HERRAMIENTA PARA LA EJECUCIÓN DE ALGORITMOS DE APRENDIZAJE REFORZADO APLICADOS A LA SOLUCIÓN DE PROBLEMAS DE SECUENCIACIÓN

Juliett M. Suárez Ferreira (jsf@uclv.edu.cu)[1]
Víctor M. García Martínez (vgmartinez@uclv.edu.cu)[1]
Yailén Martínez Jiménez (yailenm@uclv.edu.cu)[1]
Rafael Bello Pérez (rbellop@uclv.edu.cu) [1]

[1]Universidad Central "Marta Abreu" de Las Villas, Cuba.

RESUMEN: Este artículo presenta un software que utiliza el Aprendizaje Reforzado (AR) como vía de solución a problemas de secuenciación. Estos problemas de optimización, en los que se tienen que asignar operaciones a un conjunto de máquinas paralelas de diferentes tipos, son difíciles de resolver por lo que se tratan con este enfoque basado en AR. El objetivo es minimizar el tiempo final de procesamiento (*makespam*) considerando que los recursos son agentes inteligentes que tienen que elegir cuál operación van a realizar. Se utilizó el *Q-Learning* como método del AR y se implementaron dos variantes de este algoritmo, facilidad que muestra el software realizado para ejecutar ambos. Los resultados obtenidos por la ejecución de los algoritmos son comparativamente superiores a los obtenidos por otros enfoques recientes publicados en la literatura. El software fue implementado en Java.

Palabras Clave: Aprendizaje Reforzado, Q-Learning, Secuenciación.

ABSTRACT: This article presents a software that uses the Reinforcement Learning (RL) as a solution to scheduling problems. These optimization problems, in which operations are to be assigned to a set of parallel machines of different types, are difficult to solve so treated with this approach based on RL. The objective is to minimize the processing time (*makespam*) considering that resources are intelligent agents have to choose which operation to perform. We used the *Q-Learning* as a method of RL and implemented two variants of this algorithm, showing the software easily made to run both. Results obtained by the implementation of the algorithms are comparatively higher than those obtained by other recent approaches in the literature. The software was implemented in Java.

Key Words: Reinforcement Learning, Q-Larning, scheduling.

INTRODUCCIÓN

Este trabajo es el resultado final de una investigación que propone el uso de del aprendizaje reforzado en la solución de problemas de secuenciación [1], [2] basado en las ideas de Gabel y Riedmiller [3] que demuestran que interpretar y resolver el problema de secuenciación de tareas (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 [4], [5].

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 métodos del Aprendizaje Reforzado (*Reinforcement Learning*, RL a partir de ahora) 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 es aprender qué hacer con el objetivo de maximizar una señal de recompensa. Al aprendiz no se le dice qué acción debe tomar, 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 [6].

Entre los problemas resueltos por RL se encuentra el Job Shop Scheduling Problem (JSSP) [2], [3]. El mismo 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, de manera 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.

El JSSP con máquinas paralelas (JSSP-PM) consiste en la asignación de una operación a un recurso de un conjunto de máquinas paralelas candidatas (subproblema de asignación) en adición al JSSP clásico, donde las operaciones deben ordenarse en cada recurso con el objetivo de minimizar el tiempo total de producción (subproblema de secuenciación). El conjunto de máquinas paralelas candidatas es conocido como tipo de máquina. En oposición al clásico JSSP donde sólo hay un recurso para cada tipo de máquina, en los sistemas de manufactura existe un número de máquinas paralelas disponibles con el objetivo de aumentar el rendimiento y evitar que la producción se detenga cuando una máquina falla o se le hace mantenimiento a alguna de ellas. Este problema también ha sido resulto por el RL [1].

El JSSP-PM se diferencia del JSSP en que va a tener un conjunto de máquinas de un mismo tipo que van a realizar la misma labor, por lo que una vez terminada una operación tiene que decidirse cuál operación va a ser la próxima en realizarse y en cuál de las máquinas que la pueden realizar es que se va a ejecutar, lo que aumenta las posibilidades de elección. Para la solución de ambos problemas se ha utilizado el algoritmo *Q-Learning*, proponiéndose dos variantes del mismo en dependencia de la localización de los agentes del Aprendizaje Reforzado con respecto a las máquinas de los problemas de secuenciación.

