Nuevo método de aceleración de los procesos de decisión de Markov

Ma.de Guadalupe García-Hernández*, José Ruiz-Pinales*, Sergio Ledesma-Orozco*,
J. Gabriel Aviña-Cervantes*, Edgar Alvarado-Méndez*

RESUMEN

En este artículo se presenta un nuevo método de aceleración para resolver a los procesos de decisión de Markov. El clásico algoritmo de iteración de valor ha resuelto satisfactoriamente a estos procesos estocásticos, pero este algoritmo y sus variantes aceleradas han sido lentos con factores de descuento cercanos a la unidad y sus propiedades de convergencia han dependido, en gran medida, de un buen ordenamiento en la actualización de estados. Recientemente se mostró que la iteración de valor presenta buena velocidad de convergencia gracias al uso de un algoritmo de ordenamiento topológico mejorado. Sin embargo, la desventaja de este algoritmo es debida a sus requerimientos de memoria. Aquí se presenta un método diferente para obtener un buen ordenamiento de estados actualizados con menor requerimiento de memoria. De igual manera se presentan los resultados experimentales obtenidos sobre un problema de ruta estocástica más corta.

ABSTRACT

In this paper we propose a new acceleration method for solving Markov decision processes. Value iteration is a classical algorithm for solving Markov decision processes, but this algorithm and its variants are quite slow for discount factors close to one and their convergence properties depend to a great extent on a good state update order. Recently, it has been shown that improved topological value iteration presents a good convergence speed thanks to the use of an improved topological ordering algorithm. Nevertheless, the drawback of this algorithm is due to its memory requirements. So, we present a different method to obtain a good state backup order with less memory requirements. Experimental results obtained on a stochastic shortest path problem are presented.

Recibido: 3 de marzo de 2011
Aceptado: 30 de junio de 2011

INTRODUCCIÓN

En planificación bajo incertidumbre el objetivo es encontrar una política que optimice alguna utilidad esperada. Muchos de los enfoques que encuentran tales políticas utilizan planificación basada en teoría de decisiones (Boutilier, 1999). A pesar de su aplicabilidad general y solidez matemática, la tarea de generar políticas (estrategias) óptimas para problemas considerablemente grandes es un reto computacional. Los procesos de decisión de Markov (MDP por sus siglas en inglés) (Bellman, 1957) (Puterman, 2005), han resuelto exitosamente problemas de decisión en control de procesos, análisis de decisiones y economía, entre otros. Sin embargo, la complejidad computacional de estos procesos es significativa para el caso de dominios de gran dimensión o continuos, tornando inabordable el tiempo de solución de problemas considerablemente grandes (Wingate, 2005). Este artículo explora la idea de minimizar el esfuerzo computacional necesario para calcular la política óptima (con su función de valor) de un MDP estacionario y discreto, variando un método iterativo tal como lo es el clásico algoritmo de iteración de valor. El objetivo es que actualice los estados en un orden óptimo, pues ciertas actualizaciones ejecutadas por iteración de valor pueden ser inútiles. En la literatura se ha discutido que las actualizaciones útiles son aquéllas que corresponden a estados en los que su función de valor presentó cambio en el barrido previo (Wingate, 2005). En este artículo se exploran diferentes métodos para calcular el óptimo ordenamiento de estados durante la solución de MDPs usando

Palabras clave:
Procesos de decisión de Markov; ordenamiento topológico; ruta más corta.

Keywords:
Markov decision processes; topological ordering; shortest path.

* División de Ingenierías, Campus Irapuato-Salamanca, Universidad de Guanajuato. Comunidad de Palo Blanco s/n, Salamanca, Guanajuato, México. Correos electrónicos: garciag@ugto.mx, pinales@ugto.mx, selo@ugto.mx, avina@ugto.mx, ealvarad@ugto.mx

al algoritmo de iteración de valor. Para manejar grandes MDPs, aquí se utiliza una representación compacta de dichos procesos, que se logra codificando las transiciones de estados con probabilidad diferente de cero (no nulas) como una lista, donde cada transición consiste de un estado inicial, su estado alcanzado, su probabilidad de transición y la acción aplicada. También se utilizan distintas combinaciones de técnicas de aceleración del estado del arte para resolver con mayor velocidad a los MDPs. Se emplean, entre otras, actualización asíncrona y priorización de estados. De esta última, la variante de iteración de valor utilizada es el algoritmo de barrido priorizado (McMahan, 2005), el cual ha sido eficiente al resolver MDPs determinísticos, pero ante MDPs estocásticos no ha garantizado la convergencia (Li, 2009) (Dai, 2007). Sin embargo, se visualiza que es posible usar este algoritmo para obtener buenos ordenamientos de estados para, enseguida, acelerar su convergencia al plan óptimo. Por lo que este artículo está organizado como sigue. Empieza con una breve introducción a los MDPs, seguido de la descripción del clásico algoritmo de iteración de valor para resolver estos procesos y sus variantes mejoradas. Después se presenta el algoritmo propuesto y finalmente se presentan los resultados experimentales, así como las conclusiones.

