Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 175-188

Tipo de artículo: Artículo original
Temática: Inteligencia Artificial
Recibido: 09/09/2016 | Aceptado: 25/04/2017

Enfoque bi-objetivo basado en Aprendizaje Reforzado para Job Shop scheduling

Bi-objective approach based in Reinforcement Learning to Job Shop scheduling

Beatriz M. Méndez-Hernández [1*], Liliana Ortega-Sánchez [1], Erick D. Rodríguez-Bazán [1], Yailen Martínez-Jiménez [1], Yunior C Fonseca-Reyna [2]

[1] Universidad Central "Marta Abreu" de Las Villas. Carretera a Camajuaní km. 5 ½ . Santa Clara, Villa Clara, Cuba.
[2] Universidad de Granma. Carretera de Manzanillo Km 17½. Granma, Cuba.

[*] Autor para correspondencia: bmendez@uclv.edu.cu

Resumen

Los problemas de secuenciación de tareas requieren organizar en el tiempo la ejecución de tareas que comparten un conjunto finito de recursos, y que están sujetas a un conjunto de restricciones impuestas por diversos factores. Este tipo de problemas aparecen con frecuencia en la vida real en numerosos entornos productivos y de servicios. El problema consiste en optimizar uno o varios criterios que se representan mediante funciones objetivo. En este artículo se analizaron los problemas de secuenciación tipo Job Shop con los principales objetivos a optimizar para este tipo de problemas, seguidamente se propuso un algoritmo desde un enfoque bi-objetivo basado en la Frontera de Pareto y utilizando Aprendizaje Reforzado para optimizar dos de los objetivos analizados, el tiempo de terminación de todos los trabajos y la suma del tiempo de finalización de todos los trabajos, y se aplicó a un conjunto de instancias de prueba. Por último, se describen los resultados satisfactorios obtenidos de acuerdo a dos métricas propuestas en la literatura para la evaluación de algoritmos bi-objetivo.

Palabras clave: Job Shop, multi-objetivo, Pareto, Aprendizaje Reforzado.

Abstract

Scheduling problems require organizing the execution of tasks which share a finite set of resources, and these tasks are subject to a set of constrains imposed by different factors. This kind of problems frequently appears in many production and service environments. The problem is to optimize one or more criteria represented by objective functions. In this paper, the main objectives to optimize were analyzed for Job Shop scheduling problems. After that, a

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 175-188

bi-objective algorithm was proposed based on the Pareto Front and using Reinforcement Learning, which optimizes two objectives: the makespan and the total flow time, and this algorithm was applied to benchmarks. To finish, successful results of the algorithm are described according to two metrics proposed in the literature.

Keywords: *Job Shop, multi-objective, Pareto, Reinforcement Learning.*

Introducción

Los problemas de secuenciación de tareas consisten en la programación temporal de las operaciones o tareas en las que se descomponen un conjunto de trabajos teniendo en cuenta que éstas deben ser ejecutadas en varias máquinas y que cada máquina solamente puede ejecutar una tarea simultáneamente. Se busca aquella solución que dé lugar a un tiempo total de ejecución (C_{max}) mínimo, aunque pueden existir otras funciones objetivo.

El Job Shop Scheduling Problem (JSSP) consta básicamente de un grupo de trabajos, donde cada uno tiene un conjunto de operaciones a ser procesadas en un conjunto de recursos limitados, a los que denominaremos máquinas. Cada trabajo tiene un orden en el que se deben ejecutar las operaciones, dichas operaciones tienen un tiempo de procesamiento en cada una de las máquinas y este no es modificable. Se trata de uno de los problemas de optimización combinatoria más difíciles de resolver. No sólo es del tipo NP-Completo, sino que de entre los que pertenecen a esta tipología, es de los más difíciles de resolver.

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

Uno de los paradigmas más usados ha sido el de los algoritmos evolutivos, principalmente los algoritmos genéticos, de estos se pueden citar varios ejemplos que han mostrado soluciones satisfactorias como son el Vector Evaluated Genetic Algorithm (VEGA) (Schaffer, 1985), el Multi-objective Genetic Algorithm (MOGA) (Fonseca and Fleming, 1993), el Niche Pareto Genetic Algorithm (NPGA) (Zitzler et al., 1999), el Pareto-Archived Evolutionary Strategy (PAES) (Knowles and Corne, 2000a), Pareto converging genetic algorithm (PCGA) (Kumar and Rockett, 2002) y por último el Modified micro Genetic Algorithm (MmGA) (Jun Tan et al., 2015).

