UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Departamento de Sistemas Informáticos y Computación

SIMPLIFICACIÓN DE LOS PROCESOS DE DECISIÓN DE MARKOV MEDIANTE REGLAMENTACIÓN DE ACCIONES Y PRIORIZACIÓN DE ESTADOS

Presentada por **Ma. de Guadalupe García Hernández**

Dirigida por Dr. Dn. José Ruiz Pinales
y por Dr. Dn. Alberto Reyes Ballesteros
Tutorada por Dra. Dña. Eva Onaindía De La Rivaherrera

PARA LA OBTENCIÓN DEL GRADO DE

DOCTOR EN RECONOCIMIENTO DE FORMAS E INTELIGENCIA ARTIFICIAL

POR LA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Valencia, España

Noviembre de 2012

A la memoria de mi papá, Ing. Felipe García Cardona (1931-2012).

A Guillermo, mi esposo. A Guillermo, Jessy y Edgar, mis hijos.

A Carolina, mi mamá.

"En lugar de preocuparnos acerca de si una máquina puede ser inteligente, es más importante desarrollar software que sea realmente inteligente".

Oliver G. Selfridge

"La inteligencia artificial de hoy en día se centra en buscar nuevas maneras de conectar personas con ordenadores, personas con conocimiento, personas con el mundo físico y personas con personas".

Patrick Winston

Índice general

Índice de tablas

Índice de figuras

Resumen

El problema de resolver grandes procesos de decisión de Markov con precisión y rapidez ha conducido a un reto computacional. Dado que el esfuerzo computacional es considerable, la investigación actual se centra en la búsqueda de técnicas superiores de aceleración. Por ejemplo, las propiedades de convergencia de los métodos de solución actuales dependen, en gran medida, del orden de las operaciones de actualización. Por un lado, algoritmos tales como el de ordenamiento topológico han sido capaces de encontrar buenos ordenamientos, pero sus costes de inicio han sido usualmente altos. Por otro lado, los métodos de ruta más corta tales como el clásico algoritmo de Dijkstra, que está basado en colas de prioridad, han sido aplicados exitosamente a la solución de procesos de decisión de Markov de ruta determinística más corta. Aquí se propone un nuevo algoritmo de iteración de valor basado en el algoritmo de Dijkstra para resolver procesos de decisión de Markov de ruta estocástica más corta. Los resultados experimentales obtenidos en un problema de estrategias de navegación marítima muestran la factibilidad del enfoque propuesto.

Abstract

The problem of solving large Markov decision processes accurately and quickly has led to a computational challenge. Since the computational effort is considerable, current research focuses on finding superior acceleration techniques. For instance, the convergence properties of current solution methods depend, to a great extent, on the order of backup operations. On one hand, algorithms such as a topological sorting are able to find good orderings, but their overhead is usually high. On the other hand, shortest path methods, such as Dijkstra's algorithm which is based on priority queues, have been applied successfully to the solution of deterministic shortest-path Markov decision processes. Here, we propose a new value iteration algorithm based on Dijkstra's algorithm for solving shortest-path Markov decision processes. The experimental results on a sailing strategies problem show the feasibility of our approach.

Resume

El problema de resoldre grans processos de decisió de Markov amb precisió i rapidesa ha conduït a un repte computacional. Atés que lésforç computacional és considerable, la investigació actual se centra en la busca de tècniques superiors dácceleració. Per exemple, les propietats de convergència dels mètodes de solució actuals depenen, en gran manera, de lórde de les operacions dáctualització. Dúna banda, algoritmes com ara el dórdenament topològic han sigut capaços de trobar bons ordenaments, però els seus costos dínici han sigut usualment alts. Dáltra banda, els mètodes de ruta més curta com ara el clàssic algoritme de Dijkstra, que està basat en cues de prioritat, han sigut aplicats reeixidament a la solució de processos de decisió de Markov de ruta determinística més curta. Ací es proposa un nou algoritme díteració de valor basat en lálgoritme de Dijkstra per a resoldre processos de decisió de Markov de ruta estocástica més curta. Els resultats experimentals obtinguts en un problema déstratègies de navegació marítima mostren la factibilitat de lénfocament proposat.

Capítulo 1

Introducción

Para que se construyan rumbos de acción en ambientes reales, se debe considerar que las acciones pueden tener efectos distintos en el mundo (no determinismo) y se debe ponderar el potencial de algún plan alternativo para alcanzar las metas del problema, considerando sus costes y recompensas (metas extendidas). Al respecto, la planificación basada en teoría de decisiones [Boutilier, 1999] ha permitido solucionar problemas estocásticos estableciendo rumbos de acción que involucran cantidades de información difíciles de procesar por el ser humano, evaluando sus fortalezas y debilidades con base en las teorías de probabilidad y de utilidad. Esta metodología ha incrementado últimamente su investigación debido a los avances en representación e inferencia Bayesianas, así como al éxito de los **procesos de decisión de Markov** (MDP) en problemas de investigación de operaciones, teoría de control, economía e inteligencia artificial, entre otros. Estos procesos están basados en la ecuación de Richard Bellman [Bellman, 1957], la cual ha sido resuelta exitosamente mediante programación dinámica por el clásico algoritmo de iteración de valor. Sin embargo, este algoritmo y sus variantes han presentado intratabilidad al intentar resolver problemas complejos, debido a que su espacio de búsqueda crece cuadráticamente con el número de variables [Puterman, 1994].