En este trabajo se presenta un software que facilita la ejecución de los algoritmos implementados permitiendo variar los parámetros del algoritmo a consideración del usuario. El software muestra los resultados de los algoritmos posibilitando el análisis de las soluciones propuestas.

El estudio del Aprendizaje Reforzado y la factibilidad de su aplicación para encontrar una solución a los problemas de secuenciación de tareas, 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. Este software posibilita la visualización de los resultados de los algoritmos en forma de matriz que muestra el orden a seguir por los trabajos en las máquinas lo que permite un apoyo a la toma de las decisiones óptimas en problemas de secuenciación.

DESARROLLO

Los problemas JSSP y JSSP-PM

Los problemas de secuenciación, como ejemplo de problemas de Optimización Combinatoria, consisten en la localización o asignación de recursos en el tiempo a un conjunto de tareas o actividades. Dentro de ellos, aparecen los problemas de secuenciación Job Shop Scheduling Problem (JSSP) y secuenciación en máquinas paralelas Job Shop Scheduling Problem – Paralell Machines (JSSP-PM).

En JSSP se dispone de un conjunto de máquinas y un conjunto de trabajos con el objetivo de encontrar la mejor secuenciación de los trabajos en las diferentes máquinas, de forma tal que se minimice el tiempo total de producción, denominado *makespan*. En JSSP-PM se tienen 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 un conjunto de n trabajos que deben ser procesados en cada una de estas máquinas en un orden determinado. Su objetivo es encontrar una secuencia factible que minimice el tiempo de procesamiento, es decir, el tiempo necesario para completar todos los trabajos.

Las restricciones de estos problemas se muestran en la Tabla 1:

Tabla I: Restricciones de los problemas JSSP y JSSP-PM

JSSP	JSSP-PM
No se pueden procesar dos operaciones del mismo trabajo simultáneamente. Cada trabajo debe ser procesado hasta terminarse. Los trabajos deben esperar que la próxima máquina en su orden de procesamiento esté disponible. Ninguna máquina puede procesar más de una operación a la vez. Las máquinas pueden estar inactivas durante el proceso de secuenciación. El orden de procesamiento de cada trabajo es conocido de antemano y es inviolable.	
Ningún trabajo se procesa dos veces en la misma máquina. Existe solo una máquina de cada tipo.	Ningún trabajo se procesa dos veces en el mismo tipo de máquina ni en la misma máquina. Existen k máquinas de cada tipo.
El objetivo de encontrar la mejor secuenciación de los trabajos en las diferentes máquinas de forma tal que se minimice el tiempo total de producción.	

Aprendizaje Reforzado

En el paradigma del aprendizaje reforzado un agente se conecta a su ambiente mediante la percepción y acción, como lo descrito en la Figura 1. En cada paso de la interacción, el agente se da cuenta del 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 [6].

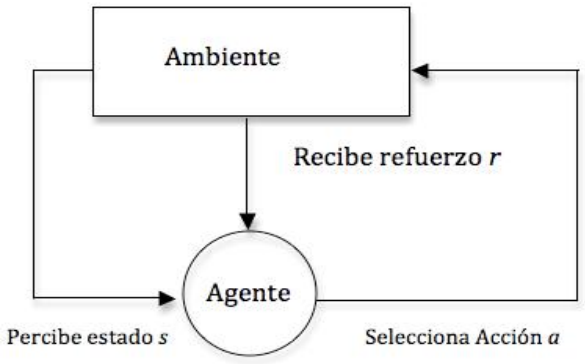


Figura 1. Paradigma del Aprendizaje Reforzado

Uno de los retos del aprendizaje reforzado es el intercambio 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.

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 posibles acciones $A(s_t)$. 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 $s_t = s$, en otras palabras 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.