PROCESOS DE DECISIÓN DE MARKOV

Los procesos de decisión de Markov o MDPs proveen un marco matemático para modelar problemas de decisión secuencial en ambientes dinámicos inciertos (Puterman, 2005). Formalmente, un MDP es una tupla (S, A, P, R), donde S es un conjunto finito de estados $\{s_1,\ldots,s_n\}$, A es un conjunto finito de acciones $\{a_1,\ldots,a_n\}$, $P: S \times A \times S \rightarrow [0,1]$ es la función de probabilidad de transición, la cual asocia un conjunto de probables estados alcanzados a una acción dada en el estado en evaluación. $P(a, s, s')$ denota la probabilidad de transición de alcanzar al estado s', si se aplica la acción a en el estado s . $R(s, a)$ denota la recompensa obtenida si se aplica la acción a en el estado s . $\pi(s)$ denota una política (o estrategia) que produce una acción para cada estado, es una regla que especifica cuáles acciones podrían aplicarse en cada estado. La *propiedad Markoviana* garantiza que s' sólo depende del par (s, a). El principal problema de los MDPs es encontrar la política óptima que maximice la recompensa total esperada (Puterman, 2005). La función de valor, que es la recompensa esperada (o utilidad) cuando inicia en el estado s y sigue una política π, está dada por:

$$U^{\pi}(s) = E\left[\sum_{t} \gamma^{t} R(s_t, \pi(s_t)) \,\middle|\, s_0 = s\right], \qquad (1)$$

donde $\gamma \in [0,1]$ es un factor de descuento, que puede ser usado para recompensas futuras exponencialmente decrecientes. Para el caso de MDPs descontados ($0 < \gamma < 1$), la utilidad de una secuencia de estados infinita es siempre finita. De modo que el factor de descuento expresa que las futuras recompensas tienen menor valor que la recompensa del estado en evaluación (Russell, 2004). Para el caso de MDPs aditivos ($\gamma = 1$) y con horizonte infinito, la recompensa total esperada puede ser infinita y el agente debe garantizar su llegada a un estado meta. La función de valor $U(s)$ es usada para representar la recompensa esperada de la política dada. Así, la política óptima π^* es aquélla que maximiza su función de valor. Entonces, la función de valor óptima $U^*(s)$ está dada por (Puterman, 2005):

$$U^{*}(s) = \max_{\pi} U^{\pi}(s). \qquad (2)$$

Se sabe que la función de valor óptima $U^*_t(s)$ en la etapa t satisface la *ecuación de Bellman* (Bellman, 1957) (Puterman, 2005), la cual está dada por:

$$U_t(s) = \max_{a} \left\{ R(s,a) + \gamma \sum_{s'} P(a,s,s') U_{t+1}(s') \right\}. \qquad (3)$$

Los algoritmos de iteración de valor y de iteración de política (ambos pertenecen a programación dinámica) así como la programación lineal, son tres de las más conocidas técnicas para encontrar la función de valor óptima $U^*(s)$ y la política óptima π^* para problemas con horizonte infinito (Chang, 2007). Por último, para el caso de grandes MDPs, el uso de una representación compacta de su función de probabilidad de transición resulta en un ahorro de memoria considerable (Agrawal, 2002), la cual únicamente codifica a aquellas transiciones de estado que tienen probabilidad diferente de cero (no nula). De esta manera, es posible resolver problemas grandes que otros métodos no han podido resolver.

ACELERACIÓN DEL ALGORITMO DE ITERACIÓN DE VALOR

Como ya se mencionó, el clásico algoritmo de programación dinámica de iteración de valor ha resuelto satisfactoriamente a los MDPs. En cambio, el algoritmo de iteración de política, también de programación dinámica, y la programación lineal han tenido un alto costo computacional al tratar de resolver problemas con grandes espacios de estados. Esto es debido, principalmente, a que ambos requieren la solución (en cada iteración) de un sistema de ecuaciones lineales del mismo tamaño que el espacio de estados. En contraste, iteración de valor evita este problema usando

una aproximación recursiva de programación dinámica (Chang, 2007). Comenzando desde una función de valor inicial, el algoritmo de iteración de valor aplica actualizaciones sucesivas a la función de valor para cada estado $s \in S$ usando:

$$\hat{U}(s) = \max_a \left\{ R(s,a) + \gamma \sum_{s'} P(a,s,s')U(s') \right\}. \quad (4)$$