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 175-188

También se encuentran algunas meta-heurísticas enfocadas a resolver problemas multi-objetivo como son Multi-objective Tabu Search (MOTS) (Hansen, 1997), Pareto Simulated Annealing (PSA) (Czyżak and Jaszkiewicz, 1997), Memetic Pareto Archived Evolutionary Strategy (M-PAES) (Knowles and Corne, 2000b), entre otros (Silva et al., 2004). También la Optimización basada en colonia de hormigas (ACO) ha sido utilizada para la optimización multi-objectivo en problemas Job Shop (Rudy, 2014).

Recientemente el Aprendizaje Reforzado (RL por sus siglas en inglés) ha recibido considerable atención. En (Gabel and Riedmiller, 2007) y (Gabel, 2009), los autores sugieren y analizan la aplicación de RL para resolver problemas de secuenciación de tipo Job Shop. En estos trabajos se demuestra que interpretar y resolver este tipo de escenarios a través de sistemas multi-agente y RL es beneficioso para obtener soluciones cercanas a las óptimas y puede muy bien competir con enfoques de solución alternativos.

En este artículo se presenta un nuevo algoritmo, llamado MOQL, el cual optimiza el tiempo de terminación de todos los trabajos y la suma del tiempo de terminación de todos los trabajos. Este algoritmo está basado en el algoritmo Q-Learning y en la Frontera de Pareto.

En la primera sección de este artículo se describen algunos de los objetivos que se persiguen optimizar en este tipo de problemas y a continuación en la próxima sección se introduce un nuevo enfoque bi-objetivo que optimiza dos de los principales objetivos descritos anteriormente usando Aprendizaje Reforzado y basado en la Frontera de Pareto. Por último, se muestra a través de pruebas experimentales, utilizando dos métricas propuestas en la literatura, como nuestro enfoque es capaz de competir con un enfoque basado Recocido Simulado propuesto en (Suresh and Mohanasundaram, 2006).

Objetivos a optimizar en un JSSP

Los objetivos en problemas de este tipo pueden tomar diversas formas, por ejemplo, la minimización del tiempo de fin de la última tarea (éste es el criterio de optimización más típico y se le conoce con el nombre de makespan), u otra posible función objetivo pueden ser el flow time total (que no es más que minimizar la suma de los tiempos de finalización de todos los trabajos).

A continuación se listan una serie de objetivos que pueden ser optimizados en problemas de secuenciación de tareas tipo Job Shop (Rivera and Eléctrica, 2004) .

- makespan $C_{máx} = máx_{i\in\{1..n\}} C_i$. Máximo tiempo de completamiento, es equivalente al tiempo en que el último trabajo abandona el sistema.

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 175-188

- flow time total $F_\Sigma = \Sigma_{i\in\{1..n\}} F_i$. Es el tiempo consumido por todos los trabajos.
- lateness total $L_\Sigma = \Sigma_{i\in\{1..n\}} L_i$. La suma de todos los retrasos de los trabajos.
- tardiness total $T_\Sigma = \Sigma_{i\in\{1..n\}} T_i$. La suma de todas las tardanzas de los trabajos.
- earliness total $E_\Sigma = \Sigma_{i\in\{1..n\}} E_i$. La suma de todos los tiempos de terminación previos al tiempo comprometido.
- maximum lateness $L_{máx} = máx_{i\in\{1..n\}} L_i$. El retraso del más atrasado de los trabajos.
- maximum tardiness $T_{máx} = máx_{i\in\{1..n\}} T_i$. La tardanza del más tardado de los trabajos En esta tesis nos centraremos en los tres de los que hablamos antes: el makespan, el tardiness y el flow time, dado que estos son los que con mayor frecuencia se desean optimizar en ambientes de manufactura.