Q-Learning

Q-learning es un algoritmo de aprendizaje reforzado que trabaja aprendiendo con una función de acción-valor, que da la utilidad esperada de tomar una acción dada en un estado determinado. El centro del algoritmo es la actualización de un Q-valor en cada iteración. Cada par (s, a) tiene un Q-valor asociado ("s" es un estado del conjunto de estados S, y "a" es una acción del conjunto de acciones A). Cuando la acción "a" es seleccionada por el agente que se encuentra en el estado "s" el Q-valor para ese par estado-acción es actualizado en base a la recompensa recibida cuando se seleccionó esa acción y el mejor Q-valor para el 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 ($0 \leq \gamma < 1$). 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'
```

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

```
    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 *ϵ-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.

La aplicación brinda la posibilidad de personalizar los parámetros explicados con anterioridad a consideración del usuario (Figura 2).

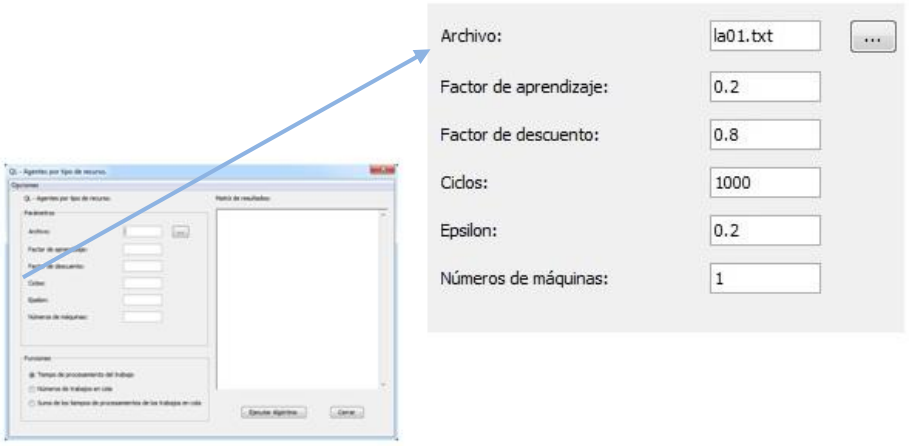


Figura 2. Parámetros del Q-Learning

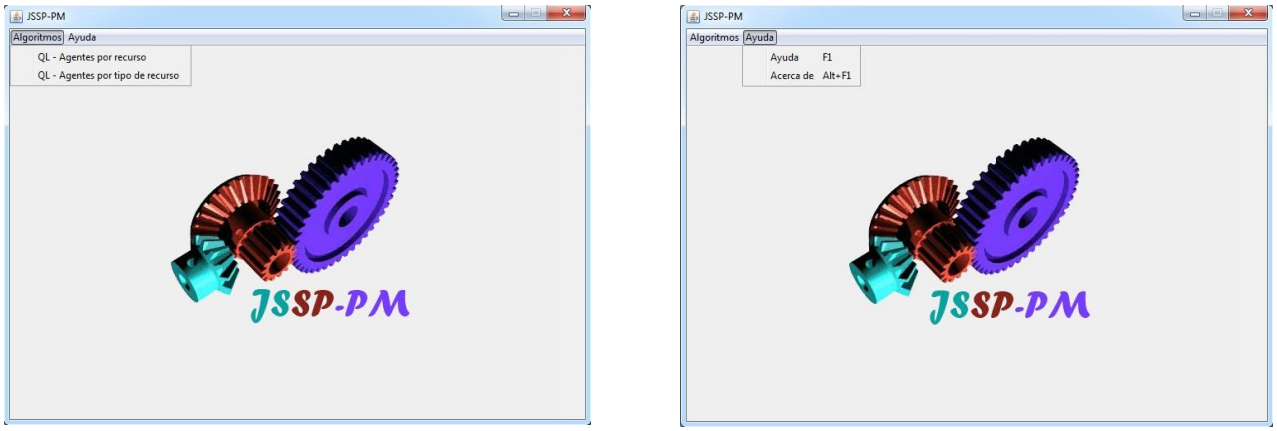
Aplicación del Q-Learning al JSSP y al JSSP-PM

Cuando se aplica aprendizaje reforzado para resolver el JSSP, un agente es asociado a cada uno de los m recursos (máquinas). Este agente decide localmente las acciones elementales. Para un agente tomar una acción, significa decidir cuál trabajo va a ser procesado, del conjunto de trabajos que se encuentran esperando por el recurso correspondiente.