Sea $\{U_n \mid n = 0,1,\ldots\}$ la secuencia de funciones de valor obtenida por iteración de valor. Entonces se puede demostrar que cada función de valor satisface $|U_n - U^*| \leq \gamma^n |U_0 - U^*|$. Así, utilizando el clásico teorema del Punto Fijo de Banach (Kirk, 2001) se puede inferir que el algoritmo de iteración de valor converge a la función de valor óptima $U^*(s)$. La potencia de este algoritmo (en la solución de problemas de gran dimensión) viene del hecho de que las funciones de valor obtenidas pueden ser usadas como límites (cotas) para la función de valor óptima (Tijms, 2003). La complejidad computacional de una actualización en el algoritmo de iteración de valor es $O(|S|^2|A|)$. Sin embargo, el número de iteraciones requeridas puede ser considerablemente grande. Afortunadamente se ha demostrado en Littman (1995), que una cota superior para el número de iteraciones requeridas (n_{it}) por iteración de valor para alcanzar la solución óptima de acuerdo con un umbral de aproximación está dado por:

$$n_{it} \leq \frac{b + \log(\frac{1}{\varepsilon}) + \log(\frac{1}{1-\gamma}) + 1}{1-\gamma} \quad (5)$$

donde $0 < \gamma < 1$, b es el número de *bits* usados para codificar las recompensas y las probabilidades de transición de estados, y ε es el error de aproximación. Por otro lado, *el error de Bellman* (Puterman, 2005) está dado por:

$$B^t(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} P(a,s,s')U^t(s') \right\} - U^t(s). \quad (6)$$

Desafortunadamente la convergencia del algoritmo de iteración de valor puede ser demasiado lenta para γ con valores cercanos a la unidad. Por esta razón se han propuesto algunas mejoras a este algoritmo (Puterman, 2005). Por ejemplo, algunas técnicas de aceleración del estado del arte pueden mejorar la velocidad de convergencia, reducir el tiempo requerido por iteración y/o usar mejores criterios de paro. Una de las formas más fáciles de mejorar la velocidad de convergencia es actualizar las funciones de valor tan pronto como éstas quedan disponibles (también conocida como actualización asíncrona). Por ejemplo, iteración de valor con Gauss-Seidel utiliza la siguiente ecuación de actualización (Puterman, 2005):

$$U^t(s) = \max_a \left\{ R(s,a) + \gamma \sum_{s'<s} P(a,s,s')U^t(s') + \gamma \sum_{s' \geq s} P(a,s,s')U^{t-1}(s') \right\}. \quad (7)$$

Por otro lado, es bien conocido que iteración de política converge con menos iteraciones que las requeridas por iteración de valor, pero la primera requiere resolver un sistema de ecuaciones lineales en cada iteración. Es decir, iteración de valor es más lenta que iteración de política, pero no requiere resolver algún sistema de ecuaciones lineales. Por lo que un enfoque combinado de ellos, denominado iteración de política modificada (Puterman, 2005), puede aprovechar las ventajas de ambos utilizando un paso de evaluación de política parcial basado en iteración de valor. Otra variante de iteración de valor es la propuesta por Dibangoye y colaboradores (2008), a la que denominaron algoritmo de iteración de valor topológico modificado (iTVI), el cual utiliza un ordenamiento estático de estados. Este algoritmo, en lugar de minimizar el número de actualizaciones por iteración o de eliminar actualizaciones inútiles de estados, intenta minimizar el número de iteraciones calculando un buen ordenamiento estático de estados. Para ello, primero hace búsqueda del tipo "primero en profundidad" desde el estado inicial, con el objeto de conocer todos los estados alcanzables. Después hace búsqueda del tipo "primero en anchura" para construir una métrica, que está definida como la distancia desde el estado inicial hasta el estado en evaluación. Entonces construye un ordenamiento estático de estados actualizados con la métrica resultante, de modo que los estados que se encuentren más cercanos al estado inicial deberán ser actualizados primeramente. Este algoritmo garantiza la convergencia a la función de valor óptima debido a que actualiza de manera recursiva a todos los estados, en la misma forma en que la iteración de valor lo hace. Otro método que ha reducido considerablemente el tiempo tomado por iteración es el que proponen Wingate y colaboradores (2005), mediante la identificación y eliminación de acciones subóptimas. Por ejemplo, los límites de la función de valor óptima pueden ser usados para eliminar acciones subóptimas. La ventaja de este enfoque es que el conjunto de acciones es progresivamente reducido con la consecuente reducción de tiempo. También el número de iteraciones puede ser ligeramente reducido usando criterios de paro mejorados basados en aproximaciones más estrictas del error de Bellman (ecuación 6). Por ejemplo, un criterio de paro podría ser detener la iteración de valor cuando la medida del error de Bellman queda bajo cierto umbral preestablecido. Otra forma de mejorar la velocidad de convergencia, así como el tiempo requerido por iteración, es mediante priorización y partición (Wingate, 2005). Este enfoque está basado en la observación de que, en cada iteración, la función de valor (o utilidad) usualmente cambia sólo para un reducido conjunto de estados, de modo que si se restringe el cálculo solamente para dicho conjunto de estados, es esperada una reducción