1.1. El Problema de Planificación

Desde el punto de vista de la Inteligencia Artificial, el problema de planificación se especifica normalmente de la siguiente manera: dada una descripción del estado actual de un sistema, un conjunto de acciones que pueden realizarse sobre el sistema y una descripción de un conjunto de estados meta para el sistema, se debe obtener una secuencia de acciones (política) para transformar el estado actual en un estado meta [Puterman, 2005]. Por ejemplo, considere el dominio de las plantas eléctricas [Reyes, 2006a], donde un sistema de generación consta de un conjunto de equipos de proceso como turbinas de gas (TG), recuperador de

calor (RC), quemadores posteriores (QP) y turbinas de vapor (TV), y cada uno de los cuales puede mantener un estado de operación coherente con el resto de los demás. Las acciones probables permiten encender un equipo para ponerlo en operación y la meta es una configuración particular de cada componente, como por ejemplo "todos los equipos encendidos". El problema consiste en determinar cuál es la secuencia de acciones necesaria, incluyendo aquellas de naturaleza paralela, para llevar desde una planta apagada a su estado de operación estable (vea Figura 1.1).

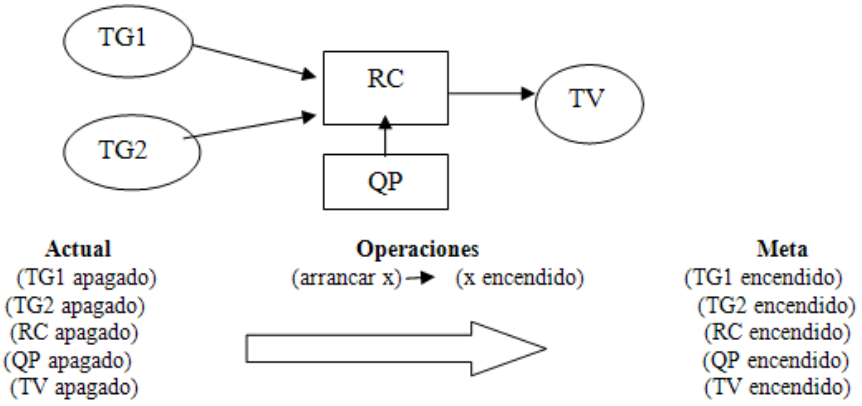

Figura 1.1: Ejemplo de planificación en el dominio de plantas eléctricas (Cortesía Dr. Alberto Reyes [Reyes, 2006a])

En este sentido, para que un agente encuentre planes que se apeguen al mundo real debe tomar en cuenta que el mundo cambia dinámicamente, que las acciones pueden tener diferentes efectos o que el conocimiento de un estado puede ser incierto [Ghallab, 2004]. Además, el agente debe balancear el potencial de algún plan para alcanzar una meta, con el riesgo de producir estados indeseables o planes costosos de elaborar. Esto elimina tres suposiciones restrictivas de la planificación clásica como son el determinismo, la observabilidad total y las metas restringidas al éxito o al fracaso [Ghallab, 2004]. A continuación se describen estas restricciones.

1.1.1. No Determinismo

La suposición de que los efectos de una acción son deterministas es irreal e impráctica ya que, como es bien sabido, nada es totalmente predecible. El no determinismo toma una postura más realista y útil modelando el hecho de que después de la aplicación de una acción o la aparición de un evento exógeno, algunos efectos son más factibles de suceder que otros. El no determinismo pue-

de modelarse asociando probabilidades a los resultados de una acción o evento [Ghallab, 2004].

Considere como ejemplo el problema de generar energía eléctrica mediante una turbina de vapor (TV) de una central de ciclo combinado. La meta es llevar el generador de la turbina de vapor el estado de "generación a plena carga" y "mayor o igual que el 90 % de su capacidad total". Esta meta puede representarse mediante los hechos tipo lista "estado TV generando" y "menor que el 90 % de su capacidad total", donde el primer término es el nombre del predicado o función y los restantes sus argumentos [Reyes, 2006a].

La Figura 1.2 muestra tres descripciones con diferente detalle de la acción "cerrar interruptorPpal" para alcanzar la meta de "generar a plena carga". La primera es determinística y podría usarse con alguno de los operadores extras (vea Figura 1.3 inciso *a*) para formar el plan de dos pasos que alcanza la meta con toda certeza: "cerrar interruptorPpal" y "girar TV". El segundo operador "cerrar interruptorPpal" del inciso *b* de la Figura 1.2 tiene dos conjuntos de efectos alternos, modelando dos estados diferentes del mundo que pudieran darse al aplicar el operador. Los números ubicados sobre los arcos son las probabilidades de llegar a cada conjunto de estados.

Bajo este modelo, el plan de dos pasos del caso anterior sólo alcanzaría la meta con una probabilidad de éxito del 70 %. En la tercera formulación del operador "cerrar interruptorPpal" del inciso *c* de la Figura 1.2, son posibles diferentes conjuntos de resultados, los cuales están basados en otras condiciones existentes al aplicar el operador y los arcos ahora muestran probabilidades condicionales. El plan de dos pasos aún tiene probabilidad de éxito del 70 %. Sin embargo, el plan de tres pasos: "encender QP", "girar TV", "cerrar interruptorPpal" tiene el 90 % de probabilidad de éxito.

Cabe señalar que la acción "encender QP" significa encender los quemadores posteriores del recuperador de calor, con la idea de soportar variaciones en el flujo de gases calientes saliendo de las turbinas de gas que permiten la generación de vapor [Reyes, 2006a].

En la anterior figura es notorio el aumento del grado de complejidad habido desde la planificación determinísta hasta la planificación probabilista condicional, pasando por la planificación probabilista simple.