Al aplicar aprendizaje reforzado para resolver el JSSP-PM, un agente es asociado a cada uno de los m tipos de recursos (tipos de máquinas) en una variante, y en otra el agente es asociado a cada máquina. Este agente decide localmente las acciones elementales. Para un agente tomar una acción, significa decidir cuál trabajo va a ser procesado, del conjunto de trabajos que se encuentran esperando. Cada agente tiene una cola con los trabajos que están esperando para ser

procesados. 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.

La aplicación en su ventana principal (Figura 3(a)) brinda la posibilidad de seleccionar que algoritmo va a utilizarse.



(a) (b)

Figura 3. Ventana Principal. (a) Algoritmos. (b) Ayuda

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.

La aplicación del Q-Learning a los problemas de secuenciación se realiza siguiendo 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*.

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

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 el número de trabajos que se encuentra esperando en la cola. 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. 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. La herramienta implementada permite la selección de la función de costo (Figura 4).

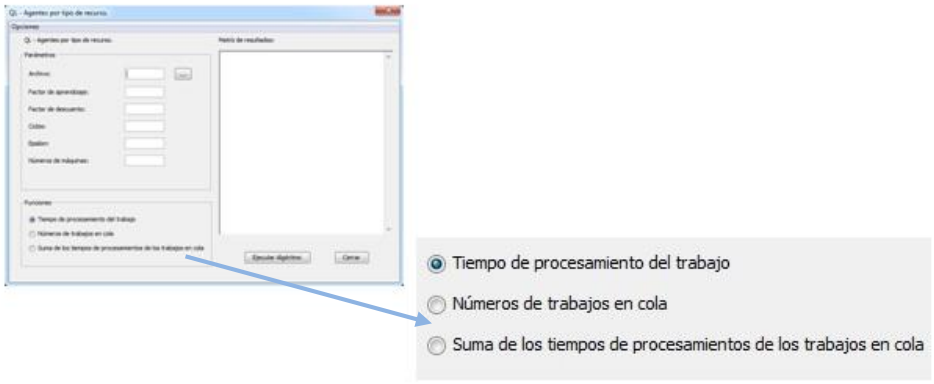


Figura 4: Funciones de costo.

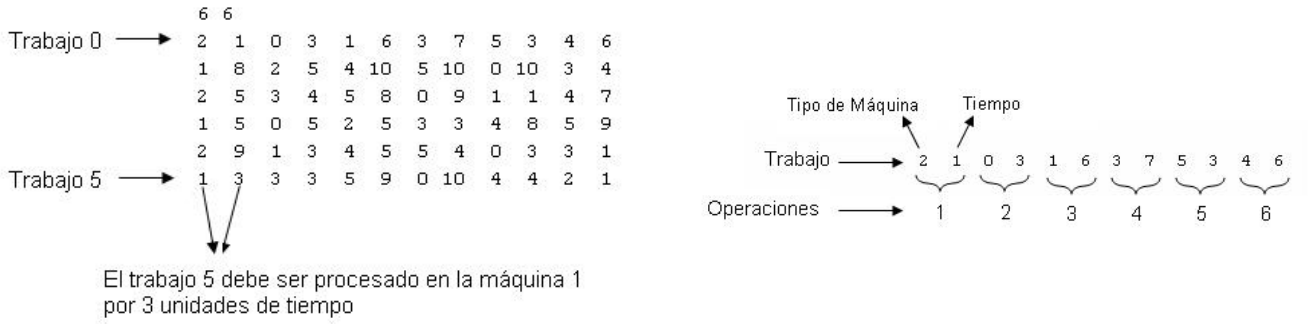
En el caso de la aplicación del Q-Learning al JSSP-PM aunque se tienen dos variantes de solución, cuando el número de máquinas paralelas es 1 se presenta el caso del JSSP, por lo tanto para ambas variantes la colocación de los agentes en el medio no difiere para el JSSP.

