicando el enfoque con agentes por recursos.

Comparando los resultados del algoritmo *Q_Learning* con los reportados por las variantes AG, ACO y AG-ACO, se concluye que el QL muestra resultados superiores con respecto a las variantes AG y ACO y estadísticamente iguales a los que muestra AG-ACO.

Conclusiones

La aplicación del Aprendizaje Reforzado ha reportado buenos resultados en la solución de problemas de secuenciación de tareas. En particular, en este trabajo se seleccionó el algoritmo *Q-Learning* para solucionar un problema de este tipo; el de secuenciación en máquinas paralelas (*JSSP-PM*) proponiéndose dos variantes relacionadas con la colocación de los agentes con los que trabaja el algoritmo, en los recursos correspondientes al problema. Se logró una implementación de cada variante del algoritmo que obtienen soluciones óptimas.

Las soluciones obtenidas por las variantes propuestas se han comprobado estadísticamente obteniéndose los resultados siguientes:

- No existen diferencias significativas con el uso de las diferentes funciones de costo para ninguna de las dos variantes propuestas del Q-Learning.
- El enfoque que propone colocar los agentes por tipo de recurso obtiene resultados significativamente superiores al enfoque que coloca los agentes por recurso.

Al comparar estadísticamente las soluciones del algoritmo Q-Learning con las reportadas por AG, ACO y AG-ACO se obtiene que el primero muestra resultados superiores con respecto a las variantes AG y ACO y estadísticamente iguales a los que muestra AG-ACO.

Recomendaciones

Aplicar el enfoque propuesto al *Flexible Job Shop Scheduling Problem,* el cual es un problema similar al problema de secuenciación en máquinas paralelas, pero difiere en que los recursos disponibles para ejecutar una operación no son idénticos, por tanto se necesita un nivel extra de decisión para poder realizar una buena asignación maquina-operación.

Introducir otros tipos de recompensa en la regla de actualización del Q-Learning de forma que los agentes tengan más posibilidades de elegir las máquinas que van a procesar las operaciones.

Bibliografía

1. Sahni, S. and T. Gonzalez, P-complete approximation problems. Journal of the Association for Computing Machinery, 1976. 38: p. 555-565.
2. Sutton, R. and A. Barto, Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA. , 1998.
3. Stachniss, C. and W. Burgard, The Markov Decision Problem, University of Freiburg. 2002.
4. Givan, B. and R. Parr, An Introduction to Markov Decision Processes, P. University and D. University, Editors. 2003.
5. Duan, Y., Q. Liu, and X. Xu, Application of reinforcement learning in robot soccer. Engineering Applications of Artificial Intelligence, 2007. 20(7): p. 936-950.
6. Zhou, C. and Q. Meng, Dynamic balance of a biped robot using fuzzy reinforcement learning agents. Fuzzy Sets and Systems 2003. 134(1): p. 169 - 187.
7. Ghory, I., Reinforcement learning in board games, Technical report, University of Bristol. 2004.
8. Eck, N.J.v. and M.v. Wezel, Application of reinforcement learning to the game of Othello. Computers and Operations Research, 2008. 35(6): p. 1999 - 2017.
9. Silver, D., R. Sutton, and M. Muller. Reinforcement Learning of Local Shape in the Game of Go. in IJCAI. 2007.
10. Gabel, T. and M. Riedmiller. On a Successful Application of Multi-Agent Reinforcement Learning to Operations Research Benchmarks. in 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning. 2007.
11. Puris, A., et al., Two-Stage ACO to solve the Job Shop Scheduling Problem. Lecture Notes in Computer Science, 2007. 4756(2008): p. 447-456.
12. Jiménez, Y.M., A Multi-Agent Learning Approach for the Job Shop Scheduling Problem, in Department of Computer Science, Computational Modeling Lab. 2008, Vrije Universiteit Brussel: Bruselas.
13. Rossi, A. and E. Boschi, A hybrid heuristic to solve the parallel machines job-shop scheduling problem. Advances in Engineering Software, 2009. 40(2009): p. 118-127.
14. Rossi, A. and G. Dini, Flexible job-shop scheduling with routing fexibility and separable setup times using ant colony optimisation method. Robotics and Computer-Integrated Manufacturing 2007. 23: p. 503-516.
15. Ikeda, K. and S. Kobayashi. GA based on the UV-structure hypothesis and its application to JSP. in PPSN-VI. Sixth international conference on parallel problem solving from nature. 2000.
16. Grossmann, I., Enterprise-wide optimization: A new frontier in process systems engineering. AIChE Journal 2005. 51(7): p. 1846 - 1857.