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

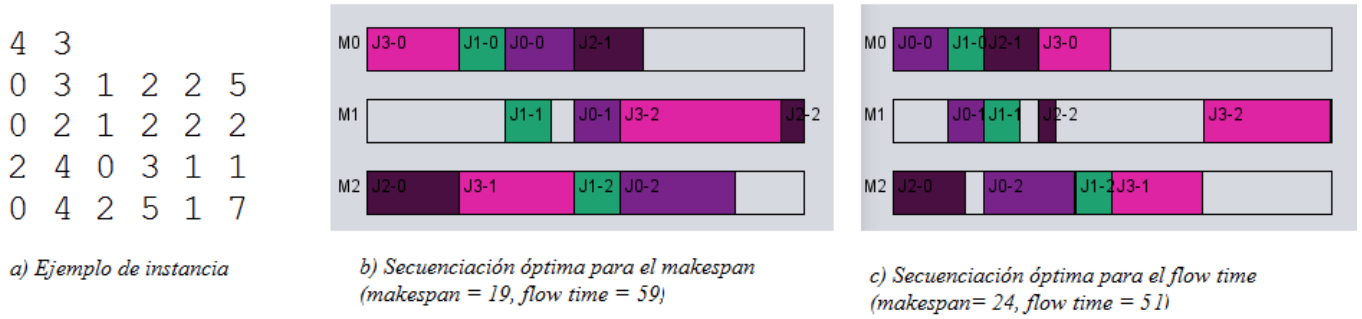
Esta minimización generalmente garantiza una alta utilización de los recursos de producción, una rápida satisfacción de la demanda de los clientes y la reducción de inventarios en curso.

El flow time total es otra función objetivo que puede tener mucho interés en problemas reales, por ejemplo, en aplicaciones en donde sea muy importante el servicio al cliente. Esta función no considera las fechas de entrega, sino que consiste en minimizar la suma del tiempo de finalización de todos los trabajos. Uno de los problemas con encontrados aquí es que apenas se pueden encontrar en la literatura métodos específicos para resolverla.

El flow time total se define como la suma de los tiempos de finalización de todos los trabajos, es decir, se pretende minimizar $\sum C_i$. este problema se denota $J|S_{ij}|\sum C_i$.

A continuación, se muestra mediante un ejemplo la diferencia entre el makespan y el flow time. El problema a resolver cuenta con 4 trabajos y 3 máquinas. En la figura se muestra los datos de la instancia utilizada además de las dos secuenciaciones óptimas obtenidas para el makespan (19) y el flow time (51) respectivamente.

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 175-188



a) Ejemplo de instancia

b) Secuenciación óptima para el makespan (makespan = 19, flow time = 59)

c) Secuenciación óptima para el flow time (makespan= 24, flow time = 51)

Figura 1. Ejemplo de la secuenciación óptima para el makespan y el flow time

Q-Learning

Un algoritmo muy conocido dentro del RL es el Q-Learning (QL) (Watkins, 1989), el cual se basa en aprender una función "acción-valor" que brinda la utilidad esperada de tomar una acción determinada en un estado específico. El centro del algoritmo es una simple actualización de valores, cada par (s;a) tiene un Q-valor asociado, cuando la acción a es seleccionada mientras el agente está en el estado s, el Q-valor para ese par estado-acción se actualiza basado en la recompensa recibida por el agente al tomar la acción. También se tiene en cuenta el mejor Q-valor para el próximo estado *s'*, la regla de actualización completa es la siguiente:

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

En esta expresión, α representa la velocidad del aprendizaje y "r" la recompensa o penalización resultante de ejecutar la acción "a" en el estado "s". La velocidad de aprendizaje "a" determina el 'grado' por el cual el valor anterior es actualizado. Por ejemplo, si α=0, entonces no existe actualización, y si α=1, entonces el valor anterior es reemplazado por el nuevo estimado. Normalmente se utiliza un valor pequeño para la velocidad de aprendizaje, por ejemplo α=0,1. El factor de descuento (γ) toma un valor entre 0 y 1 ($0 \leq \gamma \leq 1$), si está cercano a 0 entonces el agente tiende a considerar solo la recompensa inmediata, si está cercano a 1 el agente considerará la recompensa futura como más importante.

Q-L tiene la ventaja de que se ha demostrado que converge a la política óptima. Sin embargo, esto sólo es cierto para los Markov Decision Process (MDPs). El pseudocódigo de este algoritmo se muestra en la figura 2.

