# PQ-learning: Aprendizaje por refuerzo multiobjetivo

Manuela Ruiz-Montiel, Lawrence Mandow, and José-Luis Pérez-de-la-Cruz

ETSI Informática, Universidad de Málaga
Bulevar Louis Pasteur nº35, 29071, Málaga, España
{mruiz,lawrence,perez}@lcc.uma.es

**Resumen** En este artículo describimos y analizamos PQ-learning, un algoritmo para problemas de aprendizaje por refuerzo multiobjetivo. El algoritmo es una extensión de Q-learning, un algoritmo para problemas de aprendizaje por refuerzo escalares. Al contrario que otros algoritmos, PQ-learning no requiere información de preferencias sobre los objetivos, es aplicable a problemas con fronteras de Pareto no convexas y permite recuperar a partir de los Q-valores las secuencias de acción correspondientes a diferentes políticas Pareto-óptimas. PQ-learning ha sido aplicado a dos problemas pertenecientes a un banco de pruebas propuesto en la literatura de aprendizaje por refuerzo multiobjetivo.

**Key words:** Aprendizaje por Refuerzo, Optimización Multiobjetivo, Q-learning

## 1. Introducción

En este trabajo presentamos PQ-learning, un nuevo algoritmo para determinar cómo un agente debe comportarse en un entorno de modo que se maximice la recompensa acumulada a lo largo del tiempo, donde tal recompensa es un vector $\overrightarrow{r} \in \mathbb{R}^n$, es decir, consta de un componente por cada objetivo a maximizar.

En este contexto, el entorno se especifica típicamente como un *proceso de decisión markoviano* (PDM), un formalismo usado para modelar la toma de decisiones en situaciones que presentan incertidumbre. La mayoría de las aproximaciones existentes trabajan con procesos de decisión markovianos donde la recompensa es escalar. Formalmente, un PDM escalar se define por un conjunto de estados $S$, un conjunto de acciones $A$ y por la *dinámica del entorno* (el *modelo*). Esta dinámica viene dada por dos funciones: (1) $P_{sa}(s')$, que es la probabilidad de alcanzar el estado $s'$ tomando la acción $a$ en el estado $s$, y (2) $R_{sa}(s') : S \times A \times S \to \mathbb{R}$, que es la recompensa obtenida al alcanzar el estado $s'$ escogiendo la acción $a$ en el estado $s$. La solución a un PDM escalar es una política de actuación $\pi : S \to A$ que indica qué acción tomar en cada estado, de manera que se maximice el valor de la recompensa acumulada a partir de un instante $t$, $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \Sigma_{k=0}^{\infty} \gamma^k r_{t+k+1}, 0 \leq \gamma \leq 1$. El factor de descuento $\gamma$ determina la importancia que el agente otorga a recompensas obtenidas en el futuro. Si $\gamma = 1$, tienen el mismo valor que las obtenidas inmediatamente; si $\gamma < 1$, las obtenidas inmediatamente tienen más peso.

Los métodos para obtener la política óptima de un PDM escalar se clasifican a grandes rasgos en: (1) programación dinámica, donde se asume un conocimiento perfecto de la dinámica del entorno y la política óptima se calcula de manera exacta, y (2) aprendizaje por refuerzo (AR), donde la política óptima se aproxima mediante un proceso de interacción con el entorno y por tanto no se asume un conocimiento perfecto del modelo. Un ejemplo de método de AR es el algoritmo Q-learning [1], ampliamente usado en la literatura, donde el agente va optimizando los valores $Q(s,a) : S \times A \to \mathbb{R}$. Dada una política, estos Q-valores representan la calidad de una combinación $s, a$, es decir, el valor esperado al seguir la política tras tomar la acción $a$ en el estado $s$. Formalmente, $Q(s,a) = E\{\Sigma_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\}$.

Muchos procesos de decisión markovianos del mundo real se formulan mejor en términos de optimización multiobjetivo, donde las recompensas son vectores de $n$ dimensiones, una por cada objetivo a maximizar. La solución a un PDM multiobjetivo es una política de actuación $\pi : S \to A$ que maximiza el valor de la recompensa acumulada a partir de un instante $t$, $\overrightarrow{R}_t = \Sigma_{k=0}^{\infty} \gamma^k \overrightarrow{r}_{t+k+1}$. Al no poder establecerse un orden total entre vectores, en este caso el resultado de optimizar la recompensa no es un único valor, sino un conjunto de valores *no dominados* denominado *frontera de Pareto*. Decimos que un vector $\overrightarrow{v}$ domina a un vector $\overrightarrow{w}$, $\overrightarrow{v} \succ \overrightarrow{w}$, cuando $\exists i : v_i > w_i \wedge \nexists j : v_j < w_j$, donde $v_i$ es la componente i-ésima de un vector $\overrightarrow{v}$. Dado un conjunto de vectores $Q$, su frontera de Pareto contiene aquellos vectores que no son dominados por ningún otro vector en $Q$, es decir, $\{\overrightarrow{v} \in Q : \{\overrightarrow{w} \in Q : \overrightarrow{w} \succ \overrightarrow{v}\} = \varnothing\}$.

