Una Técnica de Aprendizaje por Refuerzo para la Resolución de Problemas Complejos

Tesistas
Flavia Paganelli Patricio A. Otamendi

Director
Dr. Andreas Matt

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Índice

Resumen

En problemas de aprendizaje por refuerzo (RL) con conjuntos grandes de acciones, se necesitan muchas iteraciones y bastante exploración para que el agente llegue a experimentar lo suficiente con todos los pares estado-acción. Frecuentemente sucede que no se exploran adecuadamente todas las acciones, y entonces los valores estimados de los pares estado-acción (valores-acción) pueden diferir mucho de los reales. Una política construída en base a estos valores probablemente sea subóptima. Presentamos una técnica para acelerar RL en este tipo de problemas, que denominamos propagación de valores-acción (PQ). Comúnmente en estos problemas, las acciones pueden ordenarse de tal forma que acciones vecinas o cercanas tienen efectos similares en el entorno y obtienen refuerzos similares. El método aprovecha la experiencia obtenida en cada acción elegida, propagándola a sus vecinas. En este trabajo describimos brevemente los métodos disponibles para atacar el problema de los conjuntos grandes de acciones o estados, basándonos en una minuciosa investigación de la literatura sobre el tema, y luego presentamos el nuevo método. Introducimos un simulador de RL orientado a objetos que nos permite implementar nuevas tareas de RL fácilmente usando el framework de clases. Finalmente ejecutamos experimentos en un problema gridworld para mostrar la efectividad del método propuesto.

1. Motivación

El aprendizaje por refuerzo (RL) es un método muy útil que se comporta bien en gran variedad de casos. Sin embargo, en problemas complejos donde el agente se enfrenta a una gran cantidad de acciones y estados, el desempeño puede ser malo. Por ejemplo, considérese el juego de Backgammon, que consiste de un enorme número de estados diferentes (aproximadamente 10^{20}), y acciones en cada estado. Es impracticable probar todas las acciones en todos los estados para aprender buenas estrategias de juego. Entonces el método de aprendizaje debe usar técnicas de generalización, que trabajan con un número relativamente pequeño de estados y acciones. De estos pocos estados y acciones visitados se extrae información para encontrar una estrategia para *todos* los estados y acciones. Tesauro [Tesauro, 1992] describe una forma de combinar RL con redes neuronales para jugar backgammon al nivel del mejor jugador humano.

Las aplicaciones de RL varían desde juegos a problemas de la vida real en industria y en robótica [Sutton & Barto, 1998, chapter 11]. Hay problemas donde el espacio de estados o acciones es continuo. Estos problemas por lo general se resuelven discretizando previamente el espacio continuo correspondiente.

Nuestro trabajo se enfoca en encontrar formas alternativas de mitigar el problema de los conjuntos grandes de acciones. La cuestión principal que enfrentamos es cómo extraer la mayor cantidad posible de información de cada observación durante la etapa de aprendizaje.

Para verificar los resultados necesitamos una plataforma de pruebas. En la literatura y en Internet no hemos podido encontrar un simulador fácil de extender para nuestros fines. Entonces decidimos desarrollar un framework que reflejara las entidades de aprendizaje por refuerzo (estados, acciones, entornos, etc.) y sus interacciones, y que también nos permitiera probar experimentalmente las técnicas de aprendizaje. De esta forma podemos comparar técnicas conocidas con las creadas por nosotros, en varias tareas de RL, y ofrecemos una plataforma para que otros investigadores evalúen sus propios métodos.

2. Aprendizaje por Refuerzo

En *Aprendizaje por Refuerzo* (RL) se enmarcan un conjunto de técnicas de aprendizaje donde el *agente* aprende ejecutando *acciones* por las cuales recibe castigos o recompensas, ver [Sutton, 1992], [Kaelbling *et al.* , 1996].

Estas recompensas o castigos son valores numéricos, positivos para las recompensas, negativos para los castigos. Estos valores se denominan en forma genérica *refuerzos*.

El agente interactúa con un *entorno*, en el cual *explora* probando distintas acciones en distintas situaciones para encontrar las mejores opciones. Así, observa y reacciona de acuerdo a lo visto. Una observación se lleva a cabo por medio de *sensores*, a través de los cuales el agente percibe su entorno. Por ejemplo, estos sensores pueden ser sensores de proximidad, un GPS (sistema de posicionamiento global), una cámara infrarroja o cualquier dispositivo de medición; los sensores que se usen dependerán del problema a resolver.

Comúnmente ocurre que los sensores son "incompletos"en el sentido que muestran sólo una parte del entorno. Por ejemplo, un robot con sensores de proximidad podrá reconocer la parte del entorno que está más próxima a él, pero si está en una habitación grande, no podrá ver más allá del rango de captación de sus sensores. En estos casos hablamos de un entorno *parcialmente observable* ([Cassandra *et al.* , 1994, p.1-2]), lo que también es denominado *estado escondido* por [Kaelbling *et al.* , 1996, section 6].

El resultado de la observación a través de los sensores se llama *estado*; es la situación del agente en el entorno. El estado de un entorno podría ser las coordenadas del agente, la posición de las figuras en un juego de mesa, los valores de un dispositivo sonar, etc.

Las técnicas de aprendizaje por refuerzo tratan de encontrar una estrategia para que el agente obtenga la mayor cantidad de recompensas positivas en el largo plazo. Lo que el agente trata de maximizar es lo que se llama el *retorno*, que es alguna función de la secuencia de refuerzos, por ejemplo:

$$(R_T)_t = r_{t+1} + r_{t+2} + ... + r_T,$$

donde T es el último paso –por ejemplo la última jugada de un juego de ta-te-ti–, y t el paso actual, con $0 \leq t \leq T$. El valor r_t es el refuerzo recibido en el paso t. Si no hay paso final definido, como por ejemplo en un robot que tiene que mantenerse en movimiento, esta suma puede ser infinita. Entonces para mantener la convergencia se usa el *retorno descontado* , definido como:

$$(R_\infty)_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

donde γ es el *factor de descuento*, con $0 \leq \gamma < 1$ para asegurar convergencia. Un valor cercano a cero da menos importancia a los refuerzos

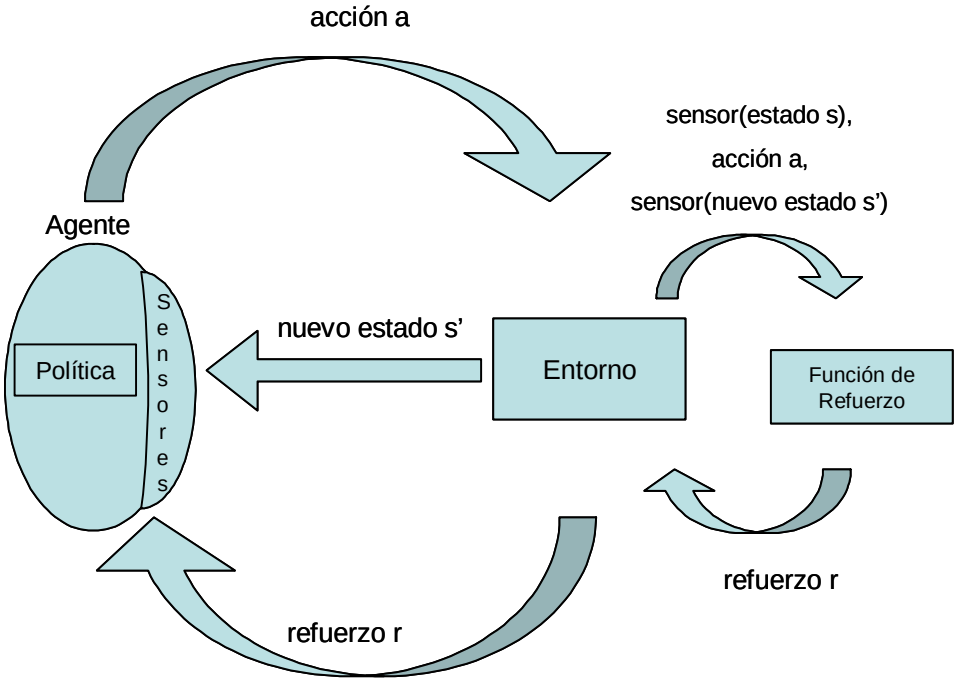


Figura 1: Interacción del agente con su entorno en Aprendizaje por Refuerzo.

futuros, mientras que un valor cercano a uno da igual importancia al refuerzo inmediato que a los refuerzos futuros.