17. Garrido, A., et al. Heuristic Methods for Solving Job-Shop Scheduling Problems. in ECAI-2000 Workshop on New Results in Planning, Scheduling and Design (PuK2000). 2000.

18. Conway, R., W. Maxwell, and L. Miller, Theory of Scheduling, Addison-Wesley Publishing, Dover Publications. 1967.

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

20. Guerequeta, R. and A. Vallecillo, Técnicas de Diseño de Algoritmos 1998: Servicio de Publicaciones de la Universidad de Málaga.

21. Kirkpatrick, S., D. Gelatt, and M.P. Vecchi, Optimization by Simulated Annealing. Science, 1983. 220 p. 671-680.

22. Glover, F., Heuristics for Integer Programming Using Surrogate Constraints. Decision Sciences, 1977. 8(156-166).

23. Baker, K.R., Introduction to Sequencing and Scheduling. 1974: Wiley.

24. Coffman, E.G., Computer and Job Shop Scheduling Theory. 1976: Wiley.

25. Garey, M.R. and D.S. Jhonson, Computer and Intractability: A Guide to the theory of NP-completeness. 1979: Freeman and Co.

26. French, S., Sequencing and Scheduling. 1992: Wiley.

27. OR-Library. Available from: http://web.cecs.pdx.edu/~bart/cs510ss/project/jobshop/jobshop/.

28. Aranda, J.B., Backtracking, Pontificia Universidad Católica de Chile. 2000.

29. GOIC, M., Programación Dinámica, Carnegie Mellon University. 1998.

30. Marco Dorigo and T. Stutzle, The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. 2000.

31. Sutton, R., Reinforcement Learning, Technical report, University of Alberta. 1999.

32. Watkins, C., Learning from delayed rewards, in Psychology Department. 1989, University of Cambridge.

33. Moriarty, D., A. Schultz, and J. Grefenstette, Evolutionary Algorithms for Reinforcement Learning. Journal of Artificial Intelligence Research, 1999(11): p. 241-276.

34. Kaelbling, L.P., M. Littman, and A. Moore, Reinforcement Learning: a survey. Journal of Artificial Intelligence Research, 1996(4): p. 237-285.

35. Binder, K. and D.W. Heermann, Monte Carlo Simulation. Statistical Physics, 1997.

36. Mahadevan, S., Machine Learning for Robots: A Comparison of Different Paradigms. 1997, University of South Florida.

37. Fishman, G., Monte Carlo: concepts, algorithms, and applications, Springer, Editor. 1996.

38. Sutton, R.S., Learning to predict by the method of temporal differences. Machine Learning, 1988. 3: p. 9-44.

39. Jaakkola, T., M.I. Jordan, and S.P. Singh, On the convergence of stochastic iterative dynamic programming algorithms. Neural Computation, 1994. 6(6): p. 1185-1201.

40. Watkins, C. and P. Dayan, Q-learning. Machine Learning, 1992. 8(3/4): p. 279-292.

41. Lin, L.-J., Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning, 1992. 8: p. 293-321.

42. Sutton, R.S., Generalization in reinforcement learning: Succesful examples using sparse coarse coding. Advances in Neural Information Processing Systems, 1996. 8: p. 1038-1044.

43. Peng, J. and R.J. Williams, Incrementalmulti-step q-learning. Machine Learning, 1996. 22: p. 283-290.

44. Sutton, R.S. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. in 7th International Conference on Machine Learning. 1990.

45. Moore, A.W. and C.G. Atkeson, Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning, 1993. 13: p. 103-130.

46. Teknomo, K. Q-Learning by Examples. 2005; Available from: http://people.revoledu.com/kardi/tutorial/ReinforcementLearning/index.html.

47. Iscen, A., Computational Learning Theory and Reinforcement Learning, Technical report, Oregon State University. 2010.