1.1.2. Observabilidad Parcial

Durante algún tiempo de ejecución, el estado de un sistema puede ser parcialmente accesible y, por tanto, puede dar lugar a estados indistinguibles. En algunos casos ciertas variables nunca son observables, en otros son observables solamente en ciertos estados y en otros sólo son observables después de

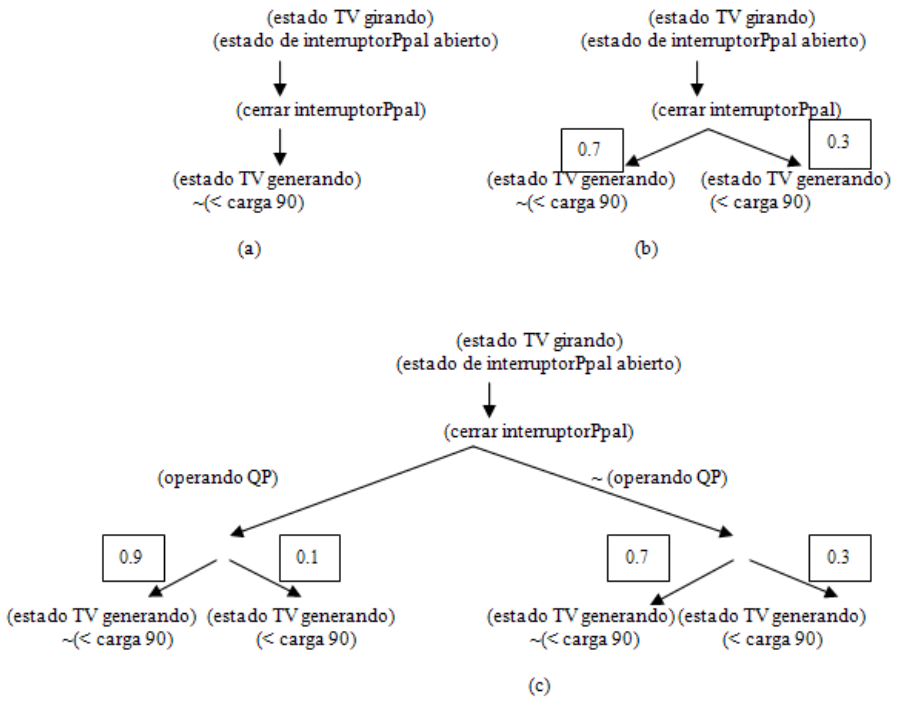

Figura 1.2: Versiones del operador: (a) determinista, (b) probabilista simple, (c) probabilista condicional (Cortesía Dr. Alberto Reyes [Reyes, 2006a]).

una rutina de sensado [Ghallab, 2004]. De ahí surgieron los MDP parcialmente observables (POMDP) [Amstron, 1965], que son aquellos MDP en los que la observación de un estado está limitada y, por lo tanto, es incierta. De ahí que su espacio de estados quede formado por un conjunto de *estados determinados* y por otro de *estados probables*.

Tanto teórica como experimentalmente se ha demostrado que la planificación con observabilidad parcial es un problema difícil de resolver. Los POMDP también han sido utilizados en investigación de operaciones y en teoría de control. Por otro lado, dado que una observación devuelve a un conjunto de estados, la consecuencia principal de la observabilidad parcial es que la magnitud de ese conjunto después de la observación puede ser exponencialmente mayor que la del conjunto de estados del dominio. Peor aún en el caso de planificación bajo incertidumbre, donde los observadores devuelven distribuciones de probabilidad sobre conjuntos de estados que pueden hacer infinito el espacio de búsqueda [Reyes, 2006b].

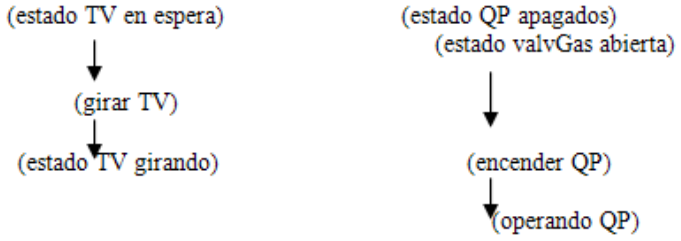

Figura 1.3: Otros operadores con precondiciones y efectos deterministas.

1.1.3. Metas Extendidas

En sistemas del mundo real, las metas necesitan especificar implícitamente consideraciones acerca de posible fallo, de tal suerte que cuando intente alcanzar una meta entonces el sistema garantice que, si no lo logra, al menos se mantendrá en un estado seguro. Una forma de lograr que las metas especifiquen las preferencias de un agente es mediante el uso de funciones de utilidad (coste y recompensa). Al respecto, una de las técnicas alternativas es representar las metas extendidas mediante fórmulas de lógica temporal [Ghallab, 2004]. Sin embargo, la planificación con metas extendidas es un problema difícil de resolver, ya que estas metas agregan complejidad adicional a un problema de por sí complicado.

1.2. Planificación basada en Teoría de Decisiones

Atendiendo a las anteriores restricciones el agente deberá encontrar una política que optimice alguna utilidad esperada. Para ello existen enfoques que encuentran tales políticas basándose en el marco de la teoría de decisiones [Luce, 1957], la cual ha permitido evaluar fortalezas y debilidades de planes alternativos, con base en teorías de probabilidad y de utilidad.

Dada una distribución de probabilidades sobre los posibles efectos de una acción en un estado y una función razonable de referencia sobre los resultados de una decisión, es posible definir una función de utilidad tal que, cada vez que un agente prefiera un plan sobre otro, el plan preferido tenga la mayor utilidad esperada. Es notorio que la planificación automática y la teoría de decisiones son técnicas complementarias, resultando así la **planificación basada en teoría de decisiones** [Boutilier, 1999].