La estrategia que el agente usa se llama *política*, y determina qué acciones va a tomar en cada estado, y con qué probabilidad. La política se denota π, donde $\pi(a, s)$ es la probabilidad de tomar una acción a desde un estado s. Por tanto,

$$\sum_{a} \pi(a, s) = 1,$$

para todas las acciones a que pueden elegirse en el estado s.

Las técnicas de RL tratan de encontrar políticas que maximicen el retorno o retorno descontado recibido por el agente al interactuar con el entorno.

La figura 1 muestra la interacción entre el agente y su entorno en aprendizaje por refuerzo.

2.1. Proceso de Markov

En síntesis, un sistema de aprendizaje por refuerzo consiste de un entorno, un agente y una función de refuerzo. El entorno contiene:

- S, un conjunto finito de estados s
- $A(s)_{s \in S}$, los conjuntos finitos de acciones disponibles en cada estado,
- $P(s' \mid a, s)$, la función de probabilidades de transición

Los refuerzos vienen dados por

- $R(s', a, s)$, la función de refuerzo.

La función de *probabilidades de transición* $P(s', a, s)$ es la probabilidad de ir a un estado s' dado que el agente se encuentra en el estado s y ejecuta la acción a. La función de refuerzo es específica para un determinado comportamiento, para un determinado problema, como por ejemplo el juego de ajedrez, ta-te-ti, un laberinto en particular, etc. La función P suele llamarse el *modelo del entorno*. Para algunos métodos de RL, como por ejemplo *Programación Dinámica* ([Howard, 1960]), el modelo debe ser conocido, para otros como Q-learning no hace falta un modelo dado en forma explícita.

La teoría de RL está basada en la *propiedad Markov*. Un estado tiene la propiedad Markov cuando contiene toda la información relevante de la historia de estados pasados. Por ejemplo, en el juego de ta-te-ti, es suficiente para el estado incluir qué posiciones están marcadas por qué jugador, pero el orden en que se realizaron las movidas es irrelevante. El jugador tiene toda la información del juego resumida en el estado actual: es un estado Markov.

Esto significa que los futuros estados dependen sólo del estado actual y no de los pasados. En términos de probabilidad de distribución la propiedad Markov puede ser escrita así:

$$\begin{aligned} \Pr\{s_{t+1} &= s' \mid s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0\} \\ &= \Pr\{s_{t+1} = s' \mid s_t, a_t\} \end{aligned}$$

En un entorno con la propiedad Markov uno puede predecir el próximo estado y refuerzo esperado sabiendo solamente el estado actual y la acción. Una buena descripción de esta propiedad se puede encontrar en [Sutton & Barto, 1998, sección 3.5].

Una tarea de RL que satisface la propiedad Markov se denomina *Proceso Markoviano.*

Hay muchos problemas de la vida real que son inherentemente no Markovianos. Comúnmente se tratan como si lo fueran, simplemente considerando los estados como markovianos, aunque en estos casos los resultados pueden estar lejos de lo óptimo.

En *Procesos Markovianos Parcialmente Obervables* (PMPO), el estado no es completamente observable, como se describió anteriormente. En realidad éste es el caso más común en problemas de la vida real. Por ejemplo, considérese la aplicación de PMPOs al diagnósito médico, donde el médico no conoce el estado interno completo del paciente y toma acciones como indicar un análisis de laboratorio, prescribir determinados medicamentos, hacer ciertas preguntas al paciente, etc. Para una recopilación de aplicaciones de PMPOs, ver [Cassandra, 1998].

2.2. Funciones de Valor

Casi todos los algoritmos de aprendizaje por refuerzo se basan en el cálculo de las *funciones de valor*, funciones que estiman la calidad de cada estado, o específicamente el retorno esperado al empezar en ese estado. La función de valor V para una política dada y un estado se calcula como sigue:

$$V^{\pi}(s) = E_{\pi}\{(R_{\infty})_0 \mid s_0 = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} \mid s_0 = s\},$$

donde E_{π} es el valor esperado dado que el agente sigue la política π y comienza en el paso 0.

También definimos la *función de valor* $Q^{\pi}(s, a)$ que es el retorno esperado cuando se empieza en el estado s y se toma la acción a, desde allí aplicando la política π:

$$Q^{\pi}(s, a) = E_{\pi}\{(R_{\infty})_0 \mid s_0 = s, a_0 = a\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} \mid s_0 = s, a_0 = a\}.$$

A los valores $Q^{\pi}(s, a)$ los llamaremos también *valores-acción.*

Una política π se dice mejor que otra política π' si hay al menos un estado donde la función de valor es mayor mientras que en el resto de los estados es igual, es decir

$V^{\pi}(s) \geq V^{\pi'}(s)$ para todos los estados s y hay al menos un estado s' con $V^{\pi}(s') > V^{\pi'}(s')$. Siempre hay al menos una política que es mejor o igual que todas las demás, y se llama *política óptima.*

V^* y Q^* denotan las funciones de valor para las políticas óptimas.

Para un tratamiento más matemático sobre optimalidad y la existencia de funciones de valor óptimas, consultar [Matt & Regensburger, 2004].

Para comparar políticas usamos la *utilidad de una política* $V(\pi)$, ver [Matt & Regensburger, 2004]. Es el promedio de los valores de todos los estados, definida como:

$$V(\pi) = \frac{1}{|S|} \sum_{s \in S} V^{\pi}(s),$$

donde $|S|$ es el número de estados del conjunto S. Se demuestra que las políticas óptimas comparten la misma utilidad y podemos considerar que aquéllas políticas con utilidad alta son mejores que las de utilidad más baja.

2.3. Q-learning

Q-learning [Watkins, 1989] es un algoritmo de aprendizaje por refuerzo que no necesita un modelo de entorno. Como otros métodos, estima la función de valor $Q^*(a, s)$. Una vez que se estimaron estos valores, la acción óptima desde cada estado es aquélla con valor máximo.

Luego de inicializarse los valores-acción en números arbitrarios, se actualizan en cada iteración de esta forma:

$$Q^{t+1}(a, s) = Q^t(a, s) + \alpha[r + \gamma \max_{a'} Q^t(a', s') - Q^t(a, s)]$$

donde a es la acción ejecutada, s' es el estado alcanzado y r el refuerzo observado. α es la *velocidad de aprendizaje* y $0 < \gamma < 1$ es el *factor de descuento*. Está demostrado que este algoritmo converge a $Q^*(a, s)$ con probabilidad uno, si cada par estado-acción es visitado repetidamente y la velocidad de aprendizaje se va decrementando adecuadamente en cada paso [Watkins & Dayan, 1992]. La velocidad de aprendizaje α, con $0 \leq \alpha < 1$, debería tomar sucesivos valores decrecientes $\alpha_1, \alpha_2, \alpha_3, ...$, cumpliendo estas restricciones:

$$\sum_{i=1}^{\infty} \alpha_i = \infty \text{ y } \sum_{i=1}^{\infty} \alpha_i^2 < \infty$$

El método que elegimos fue:

$$\alpha(s, a) = \frac{1}{n(s, a)} = 1, \frac{1}{2}, \frac{1}{3}, ...$$

donde $n(s, a)$ es el número de veces que el estado-acción (s, a) fue visitado.