del tiempo requerido por cada iteración. También indica que, para problemas acíclicos, el ordenamiento de aquellos estados que hacen que la matriz de transición se transforme en una matriz triangular puede producir un significativo ahorro de tiempo. Por último, otra variante acelerada de la iteración de valor es el Barrido Priorizado Mejorado (IPS, por sus siglas en inglés) (McMahan, 2005), que es un método basado en priorización, originalmente concebido como una extensión del clásico algoritmo de Dijkstra para la solución de MDPs de ruta estocástica más corta (con recompensas positivas y un solo estado meta). Este método se reduce al algoritmo de Dijkstra para el caso de MDPs determinísticos, es decir acíclicos. Además, IPS es un algoritmo de un solo paso y debido a esto es uno de los más rápidos algoritmos para resolver el caso determinístico de MDPs. Desafortunadamente, para el caso de MDPs altamente cíclicos (correspondientes al mundo real, como es el de ruta estocástica más corta) la convergencia de IPS no está garantizada (Dai, 2007) (Li, 2009).

ALGORITMO PROPUESTO

En este artículo se propone un nuevo método de aceleración del algoritmo de iteración de valor con el objeto de resolver MDPs de ruta estocástica más corta. Como ya se señaló, iTVI ha resultado considerablemente rápido en resolver estos procesos aplicando un ordenamiento topológico. Sin embargo, este ordenamiento podría no entregar el orden de actualización óptimo. Además, el cálculo del ordenamiento topológico podría requerir un costo de inicio considerablemente alto (*overhead*). Dado que IPS no ha garantizado la convergencia a la política óptima en problemas altamente cíclicos (Dai, 2007) (Li, 2009), en este trabajo se decidió utilizarlo para obtener información acerca de qué tan frecuente un estado debe ser actualizado. De modo que, partiendo de esa información, se calcula un ordenamiento mejorado de estados para resolver MDPs altamente cíclicos. En primera instancia se modificó al algoritmo de iteración de valor de modo que utilice una lista de transiciones no nulas (Agrawal, 2002) del tipo (s_k, s'_k, a_k, p_k), donde s'_k es el estado alcanzado después de ejecutar la acción a_k sobre el estado inicial s_k con una probabilidad de transición p_k. Sea el conjunto de transiciones no nulas $L = \{l_k | l_k = (s_k, s'_k, a_k, p_k) | p_k = T(s'_k | s_k, a_k) \neq 0\}$, el conjunto de recompensas de estado R, el número de estados n, el máximo error permitido ε y el factor de descuento γ. En lo sucesivo se supondrá que todas las transiciones en L quedan almacenadas de modo que se encuentren en orden ascendente de su estado inicial. También se le adicionó la técnica de aceleración de la convergencia denominada actualización asíncrona de Gauss-Seidel (ecuación 7). El algoritmo asíncrono resultante (aquí denominado SVI) se muestra en el Algoritmo 1.

función $(S, A, R, L, \gamma, \varepsilon)$

$(\forall s \in S) U^0(s) = \max_a R(s,a)$

repetir

  $(\forall s \in S, \forall a \in A) J(s,a) = R(s,a)$

  para $k = 1$ **a** $|L|$ **hacer**

    $s = l_k.s$

    $s' = l_k.s$

    $a = l_k.a$

    $p = l_k.p$

    $J(s,a) = J(s,a) + \gamma p U^{t-1}(s')$

  termina para

  $(\forall s \in S) U^t(s) = \max_a J(s,a)$

hasta que $|U^t - U^{t-1}| \leq \varepsilon$

$(\forall s \in S) \pi(s) = \operatorname{argmax}_a J(s,a)$

entrega π

Algoritmo 1. SVI (Algoritmo de iteración de valor modificado para usar una lista de transiciones no nulas con actualización asíncrona).