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

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 175-188

```
Inicializar Q(s,a) arbitrariamente
Repetir (por cada episodio)
Inicializar s
Repetir (para cada paso del episodio)
            Escoger a de s usando una política derivada de Q (e.g., e-
            greedy)
Tomar acción a, recompensa r, estado futuro s'
Q(s,a) ← Q(s,a) + α[r + γ máxQ(s',a') − Q(s,a)]
```

Figura 2. Pseudocódigo del algoritmo Q-Learning

Q-Learning aplicado a JSSP

En (Jiménez, 2012) se propone un enfoque basado en el algoritmo Q-Learning para JSSP pero desde un punto de vista mono-objetivo. Para introducir el Q-Learning al problema de secuenciación JSSP desde un enfoque multi-objetivo, existen elementos importantes que deben ser definidos y que pueden ser descritos como se muestra a continuación:

Estrategia de selección de la acción: Para seleccionar las acciones la estrategia que se usa es ε-Greedy, pues el uso de la misma evita caer en óptimos locales proporcionando mayor exploración del espacio de soluciones.

Q-Valores: De acuerdo a (Gabel, 2009) el conjunto de estados para el agente "*i*" se denota como $S_i = P(A_i^r)$, esto dará un total de $|S_i| = 2^n$ estados locales para cada agente "*i*", donde *n* es la cantidad de trabajos del problema a resolver. Debido a las restricciones de orden del problema, muchos de estos estados puede que nunca sean alcanzados. Por ello el MOQL solo almacena los estados que pueden ser alcanzados, es decir, la combinación de operaciones que pueden estar a la misma vez en las colas del sistema. Estas combinaciones se van almacenando a medida que aparecen. Por ejemplo, si el algoritmo es ejecutado por solo una iteración, entonces solo se guardarán *n* estados, que son los estados en los que el agente se encontraba cuando fueron seleccionadas las acciones. Otras ejecuciones pueden conllevar a la creación de otros estados.

Recompensa: El objetivo de este trabajo es lograr optimizar los dos objetivos a la vez, el makespan (C_{max}) que es la longitud de la planificación y el flow time que es la suma de los tiempos que demoraron en procesarse todos y cada

uno de los trabajos. La principal diferencia que se tuvo en cuenta entre estos objetivos a la hora de implementar el algoritmo Q-L es específicamente las funciones de recompensa.

En el caso del makespan:

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

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

Por otra parte, la actualización global se basa en dar una recompensa a partir del resultado final de un episodio, por lo que se modificará el Q-valor (calculado a partir de las actualizaciones locales explicadas anteriormente) de las acciones tomadas en este, teniendo en cuenta el beneficio real al que ellas conllevaron. Nótese que el hecho de que una acción en particular obtenga una buena señal de recompensa no es definitivo, ya que al finalizar el episodio el conjunto de acciones tomadas por los agentes puede haber resultado en una buena o mala secuencia, por tanto, se van a estimular o penalizar respectivamente las acciones involucradas. La forma de dar esta recompensa se definió como recompensa *1/makespan*, nótese que a medida que el makespan de la secuencia sea mayor, menor será la recompensa para la acción seleccionada. Cabe señalar entonces que el mayor Q-valor determina la mejor acción en un estado cualquiera.

En el caso del flow time:

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

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 175-188

operación es seleccionada se actualiza el estado local de la máquina en que se ejecutó dando 1/tiempo de procesamiento de la operación como recompensa.

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

A continuación, se muestra en la figura 3 el pseudocódigo del algoritmo propuesto.

Para cada objetivo:
  Inicializar:
    $Q(s,a) = \{\}$
Para cada episodio:
$s = \{\}$ $acciones\_tomadas = \{\}$
Mientras existan trabajos sin procesar:
  Por cada Agente (máquina)
    Inicializar s **como el conjunto de operaciones en cola.**
    Inicializar $acciones$ **como el conjunto de posibles operaciones a ejecutar.**
    Seleccionar a **dentro de** $acciones$ **usando una política ε-*greedy*.**
    Agregar a **a** $acciones\_tomadas$**.**
    Tomar acción a**, observar** s' **y** r **como la función de recompensa temporal de cada objetivo.**