Un problema que surge es el balance entre exploración y *explotación* (usar el conocimiento adquirido para elegir acciones con mayor valor). Una estrategia comúnmente usada es la de ser "goloso"la la mayoría de las veces, pero cada tanto, digamos con probabilidad ε, explorar tomando una acción al azar. Estos métodos se denominan *ε-golosos* [Sutton & Barto, 1998, sección 5.4].

Luego se construye una política golosa basada en estos valores-acción. La política así construida se considera una buena aproximación a la óptima.

3. Gridworld

Para los experimentos realizados en este trabajo usamos una implementación del entorno *gridworld*. Un entorno gridworld es una grilla donde el agente puede estar en cualquiera de sus casilleros, excepto cuando el casillero está ocupado por un *obstáculo*. A los casilleros vacíos los denominamos *visitables*. El agente está siempre apuntando a una determinada dirección.

Obsérvese por ejemplo la figura 4 (a), donde la grilla es de 5×5 y el agente mira en dirección 45^o. Las acciones que puede ejecutar consisten en moverse un paso en alguna de las 181 direcciones (todos los posibles ángulos enteros de izquierda a derecha).

Desde el punto de vista del agente, el problema puede formularse para que vea exactamente la posición y dirección en la que está –entorno totalmente observable–, o podemos considerar que tiene unos sensores de proximidad, que por lo general resultarán en un entorno parcialmente observable. Por ejemplo, en la figura 2 el agente tiene cuatro sensores de proximidad en cuatro direcciones que detectan hasta cuatro casilleros de distancia.

Algunos problemas que podrían plantearse en este entorno son búsqueda de camino mínimo, evitación de obstáculos, búsqueda de objetivo, solución de laberintos, seguimiento de pared. Usamos este último problema para evaluar nuestra propuesta.

3.1. Seguimiento de Pared

En seguimiento de pared, el objetivo del agente es moverse manteniéndose cerca de la pared de su derecha. Para eso, el agente como mínimo deberá poseer un sensor de distancia a su derecha.

Ver por ejemplo la figura 3 donde se ve un entorno gridworld y la trayectoria que se espera del agente luego de aprender a seguir la pared.

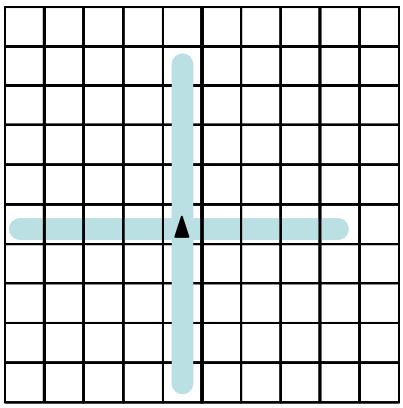


Figura 2: Agente con cuatro sensores

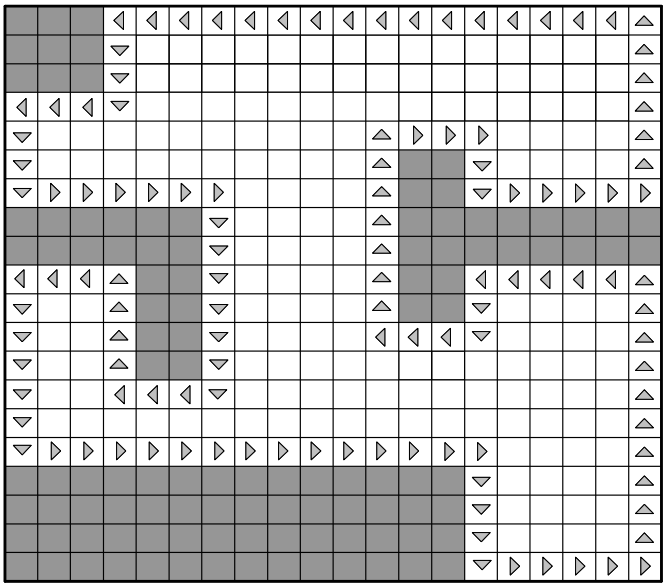
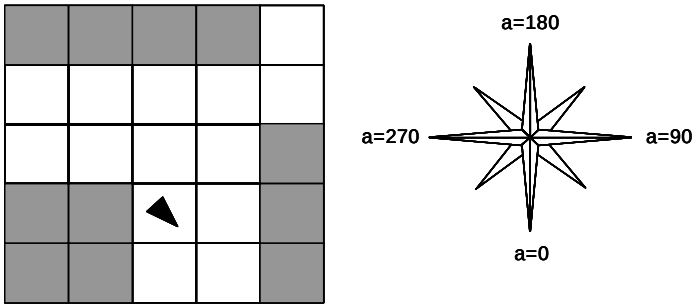


Figura 3: Seguimiento de la pared



a. El agente en la posición (2, 3, 45).

b.El agente puede apuntar a uno de los 360 ángulos *a*.

Figura 4:

3.2. Direcciones y Coordenadas

El agente puede apuntar a uno de 360 ángulos a, designados de acuerdo a la figura 4 (b).

La posición del agente está representada por sus coordenadas y una dirección. Sea $(x, y) \in \mathbb{N}^2$ las coordenadas donde el agente se encuentra, con $(0, 0)$ el casillero del rincón superior izquierdo. Sea a la dirección a la que apunta el agente, $0 \leq a < 360$. Entonces una posición está representada por la tripla (x, y, a).

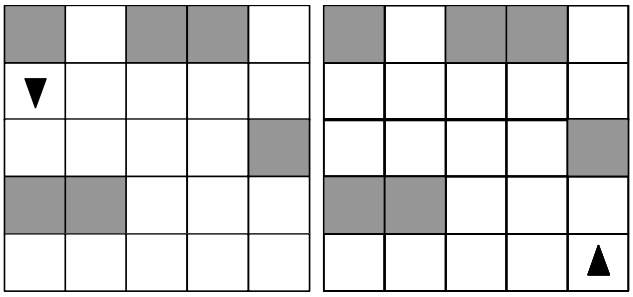
Por ejemplo, (2, 3, 45) representa la posición de la figura 4 (a).

3.3. Sensores

Para los experimentos ejecutados, usamos dos tipos de sensores. Los primeros, los *sensores de proximidad* –frontal, izquierdo, derecho y trasero– miden la distancia al objeto más cercano en un alcance dado. Usamos sensores con un alcance de cinco casilleros para cada sensor. Un obstáculo a más de cinco bloques de distancia del agente no es detectado. Los valores de los sensores, entonces, variarán entre cero y cinco, donde cinco significa que no hay ningún obstáculo a la vista, y cero significa que hay un obstáculo justo en el casillero contiguo. Codificamos los sensores de esta forma:

$$(s_I, s_F, s_D, s_T), \text{ con } 0 \leq s_I, s_F, s_D, s_T \leq 5$$

donde s_I, s_F, s_D y s_T son los sensores izquierdo, frontal, derecho y trasero respectivamente. Ver por ejemplo la figura 5 (a), donde la configuración de



(a) Sensores de proximidad con valor (4,1,0,0).

(b) Mismo valor de sensores.

Figura 5:

sensores es $(4, 1, 0, 0)$.

Posiciones diferentes del agente pueden resultar en la misma configuración de sensores. Por ejemplo, en la figura 5 (a) y (b) se ve al agente en dos posiciones distintas pero la configuración de sensores para ambas es la misma: $(4, 1, 0, 0)$. Con estos sensores, el entorno es parcialmente observable.