Con el objeto de obtener una mayor velocidad de convergencia del algoritmo de iteración de valor asíncrono, aquí se usaron diferentes combinaciones de técnicas de aceleración propuestas por Wingate y colaboradores (2005), tales como: priorización de estados con máxima recompensa, ordenamiento estático de estados y ordenamiento topológico de estados. Se llamó ASVI a la primera variante de SVI, que incluye priorización de estados. Se denominó ASVISR a la segunda variante de SVI (mostrada en el Algoritmo 2) que solamente actualiza a aquellos estados (así como a sus vecinos más próximos) cuya función de valor cambió en la iteración previa, utilizando un reordenamiento estático de estados en forma decreciente de máxima recompensa. Esto es debido a que resulta mejor usar un buen ordenamiento estático en lugar de un ordenamiento aleatorio, en cuanto a los recursos computacionales utilizados. De esta manera, el ordenamiento de estados es ejecutado una sola vez (durante el inicio), haciendo de esta manera eficaz al proceso. Cabe señalar que el ordenamiento variable es solamente efectivo cuando se utiliza actualización asíncrona (Wingate, 2005). Por último, se llamó ASVITO a la tercera variante de SVI, que usa los mismos procedimientos de aceleración que ASVISR pero que, además, utiliza el

ordenamiento topológico modificado. Cabe mencionar que, para casos especiales de MDPs acíclicos, el uso de un ordenamiento topológico sobre los estados ha producido un óptimo ordenamiento de estados. Para otros casos, una buena posibilidad sería reordenar los estados de modo que se obtenga la matriz de transición "casi triangular" (Wingate, 2005). Otra opción sería calcular el ordenamiento topológico de los estados debido a que, para MDPs acíclicos, al menos un ordenamiento topológico existe y pueden ser aplicados algoritmos clásicos de ordenamiento topológico (sort), cuyos tiempos de ejecución presentan comportamiento lineal en el número de nodos y en el número de aristas del grafo. Desafortunadamente, los problemas del mundo real involucran MDPs altamente cíclicos, en los que un ordenamiento topológico es imposible. Sin embargo, en estos casos Wingate y colaboradores (2005), utilizaron con cierto éxito un ordenamiento topológico modificado.

función $(S, A, R, L, \gamma, \varepsilon)$

sort(R, L)

$(\forall s \in S)U(s) = \max_a R(s,a)$

$(\forall s \in S)B(s) = U(s)$

$cambio = \{s \mid |B(s)| > \varepsilon\}$

repetir

  para todo $s \in cambio$ **hacer**

  $(\forall a \in A)J(s,a) = \sum_{k|l_k.s=s,l_k.a=a} l_k.pU(l_k.s')$

  $B(s) = \max_a\{R(s,a) + \gamma J(s,a)\} - U(s)$

  $U(s) = B(s) + U(s)$

  termina para todo

  para todo $s \in vecinos(cambio) - cambio$

  $J(s,a) = \sum_{k|l_k.s=s,l_k.a=a} l_k.pU(l_k.s')$

  $B(s) = \max_a\{R(s,a) + \gamma J(s,a)\} - U(s)$

  $U(s) = B(s) + U(s)$

  termina para todo

  $cambio = \{s \mid s \in cambio \cup vecinos(s), |B(s)| > \varepsilon\}$

hasta que $\max_{s \in cambio} |B(s)| \leq \varepsilon$

$(\forall s \in S)\pi(s) = \operatorname{argmax}_a\{R(s,a) + \gamma J(s,a)\}$

entrega π

Algoritmo 2. ASVISR (actualización asíncrona de estados que cambiaron entre iteraciones, con ordenamiento estático de estados y criterio de paro mejorado).

El método aquí propuesto consiste en ASVISR con un nuevo método de ordenamiento de estados, al cual denominamos ASVISR mejorado y se muestra en el Algoritmo 3.

función $(S, A, R, L, \gamma, \varepsilon)$

$(\forall s \in S)U(s) = \max_a R(s,a)$

$(\forall s \in S)B(s) = U(s)$

$cambio = \{s \mid |B(s)| > \varepsilon\}$

calcula el número de actualizaciones para cada estado mediante IPS.

calcula el nuevo ordenamiento estático de actualizaciones de estado.

repetir

  para todo $s \in cambio$ **hacer**

  $(\forall a \in A)J(s,a) = \sum_{k|l_k.s=s,l_k.a=a} l_k.pU(l_k.s')$

  $B(s) = \max_a\{R(s,a) + \gamma J(s,a)\} - U(s)$

  $U(s) = B(s) + U(s)$

  termina para todo

  para todo $s \in vecinos(cambio) - cambio$

  $J(s,a) = \sum_{k|l_k.s=s,l_k.a=a} l_k.pU(l_k.s')$

  $B(s) = \max_a\{R(s,a) + \gamma J(s,a)\} - U(s)$

  $U(s) = B(s) + U(s)$

  termina para todo

  $cambio = \{s \mid s \in cambio \cup vecinos(s), |B(s)| > \varepsilon\}$

hasta que $\max_{s \in cambio} |B(s)| \leq \varepsilon$

$(\forall s \in S)\pi(s) = \operatorname{argmax}_a\{R(s,a) + \gamma J(s,a)\}$

entrega π

Algoritmo 3. ASVISR mejorado.