Es importante señalar que el interés mostrado en este campo se debe a los avances en representación e inferencia Bayesianas y al éxito obtenido por las técnicas Markovianas en áreas como reconocimiento de voz, así como a sus

cercanas técnicas de aprendizaje por refuerzo. Más aún, con los ordenadores actuales -que tienen mayor potencia y velocidad- se han construido planificadores basados en teoría de decisiones que, en general, tienen espacios de búsqueda más complejos que sus contrapartes clásicas.

Por otro lado, una de las estrategias utilizadas exitosamente por la planificación basada en teoría de decisiones es la de los **procesos de decisión de Markov** (MDP) [Puterman, 2005], los cuales ofrecen un marco matemático para modelar y derivar políticas óptimas. Este formalismo está basado en la ecuación de Bellman [Bellman, 1954, 1957] y ha sido estudiado a profundidad desde la década de los 60, en los campos de análisis de decisiones, investigación de operaciones, economía e inteligencia artificial, iniciando con el trabajo seminal de Howard [Howard, 1960]. De ahí se derivaron otras importantes contribuciones [Bertsekas, 1987] [Puterman, 1994], aunque no lograron la automatización.

En este sentido, el **algoritmo de iteración de valor**, perteneciente a programación dinámica, ha resuelto con cierto éxito a los MDP, pero su convergencia es lenta [Littman, 1995] debido a esta depende fuertemente del orden de sus actualizaciones. Esta lenta convergencia ha sido abordada con cierto éxito mediante búsqueda heurística y priorización [Dai, 2007a].

En el primer caso, la búsqueda heurística combinada con programación dinámica es usada para reducir el número de estados relevantes, así como el número de expansiones de búsqueda. Al respecto, Hansen & Zilberstein [Hansen, 2001] consideraron solamente parte del espacio de estados para construir un grafo de solución parcial, buscando implícitamente desde el estado inicial hacia el estado meta, y expandiendo la rama más prometedora de un MDP de acuerdo con una función heurística. Bhuma & Goldsmith [Bhuma, 2003] extendieron este enfoque usando un algoritmo de búsqueda heurística bidireccional. Por otro lado, Bonet & Geffner [Bonet, 2003a,b] propusieron dos algoritmos de búsqueda heurística que usan una técnica de etiquetado inteligente para marcar estados irrelevantes. Después ellos mismos exploraron la búsqueda "primero en profundidad" para la solución de MDP [Bonet, 2006].

En el segundo caso, los métodos de priorización están basados en la observación de que, en cada iteración, la función de valor usualmente cambia solamente para un reducido conjunto de estados. De modo que estos métodos priorizan cada actualización con el objeto de reducir el número de evaluaciones [Moore, 1993] [Dai, 2007b]. En este sentido Ferguson & Stentz [Ferguson, 2004] propusieron otro método de priorización que denominaron programación dinámica enfocada, donde las prioridades son calculadas de forma diferente al barrido priorizado. Después Dai & Goldsmith [Dai, 2007a] extendieron el enfoque de Bhuma & Goldsmith utilizando concurrentemente diferentes puntos de inicio. Posteriormente ellos mismos propusieron un algoritmo de iteración de valor topológico, el cual agrupa estados que están mutua y causalmente relacionados en un metaestado, para el caso de estados conectados fuertemente (o MDP con

grafos cíclicos) [Dai, 2007b].

Po último, se han propuesto otros enfoques tales como el ordenamiento topológico [Wingate, 2005] y los métodos de ruta más corta [McMahan, 2005]. Los algoritmos de ordenamiento topológico pueden ser usados para encontrar buenos ordenamientos de actualizaciones, pero con alto coste de inicio [Wingate, 2005]. En cambio, los métodos de ruta más corta han sido aplicados para resolver MDP determinísticos de forma exitosa [McMahan, 2005a,b]. Sin embargo, estos no han garantizado la convergencia ante MDP no deterministas [Li, 2009].

1.3. Motivación y Objetivos de la Tesis

En esta tesis se considera el problema de encontrar la política óptima en una clase de **MDP no deterministas, positivos y con estados terminales absorbentes**, que son equivalentes a problemas de **ruta estocástica más corta** [Bertsekas, 1995]. Cabe señalar que los MDP aquí estudiados son planos (no factorizados) y totalmente observables.

Al respecto, McMahan & Gordon [McMahan, 2005a,b] propusieron el método del barrido priorizado mejorado para resolver problemas de ruta estocástica más corta con un estado meta, basado en el clásico algoritmo de Dijkstra. Las ventajas de su método son la reducción del algoritmo de Dijkstra para el caso de cadenas de Markov (MDP deterministas) y la mejora en la velocidad cuando es comparado con otros métodos tales como el barrido priorizado de Moore & Atkeson [Moore, 1993] y la programación dinámica enfocada de Dai & Goldsmith [Dai, 2007b]. Desafortunadamente el método de McMahan & Gordon no ha garantizado la convergencia a la política óptima para el caso de MDP no deterministas [Dai, 2007a,b] [Li, 2009].

Para abordar dicho problema, en esta tesis se propone y se demuestra la factibilidad de un nuevo algoritmo de iteración de valor priorizado basado en el algoritmo de Dijkstra que, además, utiliza una reglamentación de acciones. Es importante destacar que el enfoque propuesto garantiza la convergencia para el caso de problemas complejos de ruta estocástica más corta y es capaz de tratar con múltiples estados meta y de inicio, **disminuyendo en buena medida la complejidad de MDP de ruta más corta discretos, estacionarios y de considerables dimensiones**.