El segundo tipo de sensores son los *sensores absolutos*. Representan la posición exacta del agente dentro de la grilla, incluyendo la columna, la fila y el ángulo al que apunta. Se codifican de esta manera:

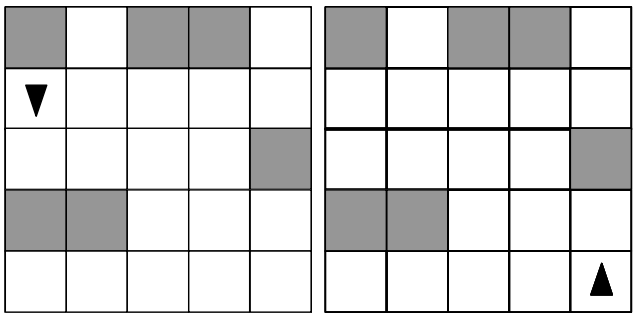
$$(c, f, \alpha), \text{ con } 0 \leq c < \text{cantidad de columnas}, 0 \leq f < \text{cantidad de filas, y } 0 \leq \alpha \leq 359$$

donde c es la columna, f es la fila y α es el ángulo. Ver por ejemplo las figuras 6 (a) y (b), donde las configuraciones de sensores son $(0, 1, 0)$ y $(4, 4, 180)$ respectivamente.

3.3.1. Acciones

Para cada estado s, se define un conjunto de acciones $A(s) = \{avanzar_i\}$, donde i (que indica el ángulo de rotación) está distrbuído uniformemente tal que $-90 \leq i \leq 90$. Por ejemplo, para $n = 3$,

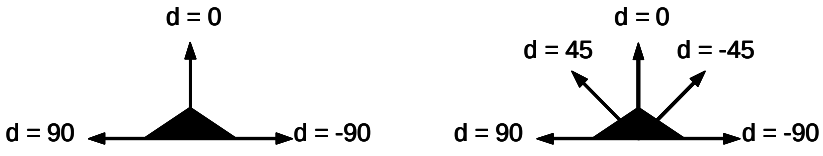
$A(s) = \{avanzar_{-90}, avanzar_0, avanzar_{90}\}$ (figura 7 (a)). En la figura 7 (b) se muestra cuáles serían las acciones si $n = 5$. En nuestro caso $A(s) = A(s')$ para todo s y $s\prime \in S$.



(a) Sensores absolutos con valor (0,1,0).

(b) Sensores absolutos con valor (4,4,180).

Figura 6:



(a) 3 acciones

(b) 5 acciones

Figura 7:

3.3.2. Refuerzos

La función de refuerzo depende del tipo de sensores que se use.

Para sensores de proximidad, definimos la función de refuerzo de la siguiente manera:

$$R_{wf}(s', a, s) = \begin{cases} -1, \text{ si } s_I = 0 \text{ y } 0 < a \leq 90 \text{ (obstáculo a la izquierda)} \\ -1, \text{ si } s_F = 0 \text{ y } (a < 90 \text{ ó } a > 270) \text{ (obstáculo enfrente)} \\ -1, \text{ si } s_D = 0 \text{ y } 270 \leq a < 360 \text{ (obstáculo a la derecha)} \\ +1, \text{ si } s'_D = 0 \\ -1, \text{ en cualquier otro caso.} \end{cases}$$

Si el agente choca contra un obstáculo, es castigado (los primeros tres casos). Si está cerca de una pared a su derecha, es recompensado. En cualquier otro caso es castigado.

Para sensores absolutos la función de refuerzo es la siguiente:

$$R_{wf}(s', a, s) = \begin{cases} -1, \text{ si } c = c' \text{ y } f = f' \\ +1, \text{ si } distanciaALaDerecha' = 0 \\ 0, \text{ si } distanciaALaDerecha' > 0 \text{ y} \\ \qquad distanciaALaDerecha' \leq distanciaALaDerecha \\ -1, \text{ en cualquier otro caso.} \end{cases}$$

donde $distanciaALaDerecha$ es la distancia del agente hasta el obstáculo más cercano hacia la derecha en el estado s, y $distanciaALaDerecha'$ es la misma distancia desde el estado s'.

Si el agente permanece en la misma posición, es castigado. Si está cerca de la pared de su derecha, es recompensado. Si está lejos de la pared pero se está acercando, el refuerzo es 0. En cualquier otro caso recibe un castigo.

4. Conjuntos Grandes de Acciones

En esta sección listamos primeramente algunas técnicas que atacan el problema de los espacios grandes de acciones o estados. Esta síntesis es producto de una búsqueda exhaustiva en la literatura actual de aprendizaje por refuerzo. Luego presentamos una nueva técnica, *propagación de valores-acción*, para acelerar el aprendizaje con conjuntos grandes de acciones.

4.1. Técnicas de la Literatura de RL

Las técnicas más específicamente enfocadas en el problema planteado son las de abstracción temporal. Éstas agrupan acciones para acelerar el

aprendizaje. Según el autor, se habla de opciones, *subgoals* o acciones abstractas (objetivos intermedios), macro-acciones o acciones extendidas temporalmente.

Sutton y Barto [Sutton *et al.* , 1999] generalizan y simplifican muchos trabajos anteriores para armar un sistema con abstracción temporal en RL y MDPs, respondiendo la pregunta "¿cuál es la mínima extensión de RL que permita incorporar abstracción temporal?". Otros trabajos sobre opciones y abstracción temporal incluyen [Precup & Sutton, 1997], [Perkins & Precup, 1999], [Precup, 2000], [McGovern *et al.* , 1997], [McGovern & Sutton, 1998] y [Hauskrecht *et al.* , 1998].

Otro de los objetivos planteados es el de descubrir automáticamente los *subgoals* en un problema dado. Por ejemplo, el objetivo *abrir la puerta* podría incluir los objetivos intermedios *llegar a la puerta*, *tomar el picaporte* y *girar el picaporte*. [Randlov, 1998], [McGovern & Barto, 2001] and [Stolle & Precup, 2002]. [Schoknecht & Riedmiller, 2002] y [Perkins & Precup, 1999] son otros acercamientos que tampoco requieren la definición previa de objetivos intermedios.

Otros trabajos sobre abstracción temporal incluyen [Provost *et al.* , 2004], [Goel & Huber, 2003] , [Iba, 1989] , [Korf, 1985b], [Korf, 1985a], [McGovern, 1998], [Kaelbling, 1993], [Sutton, 1995], [Thrun & Schwartz, 1995], [Parr & Russell, 1998], [Parr, 1998], [Singh, 1992], [Schoknecht, 2001b], [Schoknecht, 2001a], [Dietterich, 2000], [Schoknecht *et al.* , 1999].

Los métodos de generalización y aproximación de funciones por lo general se usan para problemas con espacios grandes de estados, pero también podrían aplicarse a los espacios de acciones. La generalización ataca el problema de generalizar la experiencia ganada en parte del espacio de estados para obtener una buena aproximación sobre un espacio más amplio. Con aproximación de funciones se toman muestras de la función buscada y se construye una aproximación de la función completa. De esta forma se reducen los requerimientos de memoria para resolver el problema. Algunas técnicas de aproximación de funciones incluyen las redes neuronales, métodos gradiente-descendientes lineales, lógica difusa y árboles de decisión.

Un aproximador de función se puede usar para representar las funciones de valor V o Q. En cada iteración de aprendizaje, la función aproximada es actualizada con un nuevo caso de test.

Una introducción a generalización y aproximación de funciones en RL se puede encontrar en [Sutton & Barto, 1998, chapter 8] y [Kaelbling *et al.* , 1996, section 6]. Otros trabajos sobre estos temas: [Sallans & Hinton, 2004] y [Sutton, 1996].

4.2. Propagación de valores-acción (PQ)

4.2.1. Motivación

En problemas de RL con conjuntos grandes de acciones y un tiempo finito de aprendizaje, puede suceder que el agente experimente poco o nada con algunas acciones. Entonces los valores-acción calculados pueden resultar inexactos, y una política construida en base a estos valores será -muy probablemente- subóptima. Para aprovechar mejor las acciones elegidas podemos actualizar no solamente los valores-acción de estas acciones, si no también las de sus acciones *vecinas*.

El concepto de vecindad o proximidad tiene que ver con una relación de similaridad entre las acciones. Consideraremos a cada acción como una entidad paramétrica $a(x_1, \ldots, x_n)$, donde cada x_i es un número que cuantifica el comportamiento de a. Por simplicidad, asumiremos que hay un único número representando a cada acción, de tal forma que $a(x_1, \ldots, x_n) = a_k$ para un k dado. De este modo, las acciones puede ordenarse como una lista $a_1, a_2, ...a_k, ...a_n$.