La investigación dedicada a resolver procesos de decisión markovianos multiobjetivo es relativamente reciente. Las aproximaciones se clasifican en *single-policy* y *multi-policy* [2]. Los algoritmos single-policy averiguan una única política de actuación según cierta información de preferencias que permite establecer un orden total entre los valores. Por ejemplo, Gábor et al. [3] amplían un algoritmo de programación dinámica con un orden lexicográfico sobre los objetivos. Natarajan y Tadepalli [4] amplían técnicas de programación dinámica y de AR con escalarización lineal: se establece un peso por cada objetivo, y así puede obtenerse un único valor escalar (la suma ponderada de los objetivos) con el cual trabajar. Por su parte, los algoritmos multi-policy averiguan un conjunto de políticas cuyos valores acumulados pretenden aproximar la frontera de Pareto del problema de optimización multiobjetivo. Uno de los primeros representantes de este tipo de algoritmos es el propuesto por White [5], basado en programación dinámica. En este algoritmo se van manteniendo conjuntos de vectores no dominados en lugar de valores escalares. La propuesta de Barrett y Narayanan [6] es similar a la de White, pero mantiene envolventes convexas en lugar de conjuntos no dominados, lo cual proporciona ventajas en cuanto a eficiencia. Sin embargo, cuando la frontera de Pareto del problema es no convexa, esta propuesta no averigua todas las políticas. Más recientemente, Issabekov y Vamplew [7] han estudiado el comportamiento de Q-learning ampliado con información de preferencias lexicográficas o lineales. En general, la incorporación de preferencias a cualquier tipo de algoritmo de AR multiobjetivo permite aliviar el proceso de

aprendizaje, pero tiene algunos inconvenientes: (1) para especificarlas suele ser necesario cierto conocimiento del modelo, y (2) existen limitaciones en el uso de la escalarización lineal a la hora de averiguar fronteras no convexas [8].

En este trabajo proponemos un nuevo algoritmo de AR multiobjetivo, PQ-learning, basado en el algoritmo de AR escalar Q-learning [1], que pretende solventar los inconvenientes de las aproximaciones existentes. Aprende un conjunto de políticas que aproximan la frontera de Pareto (es multi-policy), sin asumir el conocimiento del modelo que suponen las técnicas de programación dinámica, ni ningún tipo de información de preferencias. Además, PQ-learning es aplicable a problemas con fronteras no convexas y, a diferencia de las aproximaciones anteriores, proporciona información suficiente para recuperar las secuencias de acción de diferentes políticas. Hasta donde sabemos, PQ-learning es el único algoritmo que reúne estas características.

PQ-learning ha sido aplicado a dos problemas pertenecientes a un banco de pruebas propuesto en la literatura de AR multiobjetivo [2]. Los resultados obtenidos se han comparado con los arrojados por una extensión de Q-learning que utiliza técnicas habituales en programación dinámica multiobjetivo.

La Sección 2 describe y analiza el algoritmo PQ-learning. La Sección 3 presenta algunos resultados empíricos del algoritmo. Finalmente, la Sección 4 recopila las conclusiones y apunta las líneas posibles de continuación de este trabajo.

## 2. Pareto Q-learning

En esta sección describimos el algoritmo propuesto, basado en el algoritmo Q-learning [1,9], donde la política aprendida $\pi$ viene dada por la expresión $\pi(s) = argmax_a Q(s,a)$. Q-learning se clasifica como un algoritmo *off-policy*, es decir, la política que utiliza para interactuar con el entorno es distinta de la política que aprende. Por esta razon partimos de Q-learning: como queremos aprender más de una política a la vez, tenemos que usar un algoritmo que utilice una política distinta de la aprendida para interactuar con el entorno [6]. En el Cuadro 1 se muestra el esquema propuesto para el algoritmo Q-learning multiobjetivo. A continuación detallamos algunas consideraciones:

1. El operador $ND$ devuelve la frontera de Pareto de un conjunto de vectores
2. Cada $Q(s,a)$ es un conjunto y no un valor escalar. Inicializamos cada $Q(s,a)$ al conjunto vacío (línea 1)
3. Cuando se decide explotar la tabla $Q$ (línea 5), con probabilidad $1-\epsilon$ escogeremos una acción de entre todas las correspondientes a los valores del conjunto $NDU(s) = ND(\bigcup_a (Q(s,a)))$. A cada una de estas acciones se le asocia una probabilidad de ser escogida, proporcional a la cantidad de valores asociados a ella en el conjunto $NDU(s)$. Con probabilidad $\epsilon$, escogeremos una acción aleatoriamente, para explorar el entorno (esta política de exploración se denomina $\epsilon$-greedy). El estado $s'$ es el estado al que conduce la acción escogida en este paso.
4. En Q-learning escalar, $Q(s,a)$ se actualiza con el valor $r + \gamma max_{a'} Q(s',a')$. En Q-learning multiobjetivo, el conjunto $Q(s,a)$ se actualiza con $\overrightarrow{r} + \gamma NDU(s')$. Si $s'$ es final, $Q(s,a)$ se actualiza con $\{\overrightarrow{r}\}$ (línea 7)