Por otro lado, los MDP han tenido importantes aplicaciones en el mundo real como es el caso de control de procesos, donde algunas variables cambian dinámicamente por la operación de dispositivos tales como válvulas, interruptores de equipo y por la presencia de eventos exógenos (no controlables). En estos casos, si el controlador del proceso no considera la posibilidad de fallo de dichos dispositivos, entonces no será capaz de tomar una decisión inteligente ante su

eventual presencia. Este ejemplo viene a ser un problema Markoviano complejo, donde la incertidumbre juega un papel importante a considerar durante la búsqueda de soluciones. A continuación se enlistan otras aplicaciones de este formalismo:

- simulación de planificación de movimientos robóticos,
- simulación de procesos industriales,
- planificación del sensado,
- planificación de operaciones de alto nivel en centros de control de energía,
- planificación del mantenimiento industrial,
- elaboración de planes de emergencia en zonas de alto riesgo,
- simulación de una planta de vapor de ciclo combinado,
- asistentes inteligentes (sector productivo, gestión administrativa),
- tutores inteligentes (sector educativo),
- asistentes resolutores de problemas de tratabilidad.

Una vez establecida la importancia de resolver grandes MDP en tiempo tratable, a continuación se presenta el objetivo general de esta tesis: **simplificar los MDP de ruta más corta de considerables dimensiones**, mediante la reducción del espacio de búsqueda del algoritmo de iteración de valor aplicando:

- **una reglamentación de acciones**, que a través de minería de datos obtiene acciones en función de estado, para que el algoritmo de iteración de valor solamente calcule sobre el subconjunto de acciones probables para el estado en evaluación.
- **una priorización de estados** basada en el clásico algoritmo de Dijkstra para resolver problemas de ruta estocástica más corta, para que el algoritmo de iteración de valor calcule y seleccione la máxima función de valor de los predecesores de cada estado relevante, desde un estado meta hasta uno de inicio.

Por último, se prueba la robustez de esta propuesta en una tarea compleja de ruta estocástica más corta de navegación marítima.

1.4. Aportaciones

La presente tesis hace las siguientes **aportaciones** a la comunidad de planificación automática:

1. Representación y aprendizaje automático de acciones en función de estado, contenidas en reglas de asociación, mediante una técnica eficiente de minería de datos.
2. Nuevo algoritmo de iteración de valor basado en reglas de asociación, que sólo calcula sobre acciones en función del estado en evaluación [Garcia-Hernandez, 2009].
3. Nuevo algoritmo de iteración de valor priorizado con reglamentación de acciones, basado en el algoritmo de Dijkstra, que garantiza la convergencia en problemas de ruta estocástica más corta, que es capaz de tratar con múltiples estados meta y de inicio y que hace ordenamiento dinámico con métrica de prioridad de la máxima función de valor de los vecinos inmediatos de cada estado prometedor [Garcia-Hernandez, 2012a,b].

De esta manera el enfoque propuesto en esta tesis viene a robustecer a los sistemas de planificación basados en teoría de decisiones frente a problemas complejos.

1.5. Organización del Trabajo

La presente tesis está estructurada en siete capítulos. En el **Capítulo 2** se estudian los MDP, su complejidad, sus técnicas de solución y el estado del arte en estas técnicas. En el **Capítulo 3** se estudian dos estrategias del estado del arte que prometen reducir el espacio de búsqueda, tanto de acciones como de estados. Estas estrategias son: reglamentación de acciones (por minería de datos) y priorización de estados, respectivamente. En el **Capítulo 4** está contenida la propuesta de tesis y se divide en dos secciones. En la primera sección se aplica una reglamentación de acciones durante el proceso de inferencia del algoritmo de iteración de valor. En la segunda sección, al algoritmo resultante de la anterior sección se le aplica una nueva técnica de priorización de estados basada en el algoritmo de Dijkstra, obteniendo así el enfoque propuesto. En el **Capítulo 5** se muestra la preparación del ambiente de prueba utilizando un simulador de planificación de movimientos robóticos, con el objeto de resolver una tarea compleja de ruta estocástica más corta de navegación marítima. En el **Anexo 1** se presenta un ejemplo de reglamentación de acciones entregada por el algoritmo de minería de datos utilizado. En el **Anexo 2** se muestra el procedimiento para ejecutar al simulador de planificación de movimientos robóticos utilizado. En los **Anexos 3, 4, 5 y 6** se muestran ejecuciones de variantes del algoritmo de iteración de valor síncronas, asíncronas, con ordenamiento de estados sin barrido priorizado y con barrido priorizado (este último incluye al enfoque propuesto), respectivamente, en el dominio utilizado. En el **Capítulo 6** se muestran los resultados experimentales obtenidos por el enfoque propuesto y su comparación con otros enfoques pertenecientes al estado del arte. Por último, en el **Capítulo 7** se presentan las conclusiones de esta tesis.

Capítulo 2

Procesos de Decisión de Markov

Las técnicas Markovianas modelan un **problema de decisión secuencial**, en el cual un sistema evoluciona en el tiempo y es controlado por un agente. Sus dominios se modelan como sistemas estocásticos, sus metas se definen por sus funciones utilidad/coste (recompensa/descuento), donde el problema de planificación se transforma en un problema de optimización. De ahí que su solución sea la **política óptima**, asignando la mejor acción en cada estado del espacio de estados del problema. Por otro lado, la dinámica del sistema queda gobernada por una función de probabilidad de transición de estados, dados un estado actual y una acción probable en dicho estado [Puterman, 2005].

En este tipo de problemas, en cada paso el agente recibe una recompensa numérica que, en general, depende del estado en evaluación y de la acción aplicada. En este contexto, una acción es un evento perturbador ocasionado por un agente con el objeto de cambiar al estado de un sistema. El agente puede tener control sobre las acciones que desea ejecutar y sobre el momento en que las aplicará, aunque sus efectos no sean totalmente predecibles. En cambio, los eventos exógenos no se encuentran bajo control del agente y su ocurrencia puede ser parcialmente predecible [Puterman, 2005].