En esta lista, a_{k-1} y a_{k+1} son "similares"a a_k: son los vecinos de a_k (establecemos que cada acción tiene sólo dos vecinos, pero este modelo podría extenderse para que existan más de dos vecinos, por ejemplo en un arreglo de dos dimensiones se tendrían cuatro vecinos para cada acción). Los casos especiales a_0 y a_n se tratan de acuerdo al problema: podrían en algunos casos ser vecinos y en otros no.

Este método asume que el conjunto de acciones está ordenado en el sentido que dos acciones tienen un efecto similar en el entorno. Nótese que esto depende de la definición de la función de refuerzo: dos acciones vecinas, cuando se ejecutan en el mismo estado, nos llevan a estados similares –i.e. a estados con una distribución de probabilidad similar–, y bajo las mismas circunstancias reciben refuerzos similares.

4.2.2. El Método

Dadas $a, b \in A$, acciones vecinas. Entonces para todos los estados s', s se debe cumplir:

$$\begin{aligned} P(s' |\, a, s) &\approx P(s' |\, b, s) \\ &\text{y} \\ R(s', a, s) &\approx R(s', b, s) \end{aligned}$$

Como ejemplo, considérese un entorno gridworld con un conjunto de acciones que contiene las acciones $avanzar(x_i)$. Cada acción rota al agente sobre un ángulo x_i y luego lo mueve un paso hacia adelante. En este caso hay un parámetro, x_i, correspondiente al ángulo de rotación. Consideraremos las acciones donde $-90 \leq x_i \leq 90$, como se definieron en la sección 3.3.1. Estas acciones están ordenadas por ángulo. Por ejemplo, las dos acciones vecinas $avanzar(45)$ y $avanzar(46)$ tienen un efecto similar en el entorno.

El propósito de este método, que llamamos *Propagación de valores-acción* o PQ, es acelerar el cálculo de los valores-acción cuando se usa una técnica de aprendizaje basada en el cálculo de Q. Primero aplicamos el algoritmo común de aprendizaje para actualizar el valor-acción de la acción actual, y luego propagamos esta modificación a las acciones vecinas. Este proceso es aplicado en cada paso del aprendizaje.

El método propaga una porción del incremento a sus vecinos. El factor de propagación por el que se cuantifica el cambio es η, que satisface:

$$0 \leq \eta < 1 \qquad \eta \in \mathbb{R} \tag{1}$$

Este valor se decrementa en el tiempo hasta que llega a 0 en tiempo finito. Notar que cuando η es 0 este algoritmo se comporta como el algoritmo común de aprendizaje usado. Con esto se garantiza convergencia siempre que el algoritmo original converja.

Sea ahora b una acción vecina de a. Actualizamos $Q^t_{s,b}$, el valor-acción de b, que es una vecina inmediata de a, con la diferencia $Q^{t+1}_{s,a} - Q^t_{s,b}$, multiplicada por el valor η. Luego podemos repetir este proceso recursivamente para las vecinas de b hasta que no haya más.

Una expresión formal para la actualización viene dada por

$$Q^{t+1}_{s,b} = Q^t_{s,b} + (Q^{t+1}_{s,a} - Q^t_{s,a}) \cdot \eta \tag{2}$$

Donde $a, b \in A$, a y b son acciones vecinas. a es la acción que fue ejecutada, o está *más cerca* a la acción ejecutada que b. $\eta \in \mathbb{R}$ es el factor de propagación. Este valor se actualiza en cada paso del aprendizaje de la siguiente forma:

$$\eta \leftarrow \text{máx}(\eta - monto\_reducción, 0)$$

Donde $monto\_reducción$ es un valor constante que indica cuánto se reduce el valor η en cada paso.

4.2.3. Algoritmo

La propagación se realiza siguiendo el Algoritmo 4.1 y el procedimiento `Propagar` definido en el Algoritmo 4.2.

Algorithm 4.1 Propagación de valores-acción (PQ)

$\eta \leftarrow \eta_0$
for cada paso t **do**
  $Propagar(1, s_t, a_t, \eta)$
  $Propagar(-1, s_t, a_t, \eta)$
  $\eta \leftarrow \eta - monto\_reduccion$
end for

Algorithm 4.2 Procedimiento *Propagar* de PQ

Procedimiento *Propagar*(int dir, estado s, accion a, double η)
if a_{k+dir} es una acción válida **then**
  $b \leftarrow a_{k+dir}$
  $\Delta \leftarrow (Q^{t+1}_{s,a} - Q^{t}_{s,a}) \cdot \eta$
  if $|\Delta| > \epsilon$ **then**
    $Q^{t+1}_{s,b} \leftarrow Q^{t}_{s,b} + \Delta$
    Propagar(dir, s, b, η)
  end if
end if

En el procedimiento `Propagate`, `dir` es la dirección en la que los valores-acción se recorren, `s` es el estado actual visitado y `a` es la última acción cuyo valor-acción se actualizó. `b` es la acción cuyo valor-acción será actualizado.

4.2.4. Interpretación y Convergencia

Para comprender la idea del método, debemos enfatizar la importancia de su precondición, que establece la existencia de acciones vecinas similares. Dos acciones vecinas se dicen similares cuando, para un estado dado, el resultado de ejecutarlas lleva a estados similares, y la función de refuerzo devuelve un refuerzo similar.

Descubrimos que si las acciones son muy parecidas, PQ se comporta mejor con un factor de propagación cercano a uno. Por eso decimos que el factor de propagación representa también el grado de similitud entre acciones

vecinas. Cuanto más parecidas, más grande será el cambio transmitido a acciones vecinas.

A pesar de esto, el algoritmo va reduciendo el factor de propagación, no porque la similitud entre las acciones disminuya, sino porque de esta forma al llegar a cero el método se comporta precisamente como Q-learning. De este modo garantizamos dos cosas: primero, la convergencia del método; y segundo, la optimalidad del resultado.

Entonces PQ podría ser visto como un método para pre-ajustar los valores-acción para Q-learning.

Con esta interpretación del método, es fácil identificar los casos donde PQ podría no funcionar bien. Hay casos en que la precondición no se cumple. Si las acciones de un MDP no pueden ser ordenadas de forma tal que dos acciones contiguas sean similares, y el factor de propagación se fija en un valor cercano a uno, entonces preajustar los valores-acción no hará que Q-learning converja más rápido. Incluso podría hacerlo más lento.

Otro tema a mencionar es que usando PQ hay más actualizaciones en cada iteración. Se realizan más cálculos que en el algoritmo de Q-learning. Sin embargo, el costo de estos algoritmos no viene dado por la cantidad de cómputos, si no por la cantidad de lecturas de los sensores y ejecución de acciones, que son lo más costoso en aplicaciones reales. Por eso el factor más importante es la cantidad de iteraciones de aprendizaje. Y usando PQ se suelen necesitar menor cantidad de iteraciones para llegar a los mismos resultados que con Q-learning.

Los tipos de problemas más adecuados para PQ son generalmente aquellos donde las acciones son paramétricas por valores numéricos. De esa forma, las acciones con valores similares son candidatas a ser vecinas. A mayor cantidad de acciones, habrá mayor similitud entre ellas. Por ejemplo en el problema gridworld con cinco acciones no son muy similares, mientras que con 180 acciones, la diferencia de un grado entre ellas es poca.

5. Experimentos

En esta sección, mostramos ejecuciones de nuestra propuesta usando un simulador de aprendizaje por refuerzo. El problema elegido es seguimiento de pared en un gridworld (ver sección 3).

5.1. Comparación de Políticas

Ejecutamos los mismos ejemplos usando algoritmos existentes –como Programación Dinámica y Q-learning– y con PQ y luego comparamos los

resultados. Para eso hizo falta una manera de comparar políticas. Lo que usamos para medir la performance de una política fue un algoritmo Monte Carlo para calcular la utilidad de la misma.