5. La traslación y escala de conjuntos no dominados (línea 7) se define como sigue: $\overrightarrow{u} + bQ = \{\overrightarrow{u} + bq : \overrightarrow{q} \in Q, Q \neq \varnothing\}$, $\overrightarrow{u} + \varnothing = \varnothing$, $bQ = \{b\overrightarrow{q} : \overrightarrow{q} \in Q\}$

En este esquema queda por definir el método utilizado para actualizar el valor estimado hasta el momento (línea 7), $Q(s, a)$, con la nueva estimación $\overrightarrow{r} + \gamma NDU(s')$. A continuación veremos, en primer lugar, una manera directa pero inviable de hacerlo, y en segundo, el método que da lugar al algoritmo propuesto PQ-learning.

### 2.1. Aproximación Ingenua: Actualización Exhaustiva

En Q-learning escalar, la actualización del valor $Q(s, a)$ con el valor $r + \gamma max_{a'} Q(s', a')$ se realiza mediante la expresión $Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(r + \gamma max_{a'} Q(s', a'))$, es decir, mediante una suma de valores escalares. En Q-learning multiobjetivo nos limitaremos de momento a sumar los conjuntos de valores no dominados $Q(s, a)$ y $\overrightarrow{r} + \gamma NDU(s')$. Cuando ambos conjuntos son no vacíos, la expresión de actualización es $Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(\overrightarrow{r} + \gamma NDU(s'))$. Sean $Q = (1 - \alpha)Q(s, a)$ y $U = \alpha(\overrightarrow{r} + \gamma NDU(s'))$ ($U = \alpha\{\overrightarrow{r}\}$ si $s'$ es final). En trabajos previos de programación dinámica multiobjetivo [5,6] la suma de conjuntos de valores no dominados, que denominaremos como *exhaustiva*, se define como sigue: $Q + U = ND\{\overrightarrow{q} + \overrightarrow{u} : \overrightarrow{q} \in Q, \overrightarrow{u} \in U\}$. Si $Q(s, a) = \varnothing$, la actualización se realiza mediante la expresión $Q(s, a) = \overrightarrow{r} + \gamma NDU(s')$ ($Q(s, a) = \{\overrightarrow{r}\}$ si $s'$ es final). Si $\overrightarrow{r} + \gamma NDU(s') = \varnothing$, $Q(s, a)$ no cambia.

Sin embargo, a continuación veremos un ejemplo del uso del esquema propuesto donde esta suma exhaustiva no da buenos resultados. En la Figura 1 podemos ver un ejemplo de PDM multiobjetivo. El agente comienza en el estado $s_0$ y ha de llegar al estado final $s_4$. La tasa de descuento $\gamma$ es igual a 1. En este modelo sólo hay dos políticas posibles que el agente puede seguir (ambas son óptimas): (1) la que en $s_1$ toma la acción $a_{11}$ (recompensa acumulada esperada: $(-3, 2)$), y (2) la que en $s_1$ toma la acción $a_{12}$ (recompensa acumulada esperada: $(-2{,}5, 1{,}5)$). En el resto de estados sólo hay una acción disponible, luego no cabe duda de cuál escoger. Esto quiere decir que al finalizar el proceso de aprendizaje, en la posición $Q(s_0, a_0)$ de la tabla debe haber dos vectores, correspondientes a las dos políticas anteriores.

En la primera columna del Cuadro 2 se muestra una sucesión de posibles episodios de aprendizaje de Q-learning multiobjetivo con actualización exhaus-

**Algoritmo**: Q-learning multiobjetivo

1. Inicializar todas las posiciones de la tabla Q a $\varnothing$
2. Repetir (para cada episodio):
3. Inicializar $s$
4. Repetir (para cada paso del episodio):
5. Elegir $a$ desde $s$, usando política $\epsilon$-greedy derivada de $Q$
6. Tomar acción $a$, observar $\overrightarrow{r}$ y $s'$
7. $Q(s, a) \leftarrow actualizar(Q(s, a), \overrightarrow{r} + \gamma NDU(s'))$
8. $s \leftarrow s'$
9. Hasta que $s$ sea terminal

**Cuadro 1.** Esquema propuesto para Q-learning con múltiples objetivos

tiva ($\alpha = 0{,}5$). La segunda columna muestra las posiciones de la tabla $Q$ que se se ven modificadas en cada secuencia. En la séptima fila (segunda columna), en el conjunto $Q(s_0, a_0)$, el vector $(-2{,}75, 1{,}75)$ ha surgido de actualizar $(-3, 2)$, que surgió al realizar la secuencia $s_0, a_0, s_1, a_{11}, s_2, a_2, s_4$ (tercera fila), con el vector $(-2{,}5, 1{,}5)$ proveniente de sumarle la recompensa al vector $(-1{,}5, 1{,}5)$ de $Q(s_1, a_{12})$, en la sexta fila. Es decir, se está *mezclando* la secuencia $s_0, a_0, s_1, a_{11}, s_2, a_2, s_4$ con la secuencia $s_0, a_0, s_1, a_{12}, s_3, a_3, s_4$.