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

$$s \leftarrow s'$$

  Para cada acción a **en** $acciones\_tomadas$**:**
    Actualizar todos los estados en que aparezca a **tomando como r la recompensa global correspondiente a cada objetivo.**

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

Para la secuencia obtenida calculo los objetivos restantes y de ser posible incluyo la solución en el Frente de Pareto

Repetir (para cada solución *i* en el frente de Pareto):
  Si *i* domina estricta o vagamente a *solución actual* salgo del episodio
  Si *solución actual* domina a *i*
    Eliminar *i* del Frente de Pareto
Adicionar *i* al Frente de Pareto

Figura 3. Algoritmo bi-objetivo para problemas de secuenciación de tareas tipo Job Shop

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 175-188

A continuación, se muestra un diagrama para una mejor ilustración del funcionamiento del algoritmo donde *i* es la iteración actual, *niters* y *mmaq* son el número de iteraciones y el número de máquinas respectivamente.

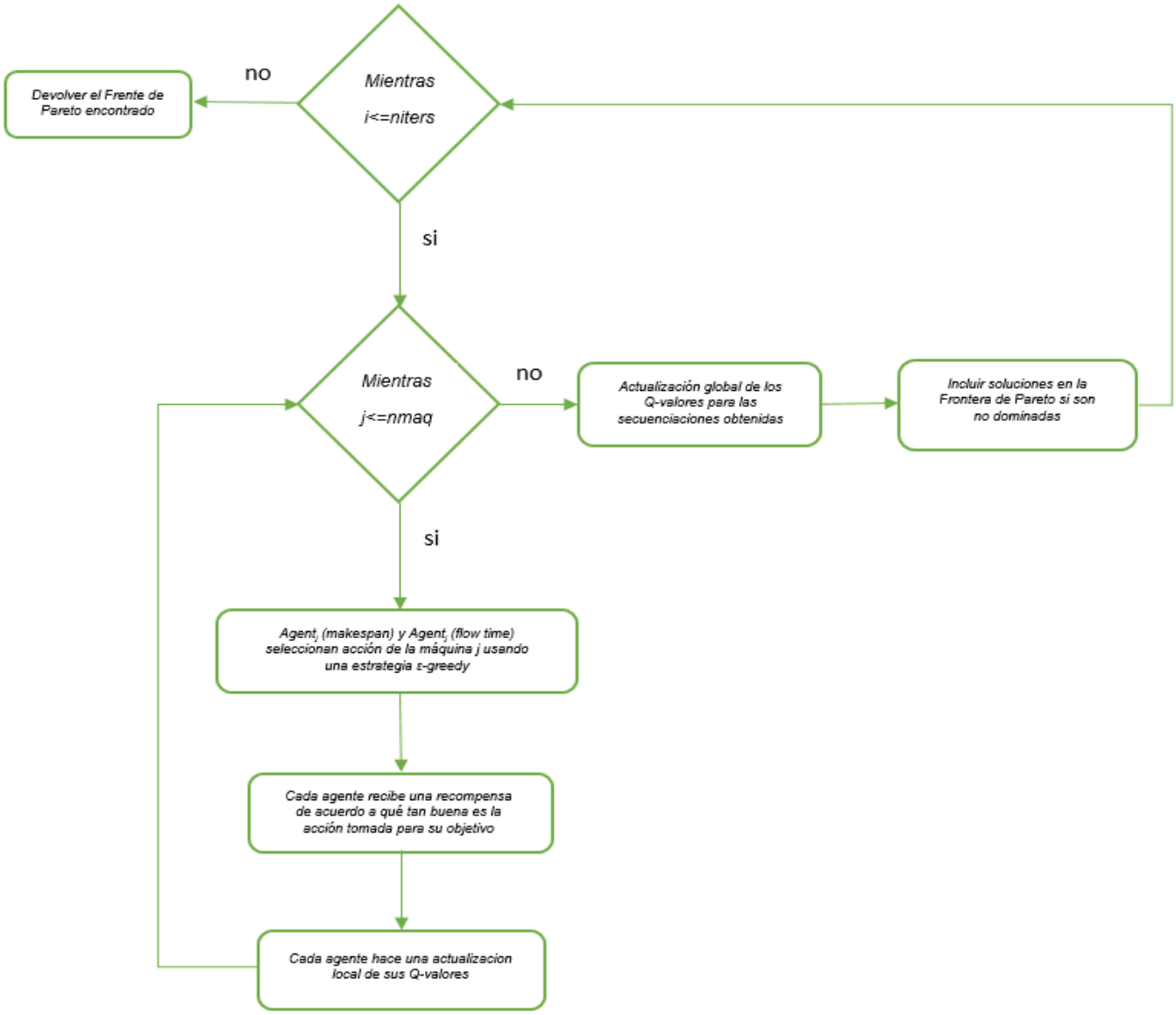


Figura 4. Diagrama del funcionamiento del algoritmo

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 175-188

Resultados experimentales

El algoritmo fue implementado en el lenguaje java JDK 1.7, como entorno de desarrollo (IDE) se utilizó NetBeans 7.4 y los experimentos fueron ejecutados en un procesador Intel Pentium IV a 3.40 GHz, con 3 GB de RAM. En el funcionamiento del algoritmo Q-L intervienen parámetros que pueden tomar diferentes valores:

- 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 α=0, no habrá actualización; si α=1 el valor antiguo es remplazado por el nuevo estimado. Usualmente es utilizado un valor pequeño para la proporción de aprendizaje, por ejemplo, α=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 γ=0.8 como un valor frecuentemente utilizado en el algoritmo.
- ε es un umbral que permite el balance entre la explotación y la exploración. Todas las acciones a realizar tienen asociadas una probabilidad generada aleatoriamente, si esta probabilidad está por debajo de ε se selecciona una acción aleatoria, de lo contrario se selecciona una acción de acuerdo con la política del agente. El valor 0.2 es usualmente utilizado para este parámetro en la literatura.
- El número de ciclos es el parámetro de parada del algoritmo.