Instancias de los problemas de secuenciación.

Para la ejecución de los algoritmos implementados se usarán los formatos de las instancias que se proponen en [7], las cuales se basan en las instancias clásicas del Job Shop Scheduling que se encuentran disponibles en la Librería de Investigación de Operaciones [7].

Las instancias de la OR-Library son archivos con extensión .txt y 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(a) se observa el ejemplo de la instancia ft06.



(a) (b)

Figura 5: (a) Instancia ft06. (b) Descripción de un trabajo

En la Figura 5(b) se puede observar la descripción de un trabajo tal y como aparecen en las instancias.

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. Estas instancias son hechas para el JSSP, pero para el JSSP-PM se k-plica los datos de las instancias en dependencia del número de máquinas paralelas que se tendrá.

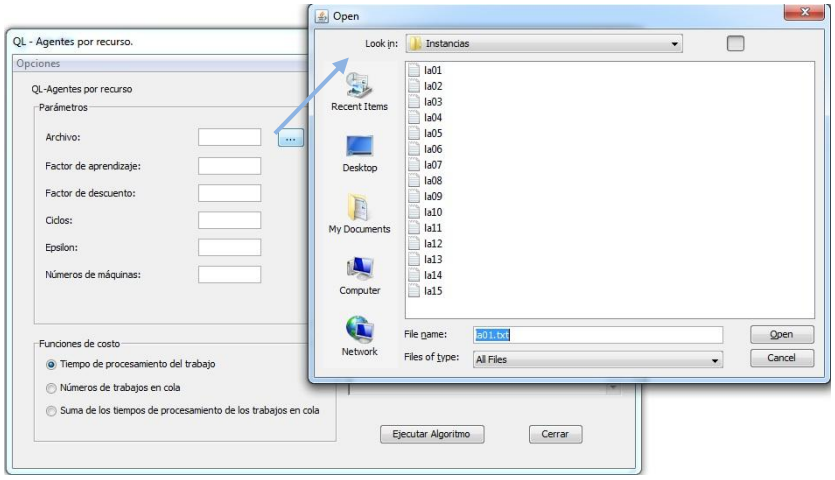


Figura 6: Cargar Instancias

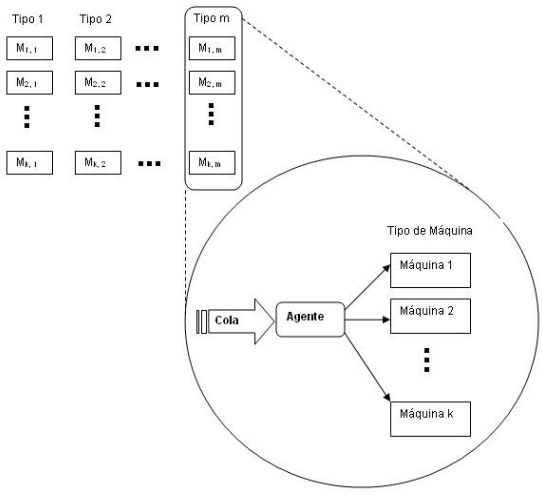
En la implementación que se realiza, el primer paso es obtener los datos necesarios de la instancia del problema a resolver y crear dos estructuras de datos principales, una que almacena el orden en el que los trabajos necesitan ser procesados y la otra con los correspondientes tiempos de procesamiento. Una vez almacenada la información, los agentes del Q-Learning comenzarán a seleccionar sus próximas acciones.