El algoritmo ASVISR mejorado ejecuta un ordenamiento de actualizaciones de estados y consiste en lo siguiente: cuando un estado es actualizado, inmediatamente es insertado su correspondiente número de actualización en una lista, la cual es priorizada mediante el algoritmo IPS en orden decreciente de máximo número de veces que se ha actualizado dicho estado. Hace actualización asíncrona de la función de valor sólo en aquellos estados (así como sus vecinos más cercanos) cuyo número de actualización cambió en la iteración previa. De esta manera el ordenamiento de estados es ejecutado una sola vez (durante el inicio del programa). Por último, se comparó

el desempeño del algoritmo aquí propuesto con dos eficientes algoritmos pertenecientes al estado del arte: iTVI y VDP (enfoque de programación dinámica que se aplica en el dominio utilizado), en una tarea compleja de estrategias de navegación, la cual se describe en el siguiente segmento.

EXPERIMENTOS

Para evaluar el desempeño del algoritmo propuesto, se seleccionó al dominio *Sailing Strategics* (Vanderbei, 1996) de estrategias de navegación en un bote de vela. Este dominio corresponde a un problema de ruta estocástica más corta, de espacio finito de estados y de acciones, donde un bote de vela debe encontrar la ruta más corta entre dos puntos en un lago bajo condiciones fluctuantes de viento. Los detalles del problema son los siguientes: la posición del bote de vela está representada por un par de coordenadas sobre una malla de medidas finitas. El controlador tiene ocho acciones que lo pueden dirigir hacia una posición vecina en el lago. Cada acción tiene un costo temporal dependiendo de la dirección de la proa del bote dada por la acción y de la dirección del viento. Para la acción cuya dirección es justamente opuesta a la dirección del viento, el costo debe ser considerablemente alto. Pero si el viento está a 45° medidos desde la proa del bote, se dice que el bote se encuentra ante una tarea *contra-el-viento*. De este modo, el dominio supone que el bote se tomará 4 s para navegar hasta uno de los puntos de su entorno inmediato. En cambio, si el viento se encuentra a 90° de la proa del bote, este se moverá más rápido y podrá alcanzar al siguiente punto de su entorno inmediato en solamente 3 s. En este caso la tarea se denomina *con-viento-de-costado*. Por otro lado, si se encuentra con el viento a escuadra en popa, entonces se dice que está en una tarea *a-favor-del-viento*, por lo que se tomará sólo 2 s en su traslado a un punto vecino. Finalmente, si el bote está navegando directamente a favor del viento, entonces se dice que está en una tarea *con-el-viento* y sólo requerirá de 1 s para su traslado a algún punto vecino. Por otro lado, si el viento está golpeando el lado izquierdo del bote, entonces se encuentra en una tarea *a-babor*. Si el viento lo golpea por el lado derecho, entonces tendrá una tarea *a-estribor*. Si el bote está navegando en la misma dirección que el viento, entonces no tendrá tarea *a-babor* ni *a-estribor*. Para que el bote cambie la tarea de *a-babor* a *a-estribor* (o viceversa), el dominio supone que el marinero desperdiciará 3 s para lograrlo. Cabe señalar que para mantener el modelo simple, se asume que la intensidad del viento es constante, pero que su dirección puede cambiar en cualquier momento. También que el viento puede venir desde una de las tres direcciones siguientes: la misma dirección del viento habido en el anterior punto donde se encontraba el bote, a 45° a la derecha o a la izquierda del viento habido en el punto anterior de ubicación del bote. La tabla 1 muestra las probabilidades de cambio en la dirección del viento.

Tabla 1.
Probabilidades de cambio de dirección del viento. La primera columna indica la dirección del viento en el punto anterior y el primer renglón indica la nueva dirección del viento en el punto en evaluación.

	N	NE	E	SE	S	SW	W	NW
N	0.4	0.3						0.3
NE	0.4	0.3	0.3					
E		0.4	0.3	0.3				
SE			0.4	0.3	0.3			
S				0.4	0.2	0.4		
SW					0.3	0.3	0.4	
W						0.3	0.3	0.4
NW	0.4						0.3	0.3