Al problema de encontrar una estrategia reactiva de control (o política de acción) en problemas de decisión secuencial, que maximice la recompensa esperada en el tiempo, se le conoce como **proceso de decisión de Markov (MDP** por sus siglas en inglés), llamado así en honor del ruso Andrei A. Markov, estudioso de la Estadística. El MDP supone que el agente siempre conoce el estado en que se encuentra al momento de iniciar la ejecución de las acciones del dominio (observación total), y que la probabilidad de transición depende solamente de este estado y no de su historia (esta es la **propiedad de Markov**), afectada por un factor de descuento que impacta negativamente a la utilidad [Puterman,

2005].

2.1. Formalismo

Los MDP proveen un marco matemático intuitivo para modelar problemas de decisión secuencial en ambientes dinámicos inciertos [Bellman, 1957] [Puterman, 2005]. Formalmente un MDP es una 4-*tupla* $< S, A, T, R >$, donde S es un conjunto finito de estados $\{s_1, \ldots, s_n\}$, A es un conjunto finito de acciones $\{a_1, \ldots, a_n\}$, $T : S \times A \longrightarrow \Pi(S)$ es la **función de transición de estados**, la cual asocia un conjunto de probables estados dada una acción en el estado en evaluación. De ahí que la probabilidad de transición para alcanzar al estado s' luego de aplicar la acción a en el estado s estará dada por $T(a, s, s')$. La función recompensa es $R : S \times A$ y la recompensa obtenida cuando se aplica la acción a en el estado s es $R(s, a)$. Una política es definida como la función $\pi : S \to A$, que selecciona una acción para cada estado [Puterman, 2005].

Con el objeto de evaluar una política es necesario establecer si el número de etapas es finito (**horizonte finito**), o en su defecto establecer el factor de descuento γ que aplicará a cada estado visitado (**horizonte infinito**), donde $[0 \leq \gamma \leq 1]$. Este parámetro es usado para descontar las recompensas futuras en una proporción geométrica. Por lo que, cuando las recompensas futuras se consideran insignificantes, el factor de descuento será $\gamma = 0$. En cambio, cuando la recompensa es idéntica al descuento esperado a futuro, su valor será $\gamma = 1$, por lo que se recomienda trabajar con $\gamma < 1$. Además, el proceso de transición se considera como **estacionario**, es decir que sólo depende del estado y no del tiempo.

Dada una política π la función de valor o utilidad $U^\pi(s)$ es el total de las recompensas esperadas (E) para que el agente alcance una meta en un horizonte infinito, de acuerdo con la siguiente expresión:

$$U^\pi(s) = E[\sum_t \gamma^t R(s_t, \pi(s_t))|s_0 = s] \tag{2.1}$$

Cuando $\gamma = 1$, las recompensas son aditivas y es un caso especial de recompensas descontadas. Por otro lado, es importante destacar que una de las características de la ecuación (2.1) es el cuidadoso balance entre coste y recompensa. Entonces la función de valor óptima estará dada por:

$$U^*(s) = max_\pi U^\pi(s) \tag{2.2}$$

Es bien conocido que la función de valor óptima $U^*(s)$ en la etapa t es dada por

la ecuación de Richard Bellman [Bellman, 1957]:

$$U_t^*(s) = \max_a \left\{ R(s,a) + \gamma \sum_{s'} P(a,s,s') U_{t+1}^*(s') \right\} \tag{2.3}$$

La actualización de estas ecuaciones pertenece a programación dinámica [Puterman, 1994], mediante **el algoritmo de iteración de valor** y **el algoritmo de iteración de política**, y a programación lineal. Con esta actualización se encuentra la función de valor óptima $U^*(s)$ y la política óptima π^* para problemas con horizonte finito [Chang, 2007]. La idea básica es calcular la utilidad de cada estado y usar esas utilidades para seleccionar la política (es decir la acción) que maximice la utilidad esperada en cada estado.

Sin embargo, el problema principal de generar políticas óptimas en este formalismo es frecuentemente un reto computacional debido al **requerimiento típico de enumerar enteramente los espacios de estados y/o de acciones**. Obviamente este formalismo se simplifica considerablemente aplicando técnicas de aceleración de la convergencia, en las que el planificador busca la ruta que ofrezca mayor recompensa con menor coste.

2.2. Técnicas de Solución

Al tratar de resolver la ecuación de Bellman resulta una ecuación para cada estado en problemas de horizonte infinito [Puterman, 2005]. Su optimización conduce a la obtención de un sistema de ecuaciones no lineales debido a que contienen una maximización. Para resolver a este sistema se han propuesto las siguientes técnicas:

- programación dinámica,
- programación lineal,
- aprendizaje por refuerzo.

La distinción básica entre estas técnicas es que **las dos primeras son informadas** debido a que conocen la función de transición y la función de valor o utilidad. En cambio **la tercera no tiene acceso a esta información**, por lo que requiere explorar para aprender estados probables, resultando por ello más compleja que las dos primeras y con la posibilidad de que no se cumpla con la citada propiedad de Markov. Otra distinción es que la programación dinámica (inducción regresiva) solo es aplicable a horizontes finitos, en cambio las otras dos se pueden aplicar a horizontes infinitos [Puterman, 2005]. Por otro lado, una vez que el aprendizaje por refuerzo obtiene la información requerida entonces, obviamente, se transforma en un método informado. A continuación se describen estas técnicas.