5.1.1. Monte Carlo para calcular la Utilidad

Los métodos Monte Carlo funcionan tomando muestras de estados-acciones-refuerzos obtenidos de la interacción con el entorno, real o simulada. La idea es promediar los valores de retorno muestreados. Usamos el enfoque de [Sutton & Barto, 1998, chapter 5].

Nuestro evaluador de políticas Monte Carlo primero toma un conjunto de valores-acción y construye una política golosa π_G con ellos. Luego para cada estado ejecuta un número fijo de iteraciones con esta política π_G, y guarda los retornos descontados obtenidos al final de cada una. Esto se repite un cierto número de veces para obtener varias aproximaciones del retorno descontado del estado, y todos esos valores son promediados. Finalmente todos los retornos así obtenidos en todos los estados son promediados para otener la utilidad de la política π_G.

El algoritmo se muestra en el listado 5.1.

Algorithm 5.1 Cálculo de Utilidad con Monte Carlo

```
Entrada: valores-acción Q
Construir una política golosa π_G usando los valores-acción Q
for all estados s en S do
  for episodio = 1 a REPS do
    Ubicar al agente en el estado s
    Seguir la política π_G durante k iteraciones,
    calculando el retorno descontado R
    Añadir R a Retornos
  end for
end for
Devolver promedio(Retornos)
```

Ejecutamos algunos de los experimentos usando sensores de proximidad y otros con sensores absolutos (ver sección 3.3).

5.2. Los Experimentos

Comparamos las políticas aprendidas por PQ con aquéllas aprendidas usando Q-learning, en diferentes situaciones, siempre usando el problema

de seguimiento de la pared en un entorno gridworld. Vamos variando el entorno, cambiando el tamaño de la grilla, su forma, la observabilidad, la cantidad de acciones disponibles y el tipo de exploración en cada caso. Las políticas son comparadas calculando su utilidad (ver sección 2.2) en distintos momentos del proceso de aprendizaje. De esta forma se obtiene una curva de aprendizaje para cada algoritmo, y podemos compararlas para ver cuál converge más rápidamente.

5.2.1. Resultados

En general PQ mostró una velocidad de aprendizaje mayor que Q-learning en los casos esperados.

Sensores Absolutos Cuando se usa una cantidad reducida de acciones –por ejemplo 5 acciones–, PQ no se comporta mejor que Q-learning, e incluso puede ser levemente peor. Esto es lógico, porque las acciones vecinas no son muy similares, pero la actualización de PQ dará valores muy parecidos a acciones cercanas.

Se ejecutaron luego experimentos con 181 acciones variando el tipo de exploración: se usó exploración pura –elegir siempre una acción al azar–, o ε-golosa (ver sección 2.3). Los resultados se pueden ver en la figura 8 para distintos valores de ε. PQ tiene un mejor desempeño que Q-learning cuando la explotación es más intensa (ε más pequeño). Esto sucede porque cuando se elige una acción golosa en PQ, se actualiza no sólo el valor-acción de la misma, si no el de sus vecinas, descubriendo así más acciones óptimas o cercanas a óptimas en menos pasos. En problemas del mundo real muchas veces la explotación debe realizarse durante el aprendizaje, por lo cual PQ sería más adecuado.

Los siguientes experimentos se realizaron en una grilla con paredes oblicuas, donde es más útil tener más variedad de acciones. Por ejemplo, se usó la grilla de la figura 9. Con PQ se observan mejores resultados más rápido que con Q-learning: ver figura 10. Un ejemplo de una política obtenida en este ejemplo se puede ver en la figura 11.

Sensores de Proximidad Con sensores de proximidad los entornos resultantes son parcialmente observables, siendo menos observables cuanto más grandes son las grillas.

Por ejemplo, el entorno de la figura 9 se usó para obtener los resultados de la figura 12. Aunque con ambos algoritmos la política aprendida es peor que usando sensores absolutos por la observabilidad parcial –ver figura 13–,

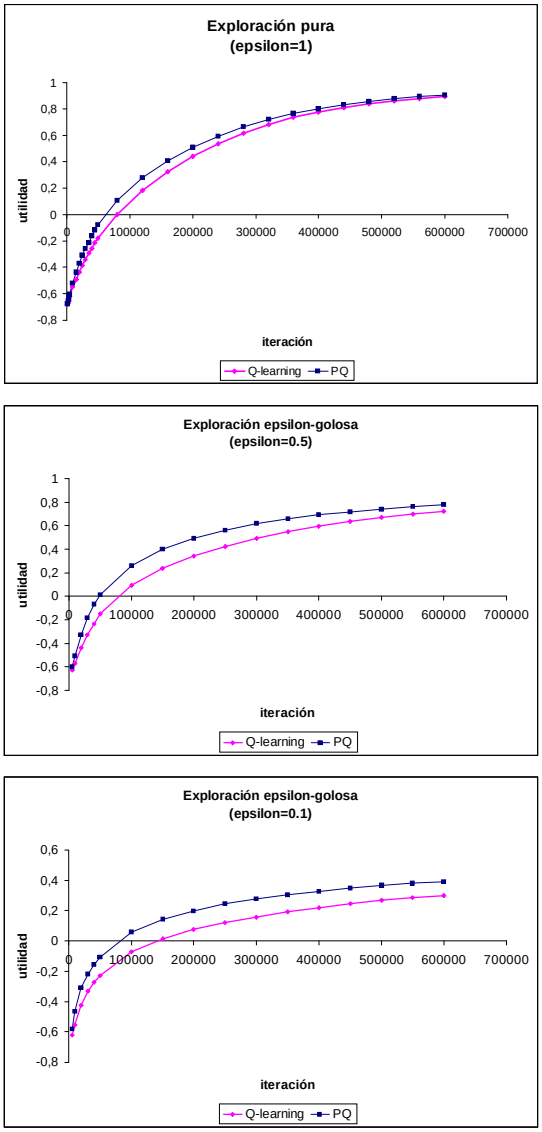


Figura 8: Exploración ε-golosa en una grilla de 10 x 10, usando diferentes valores para ε. Se usaron 181 acciones. PQ se comporta mejor cuando la explotación es más intensa.

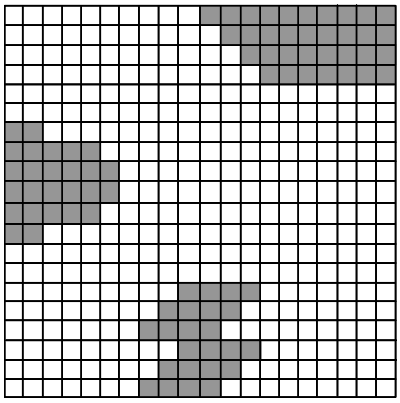


Figura 9: Una grilla de 20 x 20 con paredes oblicuas.

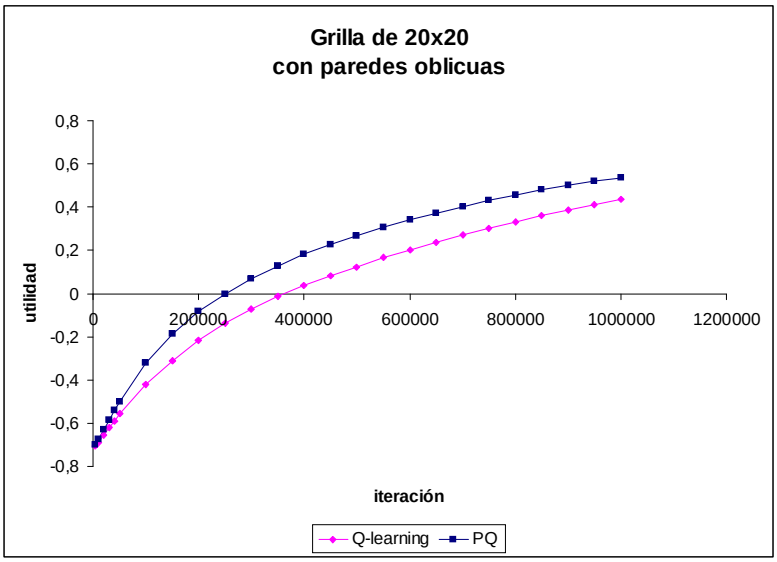


Figura 10: Utilidad en diferentes momentos del aprendizaje para Q-learning y PQ en una grilla de 20 x 20 con paredes oblicuas.