En aprendizaje con un único objetivo, las posiciones de la tabla $Q$ no son conjuntos sino valores escalares. En tal situación, mezclar secuencias no da lugar a ningún problema: el hecho de actualizar un valor asociado a una secuencia determinada con el valor correspondiente a otra distinta, significa que la nueva secuencia es mejor que la anterior. Ese único valor irá convergiendo hasta su valor óptimo conforme avanzan los episodios de aprendizaje. Sin embargo, en aprendizaje multiobjetivo, donde las posiciones de la tabla Q son conjuntos, éstos pueden crecer de manera descontrolada si usamos la suma exhaustiva, haciendo el algoritmo inviable. Podemos apreciar este fenómeno en la octava fila del Cuadro 2. Vemos que aparecen dos nuevos vectores en $Q(s_0, a_0)$. Sabemos que en $Q(s_0, a_0)$ sólo tiene sentido que haya dos vectores, luego dos de los cuatro vectores sobran. Intuitivamente, parece lógico pensar que deberíamos quedarnos con el primero y

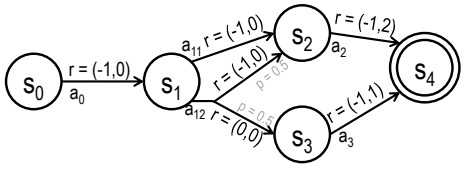


**Figura 1.** Ejemplo de PDM multiobjetivo

| **Entrenamiento** | **Act. exhaustiva** | **Act. sellos** |
|---|---|---|
| $s_0, a_0, s_1, a_{11}, s_2, a_2, s_4$ | $Q(s_2, a_2) = \{(-1, 2)\}$ | $Q(s_2, a_2) = \{(-1, 2)_{(-1,2)f3}\}$ |
| $s_0, a_0, s_1, a_{11}, s_2, a_2, s_4$ | $Q(s_1, a_{11}) = \{(-2, 2)\}$ | $Q(s_1, a_{11}) = \{(-2, 2)_{(-2,2)f3}\}$ |
| $s_0, a_0, s_1, a_{11}, s_2, a_2, s_4$ | $Q(s_0, a_0) = \{(-3, 2)\}$ | $Q(s_0, a_0) = \{(-2, 2)_{(-3,2)f3}\}$ |
| $s_0, a_0, s_1, a_{12}, s_2, a_2, s_4$ | $Q(s_1, a_{12}) = \{(-2, 2)\}$ | $Q(s_1, a_{12}) = \{(-2, 2)_{(-2,2)f3}\}$ |
| $s_0, a_0, s_1, a_{12}, s_3, a_3, s_4$ | $Q(s_3, a_3) = \{(-1, 1)\}$ | $Q(s_3, a_3) = \{(-1, 1)_{(-1,1)f3}\}$ |
| $s_0, a_0, s_1, a_{12}, s_3, a_3, s_4$ | $Q(s_1, a_{12}) = \{(-1, 5, 1, 5)\}$ | $Q(s_1, a_{12}) =$ $\{(-1{,}5, 1{,}5)_{(-2,2)(-1,1)f2},$ $(-1, 1)_{(-1,1)f3}\}$ |
| $s_0, a_0, s_1, a_{12}, s_3, a_3, s_4$ | $Q(s_0, a_0) =$ $\{(-3, 2), (-2{,}75, 1{,}75)\}$ | $Q(s_0, a_0) = \{(-3, 2)_{(-3,2)f1},$ $(-2{,}5, 1{,}5)_{(-2,5,1,5)f3}, (-2, 1)_{(-2,1)f3}\}$ |
| $s_0, a_0, s_1, a_{11}, s_2, a_2, s_4$ | $Q(s_0, a_0) = \{(-3, 2), (-2{,}75, 1{,}75),$ $(-2{,}875, 1{,}875), (-2{,}625, 1{,}625)\}$ | $Q(s_0, a_0) = \{(-3, 2)_{(-3,2)f1},$ $(-2{,}5, 1{,}5)_{(-2,5,1,5)f1}, (-2, 1)_{(-2,1)f1}\}$ |