2.2.1. Programación Dinámica

Los distintos enfoques de programación dinámica explícitamente almacenan funciones de valor del espacio de estados e iterativamente actualizan los valores para cada estado, hasta que el cambio de la función de valor es menor o igual al umbral previamente establecido. Entonces se dice que la función de valor converge a su valor óptimo, con su correspondiente acción o política óptima. Por lo que la programación dinámica ofrece un enfoque unificado para resolver problemas de control estocástico como lo son los MDP. Los métodos más populares de programación dinámica son el algoritmo de iteración de valor y el algoritmo de iteración de política, que a continuación se describen.

2.2.1.1. Iteración de Valor

El algoritmo de iteración de valor [Puterman, 1994] aplica actualizaciones sucesivas de la función de valor para cada estado $s \in S$, al ejecutar la ecuación de Bellman (vea ecuación (2.3)), siendo la recompensa inmediata del estado en evaluación su valor inicial:

$$\hat{U}(s) = \max_{a} \left\{ R(s,a) + \gamma \sum_{s'} P(a,s,s')U(s') \right\} \tag{2.4}$$

y calcula la parte derecha de la ecuación. De esta manera se actualiza la utilidad de cada estado a partir de las utilidades de sus estados vecinos inmediatos. Entonces se repite el proceso hasta que alcanza la convergencia. Cabe destacar que si esta actualización se aplica indefinidamente, entonces queda garantizado que se alcanzará un estado de equilibrio [Puterman, 2005]. El proceso se detiene cuando la diferencia entre las dos últimas iteraciones es menor que el máximo error permitido, ε. Entonces, la iteración de valor converge finalmente a un único conjunto de soluciones (vea Algoritmo 2.1).

Por otro lado, esta actualización es una contracción debida a la presencia del factor de descuento γ sobre el espacio de vectores de utilidad, cuyo valor usualmente es menor que la unidad. La contracción es una función del argumento que, cuando se aplica sucesivamente sobre dos argumentos diferentes, resultan otros dos valores más próximos entre sí que sus valores originales. En este sentido, a partir de la propiedad de contracción es demostrable que si la modificación es pequeña entre dos iteraciones, entonces el error ε es también pequeño en comparación con la función de utilidad verdadera. Con lo anterior se detiene el proceso iterativo del algoritmo.

De esta manera, la aproximación dinámica se perfila como una herramienta potencialmente poderosa para el control estocástico a escala industrial. Sin embargo, en la literatura se han encontrado errores significativos debido a la

Algoritmo 2. 1 Iteración de Valor [Puterman, 1994].

$(S, A, R, P, \gamma, \varepsilon)$

1: **para todo** $s \in S, U^o = 0$
2: $t = 0$
3: $t = t + 1$
4: **para todo** $s \in S$ **hacer**
5: **para todo** $a \in A$ **hacer**
6: $U^t(s,a) = R(s) + \gamma \sum_{s' \in S} P(a,s,s') U^{t-1}(s')$
7: $\pi^t(s) = \arg \max_a U^t(s,a)$
8: $U^t(s) = U^t(s, \pi^t(s))$
9: **fin para**
10: **fin para**
11: **si** $\left(\max_s |U^t(s) - U^{t-1}(s)|\right) \geq \varepsilon$ ir al paso 3.
12: **regresa** π^t

pobre comprensión de los algoritmos de programación dinámica, así como de su implementación deficiente [De Farías, 2003].

Sea $\{U_n | n = 0, 1, ...\}$ la secuencia de funciones de valor obtenidas por iteración de valor. Entonces puede ser demostrado que cada función de valor satisface a $|U_n - U^*| \leq \gamma^n |U_o - U^*|$. Así, por el clásico **teorema de Banach del Punto Fijo** [Kirk, 2001], se puede inferir que la iteración de valor converge a la función de valor óptima U^*.

Cabe señalar que el potencial éxito de iteración de valor para la solución de problemas de MDP de gran escala viene del hecho de que las funciones de valor obtenidas pueden ser usadas como cotas para la función de valor óptima [Tijms, 2003].

La complejidad computacional de una iteración de este algoritmo es $O(|S|^2|A|)$. Sin embargo, el número de iteraciones requerido puede llegar a ser considerablemente grande con la escalabilidad. Afortunadamente ha sido demostrado en [Littman, 1995] que una cota máxima del número de iteraciones requeridas por la iteración de valor para alcanzar una solución ε-óptima está dada por:

$$n_{it} \leq \frac{b + \log(\frac{1}{\varepsilon}) + \log(\frac{1}{1-\gamma}) + 1}{1 - \gamma} \tag{2.5}$$

donde $(0 < \gamma < 1)$, B es el número de *bits* usado para codificar a las funciones recompensa y probabilidad de transición de estados, y ε es el umbral del *error de Bellman* para el estado en evaluación $B_t(s)$:

$$B_t(s) = max_{\alpha \epsilon A}\{R(s,a) + \gamma \sum_{s' \epsilon S} T(s,a,s') U_t(s')\} - U_t(s) \tag{2.6}$$

En el estado del arte se reportan **variantes aceleradas de iteración de valor** que tienen que ver con el criterio de paro del mismo o con el manejo de variables. Una de ellas es la **actualización asíncrona** [Puterman, 2005], que consiste en actualizar los valores de la función de valor (o utilidad) tan pronto como estén disponibles. A esta variante se le conoce como **actualización de Gauss-Seidel**, la cual cumple con la siguiente ecuación:

$$U^t(s) = \max_a \left\{ R(s,a) + \gamma \sum_{s'<s} P(a,s,s')U^t(s') + \gamma \sum_{s' \geqslant s} P(a,s,s')U^{t-1}(s') \right\} \quad (2.7)$$

Por otro lado, el algoritmo de iteración de política converge con menos iteraciones que el algoritmo de iteración de valor, pero requiere que un sistema de ecuaciones lineales sea resuelto en cada iteración. Por ello, un enfoque combinado (denominado **iteración de política modificada**) [Puterman, 2005] puede explotar las ventajas de ambos. En este sentido, la iteración de política modificada usa un paso de evaluación de política parcial (resolviendo un sistema de ecuaciones lineales) basado en la iteración de valor [Puterman, 2005].