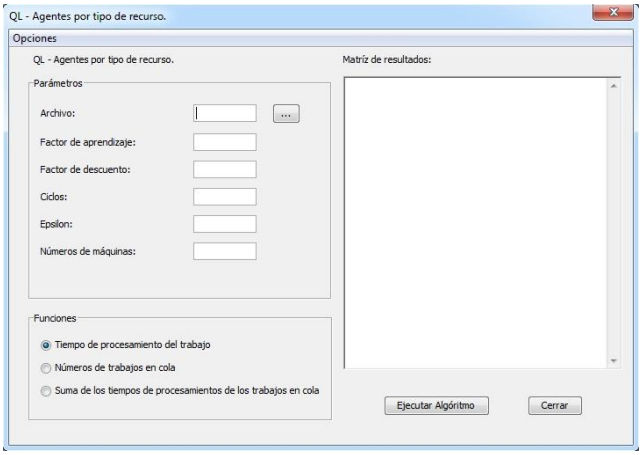
La figura anterior muestra la facilidad de la aplicación que permite cargar el archivo de la instancia seleccionada para ejecutar el algoritmo.

QL con agentes por tipo de recurso

Para el JSSP basta seleccionar como 1 el número de máquinas paralelas y un agente será colocado en cada recurso decidiendo sobre este el conjunto de acciones que debe seguir. Para almacenar los q-valores se decidió construir una matriz de m filas y n columnas ya que para cada uno de los m recursos hay n posibles trabajos a realizar (el número de filas es el número de agentes y el número de columnas es el número de trabajos).

Para el JSSP-PM 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.





(a) (b)

Figura 7: (a) Agente por tipo de recurso. (b) Ventana de ejecución del QL con agentes por tipo de recurso.

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 9 (a)).

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.

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.

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).

En la Figura 10 se observa la ventana de la aplicación que permite la ejecución del algoritmo donde se pueden seleccionar los parámetros a utilizar. A la derecha se muestra la matriz de resultados una vez corrido el algoritmo.

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.

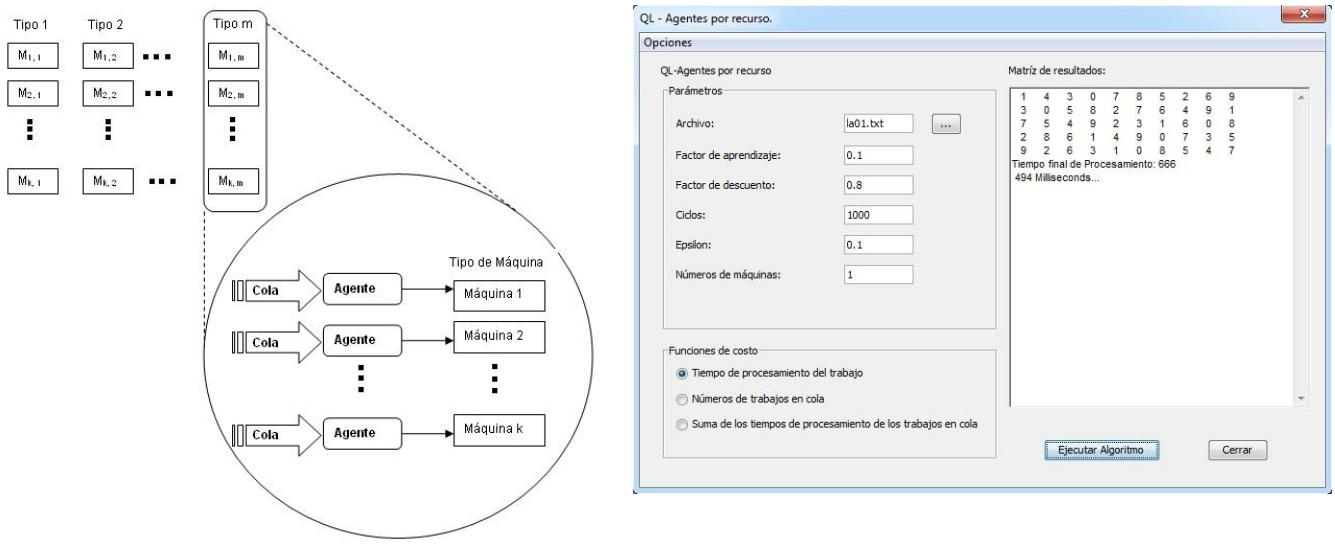
Para el caso de k = 1 de igual forma que en la variante anterior el problema es el JSSP clásico por lo que se van a tener tantos agente como recursos existan en el problema.