**Cuadro 2.** Posibles trazas de Q-learning multiobjetivo con actualización exhaustiva (columna 2) y de PQ-learning (columna 3)

con el cuarto, ya que el primer vector se corresponde con la política 1, y el cuarto es el que se parece más a la política 2, ya que ha sido actualizado dos veces con el vector correspondiente a tomar la acción $a_{12}$. Es más, podríamos ahorrarnos las actualizaciones intermedias que llevarían desde el vector $(-3, 2)$ al vector $(-2{,}5, 1{,}5)$, puesto que $(-3, 2)$ corresponde a una secuencia distinta. Es decir, en la septima fila, donde actualizamos el conjunto $Q(s_0, a_0) = \{(-3, 2)\}$ con el conjunto $\overrightarrow{r} + \gamma NDU(s_1) = \{(-3, 2), (-2{,}5, 1{,}5)\}$, podríamos copiar directamente el vector $(-2{,}5, 1{,}5)$ en $Q(s_0, a_0)$, luego $Q(s_0, a_0) = \{(-3, 2), (-2{,}5, 1{,}5)\}$. El vector $(-3, 2)$ de $Q(s_0, a_0)$ se seguiría actualizando con el vector $(-3, 2)$ de $\overrightarrow{r} + \gamma NDU(s_1)$, luego no cambiaría.

### 2.2. PQ-learning: Actualización con Sellos

Proponemos una alternativa para definir la suma de dos conjuntos, $Q + U$, de manera que se mantengan únicamente los vectores correspondientes a las secuencias que tienen sentido. Para esto utilizaremos *sellos*, que determinan qué vector de $Q$ ha de actualizar cada vector de $U$. Un *vector sello* V es una terna $\langle \overrightarrow{v_V}, sello_{V\leftarrow}, sello_{V\rightarrow} \rangle$ donde $\overrightarrow{v_V} \in \mathbb{R}^n$, $sello_{V\leftarrow} : S \rightarrow \mathbb{N} \cup \{-1\}$, $sello_{V\rightarrow} : S \times A \rightarrow \mathbb{N} \cup \{-1\}$. La función $sello_{Q\leftarrow}$ de un vector sello $V_Q \in Q(s, a)$ indica, para cada estado $s'$, qué vector del conjunto $\overrightarrow{r} + \gamma NDU(s')$ ha actualizado previamente a $V_Q$ (independientemente de $\overrightarrow{r}$, ya que las operaciones de traslación y escala no afectan a las funciones sello). La función $sello_{U\rightarrow}$ de un vector sello $V_U \in U$ indica, para cada estado $s$ y cada acción $a$, qué vector del conjunto $Q(s, a)$ ha sido actualizado previamente por $V_U$. Es decir, si el vector $V_U$ ha actualizado previamente a $V_Q$, entonces $sello_{Q\leftarrow}(s') = sello_{U\rightarrow}(s, a)$. Si $sello_{Q\leftarrow}(s') = -1$, significa que $V_Q$ nunca ha sido actualizado por ningún vector de $\overrightarrow{r} + \gamma NDU(s')$. Si $sello_{U\rightarrow}(s, a) = -1$, significa que $V_U$ nunca ha actualizado a ningún vector de $Q(s, a)$. El algoritmo de actualización con sellos consta de tres fases a la hora de actualizar el conjunto $Q(s, a)$ con el conjunto $\overrightarrow{r} + \gamma NDU(s')$:

1. Actualizar los vectores de $Q(s, a)$ que ya han sido actualizados previamente por algún vector de $\overrightarrow{r} + \gamma(NDU(s'))$, con el vector correspondiente
2. Es posible que haya valores en $Q(s, a)$ que nunca hayan sido actualizados por ningun vector de $\overrightarrow{r} + \gamma NDU(s')$, en situaciones en las que una misma acción pueda llevar a estados distintos. En este caso, se realiza una actualización exhaustiva para estos vectores huérfanos de $Q(s, a)$ con todos los vectores de $\overrightarrow{r} + \gamma NDU(s')$, creando nuevos vectores en $Q(s, a)$
3. Inspeccionar $\overrightarrow{r} + \gamma NDU(s')$ en busca de vectores que no hayan copiado su valor en $Q(s, a)$, o cuyo valor asociado en $Q(s, a)$ se haya eliminado en algún episodio previo del algoritmo. En cualquiera de los dos casos, esta fase del algoritmo intenta añadir el vector de $\overrightarrow{r} + \gamma NDU(s')$ a $Q(s, a)$. En caso de que el motivo anterior fuera el primero, se genera un nuevo sello y se modifican las funciones sello correspondientes

El Cuadro 3 muestra detalladamente este proceso dividido en tres fases. Es necesario realizar algunas aclaraciones:

1. Cada sello generado tiene asociado un *factor de confianza* ($\theta_{sello}$). Esto permite calcular el *factor de confianza global*, $\Theta_V$, de un vector sello: $\Theta_V = min\{\theta_{sello} :$

$sello \in I_{sello_{V\leftarrow}} \setminus \{-1\}\}$. A la hora de comparar dos vectores, si alguno de ellos ha sido actualizado por más de un estado distinto, el operador $ND_\Theta(Q)$ no hará efectiva la comparación si el factor de confianza global de alguno de ellos es bajo ($\Theta_V < \Theta_{umbral}, 0 \leq \Theta_{umbral} \leq 1$). Así, aunque uno de los dos vectores sea dominado, se mantendrá en la frontera, ya que es posible que más adelante, cuando ese vector sea más fiable por estar más avanzado el proceso de aprendizaje, pase a estar en la frontera por derecho propio. Si por el contrario tiene un factor de confianza global alto ($\Theta_V > \Theta_{umbral}$), entonces puede ser eliminado.