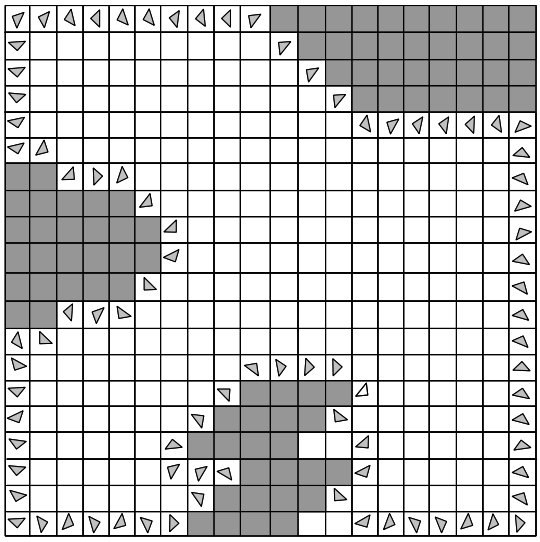


Figura 11: Un camino óptimo que podría seguirse usando una política aprendida por Q-learning o PQ. El agente de color blanco corresponde a la única situación en la cual recibe un castigo.

PQ tiene un desempeño superior, en menos pasos aprende una política mejor que Q-learning.

En todos los casos probados que cumplen con la precondición de PQ, el comportamiento de PQ es superior al encontrar mejores políticas antes que Q-learning.

6. Conclusiones y Trabajos Futuros

Implementamos PQ sólo en un ejemplo, pero hay muchas aplicaciones de RL donde se podría usar para acelerar el aprendizaje. En particular, aquellos problemas de la vida real en donde el espacio de acciones es continuo y éstas últimas se convierten a valores discretos –al igual que el problema gridworld planteado, donde tomamos únicamente los ángulos con valores enteros– son ideales para PQ.

PQ tiene una idea similar a las técnicas conocidas como del *vecino k-más cercano*, donde se toma la distancia euclidiana para medir las diferencias entre instancias vecinas, y los valores de las instancias desconocidas se estiman usando sus vecinas. Ver [Cover & Hart, 1967].

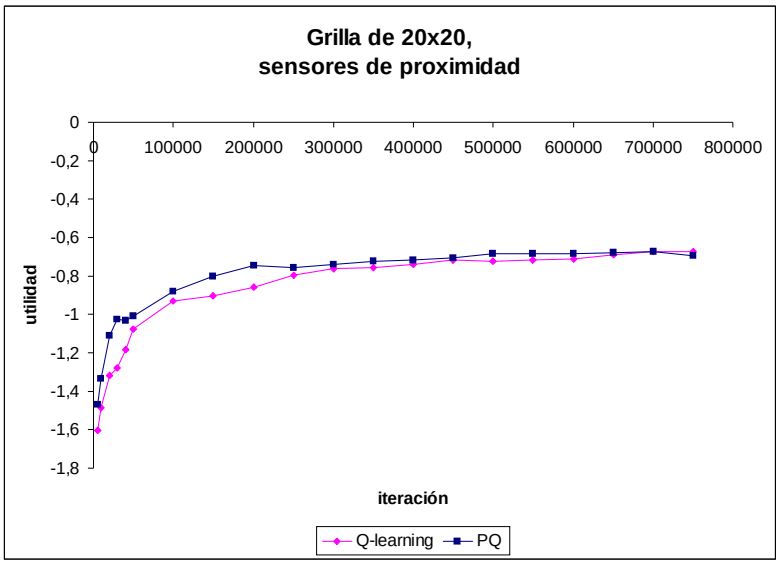


Figura 12: Utilidad en distintos momentos del aprendizaje de Q-learning y PQ en una grilla de 20 x 20 con paredes diagonales y usando sensores de proximidad.

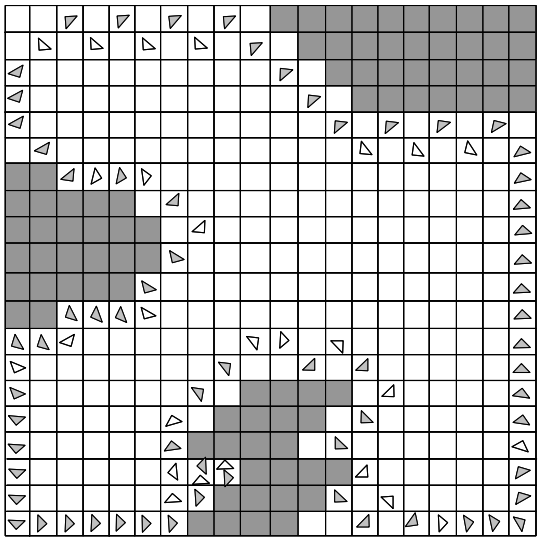


Figura 13: Un camino óptimo producto de una política aprendida por Q-learning o PQ, usando sensores de proximidad. El agente de color blanco indica los casos en que es castigado.

En robots autónomos, muchas veces empleados en entornos peligrosos o de difícil acceso –por ejemplo para neutralización de campos minados, remoción de residuos tóxicos, vehículos interplanetarios, inspección y reparación de cloacas– comúnmente las acciones pertenecen a un espacio continuo. Estas acciones son los actuadores del robot.

La robótica no es la única área donde PQ podría ser útil. Los problemas en los que las acciones pueden tomar valores numéricos variables son candidatos. Por ejemplo, en el trazado de políticas de salud, una decisión podría implicar una cantidad de dinero para ser invertido: en programas de vacunación, de educación para la salud, publicidades de prevención, etc. Las acciones en este caso consisten en la inversión de una cantidad de dinero x, donde x es el parámetro variable de la acción.

Como trabajo futuro, hay algunas cuestiones relativas a PQ que podrían explorarse en más detalle. Un tema a profundizar es cómo funcionaría la propagación cuando cada acción se define con más de un parámetro. Por ejemplo, en el problema gridworld podríamos usar acciones de la forma $avanzar(d, a)$, donde d es la distancia y α el ángulo de rotación. En esos casos no queda tan claro cómo se define el entorno de acciones vecinas.

Los experimentos ejecutados podrían reproducirse usando un robot para testear el algoritmo en el mundo físico real. El simulador de RL podría además extenderse desarrollando una interfaz gráfica que muestre al agente interactuando con su entorno.

PQ es un método basado en una idea muy simple que acelera el aprendizaje cuando se usa un algoritmo que calcula la función Q. Aunque este trabajo se enfocó en mejorar los casos donde el número de acciones es grande, se podría fácilmente adaptar PQ para mejorar también los casos de espacios grandes de estados. Ése es otro territorio en el que podríamos ahondar.

Referencias

[Cassandra, 1998] Cassandra, Anthony R. 1998. *A Survey of POMDP Applications*. Tech. rept.

[Cassandra *et al.* , 1994] Cassandra, Anthony R., Kaelbling, Leslie Pack, & Littman, Michael L. 1994. Acting Optimally in Partially Observable Stochastic Domains. *Pages 1023–1028 of: Proceedings of the Twelfth National Conference on Artificial Intelligence*, vol. 2. Seattle, Washington, USA: AAAI Press/MIT Press.

[Cover & Hart, 1967] Cover, T. M., & Hart, P. E. 1967. Nearest neighbor pattern classification. *IEEE, Transactions on Information Theory, IT-13*, **1**, 21–27.