Cada estado en evaluación s comprende una posición del bote (x, y), una tarea $t \in \{0, 1, 2\}$ y una dirección actualizada del viento $w \in \{0, 1, ..., 7\}$. Por otro lado, cuando la proa del bote se encuentra a lo largo de una de las direcciones diagonales entre los cuatro puntos cardinales, entonces el tiempo es multiplicado por $\sqrt{2}$ requiriendo de esta manera, recorrer una distancia poco más larga en su travesía que si se moviera solamente a alguno de los cuatro puntos cardinales. Todos los experimentos fueron realizados en una computadora Pentium D con 2.66 GHz y 2 GB RAM. Los algoritmos fueron implementados en lenguaje *Java* bajo un ambiente de planificación de movimientos robóticos (Reyes, 2006). Cabe señalar que se utilizó *Java* por ser una herramienta independiente de la plataforma en que se encuentra, que ofrece máxima portabilidad y seguridad y que tiene mínimas dependencias de la implementación. Por otro lado, para todos los experimentos se establecieron los siguientes valores: $\varepsilon = 10^{-7}$ y $\gamma = 1$. El último valor es debido a que el problema de estrategias de navegación trata con un MDP no descontado, donde la convergencia no está garantizada por el clásico teorema del Punto Fijo de Banach (Kirk, 2001) y el límite para el número de iteraciones dado por la ecuación 5 ya no es válido. Afortunadamente la presencia de estados absorbentes (estados con nula recompensa y 100 % de probabilidad de permanecer o quedar atracado en el mismo estado, como son las orillas del lago) permiten la convergencia del algoritmo (Hinderer, 2003). Por último, se decidió que la medida del lago tuviera una variación en los experimentos desde 50 × 50 hasta 200 × 200 segmentos por lado, resultando con esta variación que el número de estados fuera desde 55 296 hasta 940 896,

respectivamente. Se repitió diez veces cada corrida y se calculó la media aritmética, así como la desviación estándar del tiempo de solución.

RESULTADOS

La figura 1 muestra el tiempo de solución (en segundos) como una función del número de estados para todos los algoritmos probados en un ambiente de planificación de movimientos robóticos (Reyes, 2006). Ahí se puede observar que el algoritmo propuesto (ASVISR mejorado) fue significativamente más rápido que los otros algoritmos probados. En la tabla 2 se tiene que, para 940 896 estados, el algoritmo propuesto fue 2.6 veces más rápido que ASVISR, 2.9 veces más rápido que ASVI, 4.5 veces más rápido que SVI y 8.1 veces más rápido que VDP. Cabe señalar que iTVI requirió de una mayor cantidad de memoria que los otros algoritmos probados y debido a esto agotó el recurso de memoria computacional aproximadamente a los 400 000 estados.

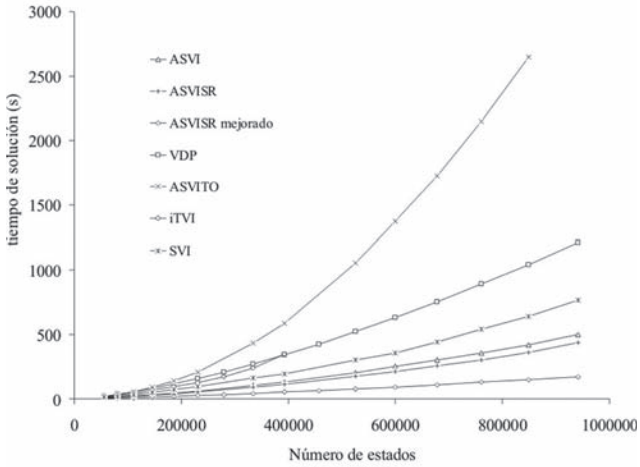

Figura 1. Tiempo de solución en función del número de estados de los algoritmos probados.

Tabla 2.
Resumen de resultados en términos de tiempo de solución (en milisegundos) para los algoritmos probados (para 940 896 estados). El tiempo de solución relativo fue calculado con respecto al tiempo de solución del algoritmo propuesto.

Algoritmo	Tiempo de solución (ms)	Tiempo de solución relativo
ASVITO	3 455 656	20.4
VDP	1 376 572	8.1
SVI	766 125	4.5
ASVI	499 172	2.9
ASVISR	436 303	2.6
ASVISR mejorado	169 485	1.0

En cuanto a los algoritmos que emplean diferente ordenamiento de estados, en la misma tabla se observa que el nuevo método de ordenamiento (en forma decreciente de máximo número de actualizaciones de estado) resulta 2.6 veces más rápido que el ordenamiento de estados en forma decreciente de máxima recompensa y 20.4 veces más rápido que el ordenamiento topológico modificado, los dos últimos métodos pertenecen a Wingate y colaboradores (2005). En este caso, el uso del ordenamiento topológico modificado no refleja una mejor solución debido a su alto costo de inicio. Una alternativa a este algoritmo es remover al más pequeño conjunto de transiciones cíclicas, de modo que el MDP se haga acíclico (esto es conocido como el problema del conjunto de arcos de retroalimentación) y entonces utilizar algoritmos de complejidad lineal para grafos acíclicos basados en búsqueda del tipo "primero en profundidad". Desafortunadamente, es sabido que el problema del conjunto de arcos de retroalimentación es del tipo NP-completo (Garey, 1990). Otra posibilidad es encontrar un ordenamiento topológico que pueda ser aplicado a un algoritmo con componentes fuertemente conectados, como lo presentan Dai y colaboradores (2007).