2. El método *generarSello*() (líneas 18 y 29) devuelve el sello más pequeño que aún no ha sido usado
3. La modificación que la expresión $sello_{V_U\rightarrow}(s,a) = sello$ (líneas 20 y 31) realiza sobre $V_U$, se refleja también en el vector sello del conjunto $Q(s',a')$ correspondiente
4. El código completo para el algoritmo PQ-learning es el mismo que el del Cuadro 1, reemplazando la línea 7 por:
   $Q(s,a) \leftarrow actualizarSellos(Q(s,a), \overrightarrow{r} + \gamma NDU_\Theta(s'), s, a, s')$

*actualizarSellos*$(Q, U, s, a, s')$
1. Si $U = \varnothing$
2. Devolver $Q$
3. $Q_{s'} = \{V \in Q : sello_{V\leftarrow}(s') \neq -1\}$
4. $\overline{Q_{s'}} = \{V \in Q : sello_{V\leftarrow}(s') = -1\}$
5. $Q_{nuevo} = \varnothing$
6. $U_{ligados} = \varnothing$
7. Repetir (para cada $V_Q \in Q_{s'}$) // **Fase 1**
8. Si $\exists V_U \in U : sello_{V_U\rightarrow}(s,a) = sello_{V_Q\leftarrow}(s')$
9. $v_{nuevo} = (1-\alpha)\overrightarrow{v}_{V_Q} + \alpha\overrightarrow{v}_{V_U}$
10. $Q_{nuevo} = ND_\Theta(Q_{nuevo} \cup \{\langle v_{nuevo}, sello_{V_Q\leftarrow}, sello_{V_Q\rightarrow}\rangle\})$
11. $U_{ligados} = U_{ligados} \cup \{V_U\}$
12. Incrementar $\theta_{sello_{V_Q\leftarrow}(s')}$
13. Repetir (para cada $V_Q \in \overline{Q_{s'}}$) // **Fase 2**
14. Repetir (para cada $V_U \in U$)
15. $\overrightarrow{v}_{nuevo} = (1-\alpha)\overrightarrow{v}_{V_Q} + \alpha\overrightarrow{v}_{V_U}$
16. $V_{nuevo} = \langle \overrightarrow{v}_{nuevo}, sello_{V_Q\leftarrow}, sello_{V_Q\rightarrow}\rangle$
17. Si $sello_{V_U\rightarrow}(s,a) = -1$
18. $sello = generarSello()$
19. $\theta_{sello} = 0$
20. $sello_{V_U\rightarrow}(s,a) = sello$
21. $sello_{V_{nuevo}\leftarrow}(s') = sello$
22. En otro caso
23. $sello_{V_{nuevo}\leftarrow}(s') = sello_{V_U\rightarrow}(s,a)$
24. Incrementar $\theta_{sello_{V_U\rightarrow}(s,a)}$
25. $Q_{nuevo} = ND_\Theta(Q_{nuevo} \cup \{V_{nuevo}\})$
26. Repetir (para cada $V_U \in U$) // **Fase 3**
27. $V_{nuevo} = \langle \overrightarrow{v}_{V_U}, -1, -1\rangle$
28. Si $sello_{V_U\rightarrow}(s,a) = -1$
29. $sello = generarSello()$
30. $\theta_{sello} = 0$
31. $sello_{V_U\rightarrow}(s,a) = sello$
32. $sello_{V_{nuevo}\leftarrow}(s') = sello$
33. $Q_{nuevo} = ND_\Theta(Q_{nuevo} \cup \{V_{nuevo}\})$
34. En otro caso, si $V_U \notin U_{ligados}$ y $s'$ no es final
35. $sello_{V_{nuevo}\leftarrow}(s') = sello_{V_U\rightarrow}(s,a)$
36. $Q_{nuevo} = ND_\Theta(Q_{nuevo} \cup \{V_{nuevo}\})$
37. Devolver $Q_{nuevo}$

**Cuadro 3.** Algoritmo de actualización con sellos

Los vectores sello correspondientes a las políticas aprendidas partiendo del estado $s$ son los correspondientes al conjunto $NDU(s)$. Para explotar la información aprendida una vez que se ha realizado el aprendizaje, el agente ha de decidirse por un vector sello $V \in NDU(s)$, y tomar la acción asociada (es decir, la acción $a$ tal que $Q(s,a)$ dio lugar al valor $V$), que le llevará a un estado $s'$. Una vez en $s'$, ha de elegir la acción asociada al vector sello $W$ de $NDU(s')$ tal que $sello_{W\rightarrow}(s,a) = sello_{V\leftarrow}(s')$.