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

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.

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.

Unas 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.



(a) (b)

Figura 8: (a) Agentes por recurso. (b) Ventana de ejecución del QL con agentes por recurso.

La Figura 12 muestra la ventana donde puede ejecutarse el algoritmo que coloca a los agentes en cada uno de los recursos. A la derecha se muestra la matriz de los resultados obtenidos una vez corrido el algoritmo, mostrando además, el tiempo máximo de procesamiento (parámetro a optimizar) y el tiempo que demora la ejecución del algoritmo.

RESULTADOS Y DISCUSION

Los algoritmos incluidos en la aplicación han brindado resultados buenos comparados con otros enfoques de solución a los problemas planteados que se han encontrado en la literatura.

En [1] y [2] se puede encontrar los resultados de la aplicación del *Q-Learning* para cada problema utilizando los juegos de datos de OR-Library [7], en específico las 15 primeras instancias Lawrence utilizando los parámetros $\alpha = 0.1$, $\gamma = 0.8$ y epsilon = 0.2 fueron los de mejor comportamiento experimental. Estos resultados que se muestran competitivos con lo reportado en la literatura proveen a ésta aplicación de un método efectivo para la solución de problemas de secuenciación.

Hasta el momento la aplicación ha sido utilizada solamente con fines investigativos; pero pudiera ser fácilmente utilizada en la industria para la planificación y optimización de tareas en una línea de producción, por poner un ejemplo, siempre que los datos de entrada sean proporcionados con el formato adecuado.

La aplicación posee una ayuda que provee al usuario los conocimientos básicos sobre los algoritmos incluidos en el software y el problema que resuelve. Puede accederse a esta ayuda a partir de la ventana principal de la aplicación (Observe la Figura 3(b)).

CONCLUSIONES

La aplicación permite la ejecución de dos variantes del algoritmo Q-Learning del Aprendizaje Reforzado para la solución de los problemas de secuenciación JSSP y JSSP-PM.

La herramienta facilita la experimentación basada en la modificación de los parámetros de los algoritmos.

El software muestra los resultados en forma de matriz (Observe Figura 8(b)), de manera que es fácil para el usuario determinar cuál es la secuencia óptima a seguir por los trabajos.

El resultado mostrado incluye el tiempo final de procesamiento y el tiempo que demora la ejecución del algoritmo.

REFERENCIAS BIBLIOGRÁFICAS

1. Suárez Ferreira, J.. "Solución al problema de secuenciación en máquinas paralelas utilizando Aprendizaje Reforzado", Tesis de Maestría. Dept. de Computación. Facultad de Matemática Física y Computación. Universidad Central "Marta Abreu" de Las Villas, Santa Clara, Cubba, 2010.
2. Jiménez, Y.M., "A Multi-Agent Learning Approach for the Job Shop Scheduling Problem", Master thesis, Department of Computer Science, Computational Modeling Lab, Vrije Universiteit Brussel, Bruselas, 2008.
3. Gabel, T. and M. Riedmiller. (2007). On a Successful Application of Multi-Agent Reinforcement Learning to Operations Research Benchmarks. Presentada en: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning. 2007.
4. Puris, A., et al. (2007) Two-Stage ACO to solve the Job Shop Scheduling Problem. *Lectures and Notes in Computer Science*, 4756(2008): p. 447-456.
5. Rossi, A. and E. Boschi (2009). A hybrid heuristic to solve the parallel machines job-shop scheduling problem. *Advances in Engineering Software*, 2009. 40(2009): p. 118-127.
6. Sutton, R. and A. Barto. *Reinforcement Learning: An introduction*, ed. M. Press. 1998, Cambridge, MA.
7. OR-Library, http: //people.brunel.ac.uk/ ~ mastjjb/jeb/orlib/jobshopinfo.html.