CONCLUSIONES

En este artículo se propuso un nuevo método de aceleración sobre el algoritmo de iteración de valor (clásico algoritmo que ha resuelto con cierta eficiencia a los MDPs) que ordena a los estados por su número de actualizaciónes, priorizando a aquellos estados que más actualizaciones han requerido durante la solución del MDP. Se comparó experimentalmente con distintas combinaciones de procedimientos de aceleración pertenecientes al estado del arte, tales como actualización asíncrona (Puterman, 2005), priorización de estados actualizados, ordenamiento topológico modificado y ordenamiento estático en forma decreciente de máxima recompensa (Wingate, 2005), con el objeto de obtener el más corto tiempo de solución en considerablemente grandes MDPs. También se comparó con otros algoritmos del estado del arte. Al menos en el dominio *Sailing Strategics* (Vanderbei, 1996), correspondiente a un MDP de ruta estocástica más corta de gran dimensión, el uso del método de ordenamiento aquí propuesto obtuvo el más corto tiempo de solución frente a una tarea altamente cíclica, eliminando de esta manera, la intratabilidad computacional.

REFERENCIAS

Agrawal, S. and Roth, D. (2002). *Learning a Sparse Representation for Object Detection*. 7th European Conference on Computer Vision, Copenhagen, Denmark, pp. 1-15.

Bellman, R. E. (1957). *Dynamic Programming.* Princeton United Press, Princeton, USA.

Boutilier, C., Dean, T. and Hanks, S. (1999). Decision-theoretic planning: structural assumptions and computational leverage. *Journal of Artificial Intelligence Research,* 11, pp 1-94.

Chang, I. and Soo, H. (2007). Simulation-based algorithms for Markov decision processes. *Communications and Control Engineering,* Springer Verlag London Limited.

Dai, P. and Goldsmith, J. (2007). *Topological Value Iteration Algorithm for Markov Decision Processes.* 20th International Joint Conference on Artificial Intelligence, IJCAI, pp 1860-1865, Hyderabad, India.

Dibangoye, J. S., Chaib-draa, B. and Mouaddib, A. (2008). *A Novel Prioritization Technique for Solving Markov Decision Processes.* 21st International FLAIRS (The Florida Artificial Intelligence Research Society) Conference, Association for the Advancement of Artificial Intelligence, Florida, USA.

Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Appendix A: List of NP-Complete Problems, W. H. Freeman.

Hinderer, K. and Waldmann, K. H. (2003). The critical discount factor for Finite Markovian Decision Processes with an absorbing set. *Mathematical Methods of Operations Research,* Springer Verlag, 57, pp 1-19.

Kirk, W. A., Khamsi, M. A. (2001). *An Introduction to Metric Spaces and Fixed Point Theory*. John Wiley, New York, USA.

Li, L. (2009). *A Unifying Framework for Computational Reinforcement Learning Theory.* PhD Thesis, The State University of New Jersey (New Brunswick, NJ, USA.

Littman, M. L., Dean, T. L. and Kaelbling, L. P. (1995). *On the Complexity of Solving Markov Decision Problems*. 11th International Conference on Uncertainty in Artificial Intelligence, pp 394-402, Montreal, Quebec.

McMahan, H. B. and Gordon, G. (2005). *Fast Exact Planning in Markov Decision Processes*. 15th International Conference on Automated Planning and Scheduling, Monterey, CA, USA.

Puterman, M. L. (2005). *Markov Decision Processes.* Wiley Interscience Editors, New York, USA.

Reyes, A., Ibargüengoytia, P., Sucar, L. E. and Morales, E. (2006). *Abstraction and Refinement for Solving Continuous Markov Decision Processes*. 3rd European Workshop on Probabilistic Graphical Models, pp 263-270, Prague, Czech Republic.

Russell, S. (2004). *Artificial Intelligence: A Modern Approach.* 2nd Edition, Making Complex Decisions (Ch-17), Pearson Prentice Hill Ed., USA.

Tijms, H. C. (2003). *A First Course in Stochastic Models.* Wiley Ed., Discrete-Time Markov Decision Processes (Ch-6), UK.

Vanderbei, R. J. (1996). *Optimal Sailing Strategies, Statistics and Operations Research Program.* University of Princeton (http://orfe.princeton.edu/~rvdb/sail/sail.html), USA.

Wingate, D. and Seppi, K. D. (2005). Prioritization Methods for Accelerating MDP Solvers, *Journal of Machine Learning Research*, 6, pp 851-881.