Ilustraremos el funcionamiento de PQ-learning mediante el PDM multiobjetivo de la Figura 1. Para cada vector $\overrightarrow{v}$ de los conjuntos Q, utilizaremos la notación $\overrightarrow{v}_{\overrightarrow{v}_a, \overrightarrow{v}_{b,\ldots,f_n}}$, donde $\overrightarrow{v}_a, \overrightarrow{v}_b, \ldots$ son los valores que han actualizado alguna vez a $\overrightarrow{v}$, y $f_n \in \{f_1, f_2, f_3\}$ es la fase de PQ-learning que ha dado lugar a ese valor (véase la tercera columna del Cuadro 2).

Las cinco primeras secuencias dan lugar a los mismos valores que la suma exhaustiva. En la sexta secuencia se perciben cambios significativos entre la suma exhaustiva y la que utiliza sellos: en $Q(s_1, a_{12})$ podemos ver que hay un valor extra que surge de copiar mediante la fase 3 el valor $(-1, 1)$. Esta operación dará lugar a un valor redundante, pero es necesaria para asegurar, en casos más complejos, que se tienen en cuenta todas las combinaciones de secuencias. Estos dos valores de $Q(s_1, a_{12})$ pasan a $Q(s_0, a_0)$ mediante la fase 3, en la siguiente ejecución de la misma secuencia (séptima fila). Por último, en la octava fila, vemos como el conjunto $Q(s_0, a_0)$ no crece de manera descontrolada, como sucede en la suma exhaustiva, gracias a los sellos. El valor $(-2, 1)$ dará lugar a un valor redundante que será eliminado por el operador $ND$, pues se irá aproximando al vector $(-2, 5, 1, 5)$ cuando el vector que lo actualiza, $(-1, 1) \in Q(s_1, a_{12})$, se actualice con el vector $(-2, 2) \in \overrightarrow{r} + NDU(s_2)$.

## 3. Resultados

En esta sección analizaremos los resultados arrojados por PQ-learning al aplicarlo a dos problemas de un banco de pruebas propuesto en la literatura de AR multiobjetivo [2]. Tal y como sugieren Vamplew et al [2], el rendimiento se ha medido en base al *hipervolumen* de la frontera aproximada, que es el volumen encerrado entre los puntos de la frontera y un punto de referencia que es dominado por todos los puntos de cualquier frontera aproximada. También mostramos el comportamiento del algoritmo que utiliza la suma exhaustiva de conjuntos sobre estos mismos problemas.

El primer problema considerado es *Deep Sea Treasure* (DST), de dos objetivos[1]. La frontera real para este problema es no convexa y tiene diez puntos y un hipervolumen de 535. La configuración paramétrica ha sido: $\alpha = 0{,}1, \gamma = 1$(ya que el problema es episódico), $\epsilon = 0{,}4, \Theta_{umbral} = 0{,}3$. El segundo problema considerado es *Resource Gathering* (RG), de tres objetivos[2]. Este problema presenta indeterminismo y su frontera real tiene ocho puntos y un hipervolumen de

[1] `http://uob-community.ballarat.edu.au/~pvamplew/MORL.html`

[2] Véase `http://uob-community.ballarat.edu.au/~pvamplew/MORL.html`. Aquí hemos reformulado RG en episodios

1,8299. La configuración paramétrica ha sido: $\alpha = 0{,}1, \gamma = 0{,}9$ (hemos mantenido la tasa de descuento inferior a 1 para dar más valor a las recompensas que son obtenidas antes, y así primar los caminos más cortos), $\epsilon = 0{,}4, \Theta_{umbral} = 0{,}3$. Para los parámetros $\alpha$, $\epsilon$ y $\Theta_{umbral}$ se han elegido unos valores prudentes que han funcionado satisfactoriamente. En la Figura 2 se muestra la evolución de la media del hipervolumen (tomada de diez ejecuciones del experimento) de la frontera aproximada durante 70000 episodios de aprendizaje ($\sim$ 25 min.) para DST y 3000 para RG ($\sim$ 2 min.).

El algoritmo Q-learning multiobjetivo con actualización exhaustiva también ha sido aplicado tanto a DST como a RG, con la misma parametrización que PQ-learning (salvo por el parámetro $\Theta_{umbral}$, que en la actualización exhaustiva no entra en juego). En DST, tras una hora únicamente había completado 9000 episodios de aprendizaje, momento en el cual el conjunto $NDU(s_i)$ tiene unos 50 vectores que no representan secuencias legítimas, tal y como veíamos en el ejemplo de la Sección 2.1. En RG, tras una hora únicamente había completado 500 episodios, momento en el cual el conjunto $NDU(s_i)$ tiene unos 70 vectores que no representan secuencias legítimas.