[Dietterich, 2000] Dietterich, Thomas G. 2000. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, **13**(NOV), 227–303.

[Goel & Huber, 2003] Goel, Sandeep, & Huber, Manfred. 2003. Subgoal Discovery for Hierarchical Reinforcement Learning Using Learned Policies. *Pages 346–350 of: Florida Artificial Intelligence Research Society (FLAIRS) Conference.*

[Hauskrecht *et al.* , 1998] Hauskrecht, Milo, Meuleau, Nicolas, Boutilier, Craig, Kaelbling, Leslie Pack, & Dean, Tom. 1998. Hierarchical solution of Markov decision processes using macro-actions. *In: Proceedings of the 1998 Conference on Uncertainty in Artificial Intelligence, Madison, Wisconsin.*

[Howard, 1960] Howard, Ronald A. 1960. *Dynamic Programming and Markov Processes*. Cambridge, Massachusetts: Technology Press-Wiley.

[Iba, 1989] Iba, Glenn A. 1989. A Heuristic Approach to the Discovery of Macro-Operators. *Machine Learning*, **3**, 285–317.

[Kaelbling, 1993] Kaelbling, Leslie Pack. 1993. Hierarchical Learning in Stochastic Domains: Preliminary Results. *Pages 167–173 of: Proceedings 10th International Conference on Machine Learning (ICML)*. San Mateo, California: Morgan Kaufmann.

[Kaelbling *et al.* , 1996] Kaelbling, Leslie Pack, Littman, Michael L., & Moore, Andrew W. 1996. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, **4**. HTML version: http://www.cs.brown.edu/people/lpk/rl-survey/rl-survey.html.

[Korf, 1985a] Korf, Richard E. 1985a. *Learning to solve problems by searching for macro-operators*. Marshfield, MA, USA: Pitman Publishing, Inc.

[Korf, 1985b] Korf, Richard E. 1985b. Macro-operators: a weak method for learning. *Artificial Intelligence*, **26**(1), 35–77.

[Matt & Regensburger, 2004] Matt, Andreas, & Regensburger, Georg. 2004. *Reinforcement Learning for Several Environments. Theory and Applications*. Ph.D. thesis, University of Innsbruck.

[McGovern, 1998] McGovern, Amy. 1998. *acQuire-macros: An algorithm for automatically learning macro-actions*.

[McGovern & Barto, 2001] McGovern, Amy, & Barto, Andrew G. 2001. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. *Pages 361–368 of: Proceedings of the 18th International Conference on Machine Learning*.

[McGovern & Sutton, 1998] McGovern, Amy, & Sutton, Richard S. 1998. *Macro-Actions in Reinforcement Learning: An Empirical Analysis*. Ph.D. thesis, University of Massachusetts, Amherst Tech. rept.

[McGovern *et al.* , 1997] McGovern, Amy, Sutton, Richard S., & Fagg, Andrew H. 1997. Roles of Macro-Actions in Accelerating Reinforcement Learning. *In: 1997 Grace Hopper Celebration of Women in Computing*.

[Parr, 1998] Parr, Ronald. 1998. *Hierarchical Control and learning for Markov decision processes*.

[Parr & Russell, 1998] Parr, Ronald, & Russell, Stuart. 1998. Reinforcement Learning with Hierarchies of Machines. *Pages 1043–1049 of:* Jordan, Michael I., Kearns, Michael J., & Solla, Sara A. (eds), *Advances in Neural Information Processing Systems*, vol. 10. The MIT Press.

[Perkins & Precup, 1999] Perkins, Theodore J., & Precup, Doina. 1999. *Using Options for Knowledge Transfer in Reinforcement Learning.* Tech. rept. UM-CS-1999-034. Department of Computer Science, University of Massachusetts, Amherst.

[Precup, 2000] Precup, Doina. 2000. *Temporal abstraction in reinforcement learning.* Ph.D. thesis, University of Massachusetts Amherst.

[Precup & Sutton, 1997] Precup, Doina, & Sutton, Richard S. 1997. Multi-time Models for Temporally Abstract Planning. *In: NIPS.*

[Provost *et al.* , 2004] Provost, Jefferson, Kuipers, Benjamin J., & Miikkulainen, Risto. 2004. Self-Organizing Perceptual and Temporal Abstraction for Robot Reinforcement Learning. *In: Proceedings of the AAAI-04 Workshop on Learning and Planning in Markov Processes.*

[Randlov, 1998] Randlov, Jette. 1998. Learning Macro-Actions in Reinforcement Learning. *Pages 1045–1051 of: NIPS.*

[Sallans & Hinton, 2004] Sallans, Brian, & Hinton, Geoffrey E. 2004. Reinforcement Learning with Factored States and Actions. *Journal of Machine Learning Research*, **5**, 1063–1088.

[Schoknecht, 2001a] Schoknecht, Ralf. 2001a. *Hierarchical Reinforcement Learning with Multi-step Actions.*

[Schoknecht, 2001b] Schoknecht, Ralf. 2001b. *Using Multi-step Actions for Faster Reinforcement Learning.*

[Schoknecht & Riedmiller, 2002] Schoknecht, Ralf, & Riedmiller, Martin. 2002. Speeding-up Reinforcement Learning with Multi-step Actions. *Proceedings of the International Conference on Artificial Neural Networks*, **2415**.

[Schoknecht *et al.* , 1999] Schoknecht, Ralf, Spott, Martin, Liekweg, Florian, & Riedmiller, Martin. 1999 (Nov). Search Space Reduction for Strategy Learning in Sequential Decision Processes. *In: Proc. of Iconip 99.*

[Singh, 1992] Singh, Satinder P. 1992. Scaling Reinforcement Learning Algorithms by Learning Variable Temporal Resolution Models. *Pages 406–415 of: Proceedings of the Ninth Machine Learning Conference.* http://www.cs.colorado.edu/~baveja/papers.html.

[Stolle & Precup, 2002] Stolle, Martin, & Precup, Doina. 2002. Learning Options in Reinforcement Learning. *Pages 212–223 of: SARA.*

[Sutton, 1992] Sutton, Richard S. 1992. *Reinforcement Learning.* Boston, London: Kluwer Academic Publishers.

[Sutton, 1995] Sutton, Richard S. 1995. TD Models: Modeling the World at a Mixture of Time Scales. *Pages 531–539 of: 12th International Conference on Machine Learning.* Morgan Kaufmann.

[Sutton, 1996] Sutton, Richard S. 1996. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. *Pages 1038–1044 of:* Touretzky, David S., Mozer, Michael C., & Hasselmo, Michael E. (eds), *Advances in Neural Information Processing Systems*, vol. 8. The MIT Press.

[Sutton & Barto, 1998] Sutton, Richard S., & Barto, Andrew G. 1998. *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press. http://www-anw.cs.umass.edu/~rich/book/the-book.html.

[Sutton *et al.* , 1999] Sutton, Richard S., Precup, Doina, & Singh, Satinder. 1999. *Between MDPs and Semi-MDPS: A Framework for Temporal Abstraction in Reinforcement Learning.* Tech. rept. University of Massachusetts, Amherst, University of Colorado, Boulder.

[Tesauro, 1992] Tesauro, Gerald. 1992. Practical issues in temporal difference learning. *Machine Learning*, **8**(3–4), 257–277.

[Thrun & Schwartz, 1995] Thrun, Sebastian, & Schwartz, Anton. 1995. Finding Structure in Reinforcement Learning. *Pages 385–392 of:* Tesauro, G., Touretzky, D., & Leen, T. (eds), *Advances in Neural Information Processing Systems*, vol. 7. The MIT Press.

[Watkins, 1989] Watkins, Christopher. 1989. *Learning from Delayed Rewards.* Ph.D. thesis, Cambridge University.

[Watkins & Dayan, 1992] Watkins, Christopher, & Dayan, Peter. 1992. Technical note: Q-Learning. *Machine Learning*, **8**, 279–292.

Índice alfabético