El algoritmo fue aplicado instancias de diferentes tamaños de la biblioteca OR-Library disponible en Internet. Las instancias utilizadas cuentan con diferentes configuraciones. La instancia ft06 tiene 6 máquinas y 6 trabajos, las instancias la01-la05 tienen 5 máquinas y 10 trabajos, las últimas 5 instancias (la06-la10) tienen 10 trabajos y 10 máquinas. A estas instancias se le halló el Frente de Pareto y se compararon los resultados obtenidos de acuerdo a dos métricas propuestas en la literatura (Grosan et al., 2003, Knowles and Corne, 2002) con un algoritmo basado en la técnica de Recocido Simulado llamado *Pareto archived simulated annealing for Job Shop scheduling with multiple objectives* (PASA) propuesto en (Suresh and Mohanasundaram, 2006). Este genera una solución inicial aleatoria, luego usa un nuevo mecanismo de perturbación llamado *Segment-Random Insertion* (SRI) para generar un conjunto de soluciones ubicadas en la vecindad de la solución actual. Se agruparon las instancias de acuerdo al número de trabajos y de máquinas.

Por ciento de incremento medio relativo (MRPI)

Esta métrica mide la efectividad del frente de Pareto encontrado considerando las soluciones extremas (mejor *makespan* y mejor *flow time* encontrado) como referencia. Para calcular esta métrica se obtiene el error medio relativo

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 175-188

como la diferencia entre lo alcanzado por nuestro algoritmo y la mejor cota superior reportada en la literatura, esta diferencia se divide entre la cota y el resultado se multiplica por 100.

En la tabla siguiente se muestra los resultados obtenidos al evaluar esta métrica a los resultados obtenidos por los dos algoritmos analizados.

Tabla 1. MRPI obtenido al aplicar los algoritmos a las distintas instancias

	MRPI *Makespan*		**MRPI *Flow time***	
Instancias	MOQL	PASA	MOQL	PASA
ft06	0.00	0.00	0.00	0.00
la01-la05	1.994	0.00	0.108	0.16
la06-la10	0.00	0.00	0.668	0.17

Tabla 2. Resultados de la prueba de Wilcoxon entre los algoritmos PASA and MOQL

Algoritmos	**R+**	**R-**	**Sig.**
PASA-MOQL (Makespan)	3	0	0.180
PASA-MOQL (Flow time)	5	1	0.285

Al aplicar la prueba de Wilcoxon a estos resultados se obtuvo una significación en el caso del *makespan* de 0.180 y en el caso de *flow time* de 0.285, no siendo significativas ninguna de ellas, por lo que el comportamiento de los dos algoritmos de acuerdo a esta métrica es similar (ver Tabla 2).