## 4. Conclusiones

Este artículo describe y analiza PQ-learning, una extensión de Q-learning para problemas de AR multiobjetivo. Como Q-learning, el algoritmo no asume un conocimiento perfecto de la dinámica del entorno. PQ-learning descubre un conjunto de políticas que pretende aproximar a la frontera real del problema. A diferencia de otros trabajos, el algoritmo no requiere información de preferencias, es aplicable a problemas con fronteras de Pareto no convexas y permite recuperar las secuencias de acciones correspondientes a distintas políticas Pareto-óptimas.

Hasta donde sabemos, PQ-learning es el primer algoritmo destinado a resolver procesos de decisión markovianos multiobjetivo sin ningún tipo de información del modelo ni de preferencias. Por esta razón no ha sido comparado con otros algoritmos de la literatura. El algoritmo ha sido aplicado a dos problemas

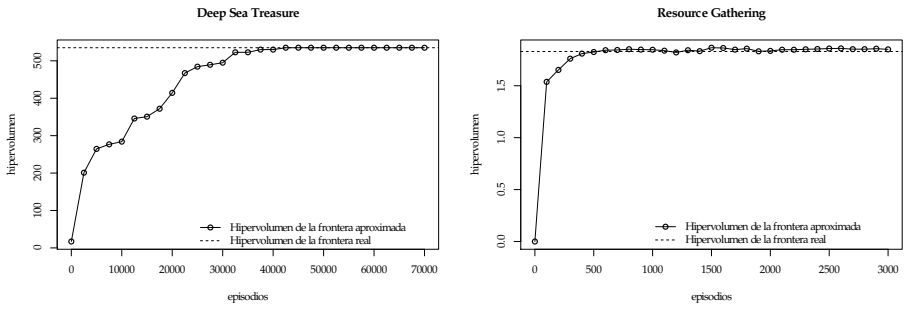


**Figura 2.** Resultados al aplicar PQ-learning sobre DST y RG

pertenecientes a un banco de pruebas propuesto en la literatura de AR multiobjetivo. PQ-learning ha sido comparado con una extensión de Q-learning que emplea técnicas habituales en programación dinámica multiobjetivo, resultando este último inviable para los problemas analizados.

La continuación natural del trabajo aquí expuesto es la realización de un estudio teórico completo del algoritmo, de la influencia de diferentes configuraciones de parámetros sobre el rendimiento y de escalabilidad, así como la aplicación del algoritmo a problemas reales. En concreto, los autores han aplicado el algoritmo Q-learning a problemas de diseño computacional en trabajos previos [10,11,12] que potencialmente podrán beneficiarse de PQ-learning.

**Agradecimientos.** Este trabajo está parcialmente financiado por el Plan Nacional de I+D+I, proyecto TIN2009-14179 (Gobierno de España, Ministerio de Ciencia e Innovación) y por la Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech. Manuela Ruiz-Montiel disfruta de una beca FPU (Gobierno de España, Ministerio de Educación).

## Referencias

1. Watkins, C.J.: Learning from delayed rewards. PhD thesis, University of Cambridge (1989)
2. Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., Dekker, E.: Empirical evaluation methods for multiobjective reinforcement learning algorithms. Mach. Learn. **84**(1-2) (2011) 51–80
3. Gábor, Z., Kalmár, Z., Szepesvári, C.: Multi-criteria reinforcement learning. ICML '98, Morgan Kaufmann Publishers Inc. (1998) 197–205
4. Natarajan, S., Tadepalli, P.: Dynamic preferences in multi-criteria reinforcement learning. ICML '05, ACM (2005) 601–608
5. White, D.J.: Multi-objective infinite-horizon discounted markov decision processes. Journal of Mathematical Analysis and Applications **89** (1982)
6. Barrett, L., Narayanan, S.: Learning all optimal policies with multiple criteria. ICML '08, ACM (2008) 41–47
7. Issabekov, R., Vamplew, P.: An empirical comparison of two common multiobjective reinforcement learning algorithms. In: AI 2012: Advances in Artificial Intelligence. Springer (2012) 626–636
8. Vamplew, P., Yearwood, J., Dazeley, R., Berry, A.: On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In: AI 2008: Advances in Artificial Intelligence. Springer (2008) 372–378
9. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction. MIT Press, Cambridge, Ma. (1998)
10. Ruiz-Montiel, M., Boned, J., Gavilanes, J., Jiménez, E., Mandow, L., Pérez-de-la Cruz, J.L.: Proyecto arquitectónico mediante gramáticas de formas sencillas y aprendizaje. In: Avances en Inteligencia Artificial: Actas de CAEPIA 11, 1. (2011)
11. Ruiz-Montiel, M., Mandow, L., Pérez-de-la Cruz, J.L., Gavilanes, J.: Shapes, grammars, constraints and policies. In: CEUR: SHAPES 1.0, 812. (2011)
12. Ruiz-Montiel, M., Boned, J., Gavilanes, J., Jiménez, E., Mandow, L., de-la Cruz, J.L.P.: Design with shape grammars and reinforcement learning. Advanced Engineering Informatics **27**(2) (2013) 230 – 245