48. Demsar, J., Statistical comparisons of classifiers over multiple data sets. Journal Machine Learn, 2006: p. 1-30.

49. García, S. and F. Herrera, An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. Journal of Machine Learning Research, 2008. 9: p. 2677-2694.

50. Sheskin, D., Handbook of parametric and nonparametric statistical procedures, ed. C. Editor. 2006: Chapman & Hall.

51. García, S., A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. Soft Computing and Applications, 2009. 13(10): p. 959-977.

52. García, S., A Study on the Use of Non-Parametric Tests for Analyzing the Evolutionary Algorithms' Behaviour. Journal of Heuristics, 2009.

53. Luengo, J., S. García, and F. Herrera, A Study on the Use of Statistical Tests for Experimentation with Neural Networks: Analysis of Parametric Test Conditions and Non-Parametric Tests. Expert Systems with Applications, 2009. 36: p. 7798-7808.

54. Iman, R.L. and J. M. Davenport, Approximations of the critical region of the Friedman statistic. Commun Stat, 1980. 18: p. 571-595.

55. Holm, S., A simple sequentially rejective multiple test procedure. Scand Journal Stat, 1979. 6: p. 65-70.

56. Shaffer, J.P., Multiple hypothesis testing. Annual Review of Psychology, 1995. 46: p. 561-584.

57. Wilcoxon, F., Ajusted p-values for simultaneous inference. Biometrics, 1945. 48: p. 1005-1013.

Anexos

Anexo 1. Resultados del *Q-Learning* con agentes por tipo de recurso

Instancia	Óptimo	QL_ATR1	QL_ATR2	QL_ATR3
la01’	**666**	**666**	**666**	**666**
la02’	**655**	752	783	802
la03’	**597**	598	598	598
la04’	**590**	606	614	614
la05’	**593**	**593**	627	607
la06’	**926**	947	**926**	956
la07’	**890**	**890**	968	934
la08’	**863**	878	**863**	870
la09’	**951**	985	**951**	998
la10’	**958**	960	962	**958**
la11’	**1222**	1269	1232	**1222**
la12’	**1039**	**1039**	1045	1052
la13’	**1150**	1174	**1150**	1162
la14’	**1292**	**1292**	**1292**	1294
la15’	**1207**	1340	1304	1327
la01”	**666**	674	**666**	671
la02”	**655**	745	785	802
la03”	**597**	603	589	609
la04”	**590**	614	614	619
la05”	**593**	**593**	614	606
la06”	**926**	928	927	935
la07”	**890**	925	969	936
la08”	**863**	892	883	902
la09”	**951**	964	**951**	958
la10”	**958**	959	**958**	961
la11”	**1222**	1228	1252	**1222**
la12”	**1039**	1044	**1039**	1052
la13”	**1150**	1152	**1150**	1155
la14”	**1292**	**1292**	1293	**1292**
la15”	**1207**	1363	1281	1363

Anexo 2. Resultados del *Q-Learning* con agentes por recurso

Instancia	Óptimo	QL_AR1	QL_AR2	QL_AR3
la01’	**666**	668	**666**	674
la02’	**655**	721	732	726
la03’	**597**	640	636	636
la04’	**590**	651	681	647
la05’	**593**	**593**	606	595
la06’	**926**	977	**926**	965
la07’	**890**	938	965	971
la08’	**863**	924	934	894
la09’	**951**	985	**951**	**951**
la10’	**958**	962	**958**	975
la11’	**1222**	1232	1259	**1222**
la12’	**1039**	1046	**1039**	1062
la13’	**1150**	1182	**1150**	1169
la14’	**1292**	**1292**	1317	1298
la15’	**1207**	1345	1286	1244
La01”	**666**	718	749	744
La02”	**655**	751	758	747
La03”	**597**	646	672	654
La04”	**590**	685	751	696
La05”	**593**	**593**	640	638
La06”	**926**	942	1014	1051
La07”	**890**	947	987	981
La08”	**863**	1001	893	956
La09”	**951**	1040	1016	992
La10”	**958**	990	**958**	1005
La11”	**1222**	1257	1314	1228
La12”	**1039**	1106	1054	1084
La13”	**1150**	1153	1195	1205
La14”	**1292**	1310	1321	**1292**
La15”	**1207**	1344	1365	1374