Net front contribution ratio (NFCR)

Suponga que se quieren evaluar dos algoritmos A1 y A2, se tiene un frente a resultante del algoritmo A1 y un frente b obtenido mediante el algoritmo A2. Se forma entonces un frente c que está compuesto por la combinación de las soluciones de los frentes a y b que no se dominan entre ellas. Por tanto, se tienen n1 y n2 que son la cantidad de soluciones no dominadas del frente a y del frente b respectivamente que pasaron a formar parte del frente c. El NFCR de cada algoritmo se calcula mediante las siguientes fórmulas:

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

$$NFRC_{A2}=n2/n,$$

donde n es la cantidad de soluciones no dominadas que forman el frente c. En la tabla 3 se muestran los NFCR de los dos algoritmos para las instancias analizadas.

Al aplicar la prueba de Wilcoxon a estos resultados se obtuvo una significación de 0.905, lo cual implica que no existen diferencias significativas entre estos dos algoritmos de acuerdo a esta métrica tampoco.

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 175-188

Tabla 3. NFCR calculado para los resultados obtenidos por los dos algoritmos evaluados

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

Tabla 4. Resultados de la prueba de Wilcoxon para la métrica NFCR entre los dos algoritmos.

Algoritmos	**R+**	**R-**	**Sig.**
PASA-MOQL	4.70	23.50	0.905

Conclusiones

Se implementó un algoritmo bi-objetivo para la optimización del *makespan* y el *flow time* en problemas de secuenciación de tareas usando Aprendizaje Reforzado y con soluciones basadas en la Frontera de Pareto probando de esta forma que el Aprendizaje Reforzado es una buena elección para resolver también problemas multi-objetivo. Este enfoque no necesita una configuración inicial compleja como los algoritmos genéticos, pero sin embargo en las pruebas experimentales que se realizaron se demostró que nuestro enfoque brinda resultados tan buenos como uno de los mejores algoritmos multi-objetivo reportado en la literatura.

Referencias

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

FONSECA, C. M. & FLEMING, P. J. Multiobjective genetic algorithms. Genetic Algorithms for Control Systems Engineering, IEE Colloquium on, 1993. IET, 6/1-6/5.

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

GABEL, T. & RIEDMILLER, M. M.: Adaptive Reactive Job-Shop Scheduling with Learning Agents. International Journal of Information Technology and Intelligent Computing, 2007. Citeseer.

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

GROSAN, C., OLTEAN, M. & DUMITRESCU, D. Performance metrics for multiobjective optimization evolutionary algorithms. Proceedings of Conference on Applied and Industrial Mathematics (CAIM), Oradea, 2003.

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

JIMÉNEZ, Y. M. 2012. *A Generic Multi-Agent Reinforcement Learning Approach for Scheduling Problems.* Ph.D., VUB.

JUN TAN, C., HANOUN, S. & PENG LIM, C. A multi-objective evolutionary algorithm-based decision support system: A case study on job-shop scheduling in manufacturing. Systems Conference (SysCon), 2015 9th Annual IEEE International, 2015. IEEE, 170-174.

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

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

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

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

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

RUDY, J. 2014. Solving multi-objective job shop problem using nature-based algorithms: new Pareto approximation features. *An International Journal of Optimization and Control: Theories & Applications (IJOCTA),* 5**,** 1-11.

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

SILVA, J. D. L., BURKE, E. K. & PETROVIC, S. 2004. An introduction to multiobjective metaheuristics for scheduling and timetabling. *Metaheuristics for multiobjective optimisation.* Springer.

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

WATKINS, C. J. C. H. 1989. *:earning from Delayed Rewards.* Ph.D., Cambridge.

ZITZLER, E., DEB, K. & THIELE, L. Comparison of multiobjective evolutionary algorithms on test functions of different difficulty. Proceedings of the 1999 Genetic and Evolutionary Computation Conference. Workshop Program, 1999. Orlando, Florida, 121-122.