Existe otro método para reducir el tiempo requerido por iteración que consiste en la identificación y **eliminación de acciones subóptimas** [Puterman, 2005]. Por ejemplo, las cotas de la función de valor óptima pueden ser usadas para eliminar a las acciones subóptimas. La ventaja de este método es que el conjunto de acciones es progresivamente reducido, con la consecuente reducción de tiempo [Wang, 2006].

Por último, el número de iteraciones puede ser reducido mediante el uso de criterios de paro de iteraciones basados en **cotas o umbrales ajustados al error de Bellman** (vea ecuación (2.6)), de índole inferior y superior. Por ejemplo, un criterio de paro podría ser detener la iteración de valor cuando el rango de este error se encuentre debajo del umbral de valor dado.

2.2.1.2. Iteración de Política

Otro algoritmo de programación dinámica es la iteración de política (vea Algoritmo 2.2). En este método, la política es repetidamente mejorada al encontrar una acción en cada estado que tenga un valor más alto que el valor de la acción escogida anteriormente para ese estado. La política inicial es aleatoria y el proceso termina cuando ya no es posible realizar más mejoras. Cabe señalar que este proceso garantiza la convergencia a la política óptima [Puterman, 2005] y se muestra a continuación.

La dinámica de este algoritmo es la siguiente: toma el MDP, un valor de descuento, un parámetro de convergencia y devuelve políticas óptimas de horizonte finito sucesivas, terminando cuando el máximo cambio entre la función del valor actual y la del valor previo es menor que el margen de error dado, ε.

Algoritmo 2. 2 Iteración de Política [Puterman, 1994].

(S, A, R, P, γ)

1: **para todo** $s \in S, \pi(s) =$ elementoAleatoriode (A)
2: **computar** $s \in S, U^o = 0$
3: **para todo** $s \in S$ **hacer**
4: encontrar una acción a tal que:
5: $[R(s,a) + \gamma \Sigma_{u \in S} P(a,s,u) U^{\pi}(u)] > U^{\pi}(s)$
6: **hacer** $\pi^*(s) = a$ **si esta acción existe;**
7: **en caso contrario hacer** $\pi^*(s) = \pi(s)$
8: **fin para**
9: **si** $\pi^*(s) \neq \pi(s)$ **para algún** $s \in S$ ir al paso 2.
10: **regresa** π

De esta manera logra la convergencia y se detiene.

Con este algoritmo es relativamente sencillo encontrar la política óptima en n pasos $\pi^*_(n)$, con la función de valor U^*_n, usando la siguiente relación de recurrencia [Puterman, 2005]:

$$\pi^*_n = argmax_a\{R(s,a) + \gamma \sum_{s',\epsilon,S} T(s,a,s') U^*_{n-1}(s')\} \tag{2.8}$$

Por último, es importante señalar que el algoritmo de iteración de política se encuentra en desventaja frente al algoritmo de iteración de valor, pues en este último las políticas óptimas son producidas por horizontes finitos sucesivamente más largos hasta que logra la convergencia. Además, la iteración de valor utiliza menos recursos computacionales que la iteración de política [Puterman, 1994].

2.2.2. Programación Lineal

Esta técnica resuelve un problema indeterminado formulado a través de ecuaciones lineales, para ello cuenta con el método *Simplex*. Este método fue desarrollado por G. B. Dantzig en 1947 y consiste en la optimización de la función objetivo al considerar las restricciones habidas [Puterman, 2005]. Estas restricciones son inecuaciones o desigualdades que acotan a la región que contiene a todos los valores factibles para dichas expresiones matemáticas. Con este método, la iteración toma el valor de un vértice e investiga si es su máximo valor, de no ser así repite el proceso en forma iterativa hasta encontrarlo.

Un problema de programación lineal es un caso especial de programación matemática, el cual trata de identificar un punto extremo (máximo o mínimo) de una función $f(x_1, \ldots, x_n)$, que además satisface un conjunto de restricciones

$g(x_1, \ldots, x_n)$.

La programación lineal especializa a la programación matemática en el caso donde la función $f(x)$ es llamada **función objetivo**, donde las restricciones del problemas son lineales. Se puede demostrar que si U satisface a $U(s) \geq [R(s,a) + \gamma T(s,a,s')U(s')]$, entonces U es límite superior para el valor óptimo, U^*.

Con base en este teorema se plantea el programa lineal *primal* para resolver un MDP descontado, el cual se expresa de la siguiente manera:

$$Minimizar \sum_{s' \epsilon S} U(s) \tag{2.9}$$

$$U(s) - \gamma \sum_{s' \epsilon S} T(s,a,s')U(s') - R(s,a) \geq 0 \tag{2.10}$$

donde los valores de $U(s)$ por cada estado son tratados como variables. Normalmente se prefiere la formulación dual sobre la primal, que tiene $|S|$ filas y $\sum_{s \epsilon S} |A_s|$ columnas (la primera tiene $|S|$ columnas y $\sum_{s \epsilon S} |A_s|$ filas). Entre los métodos eficientes de solución se destacan aquellos basados en el método *Simplex Coseno* [Trigos, 2005].

Por último, aún cuando la programación lineal tradicional se limitaba a la solución de MDP descontados con horizontes finitos, en la década pasada emergió esta área como una alternativa prometedora para la solución de MDP altamente dimensionales y con espacios continuos [Koller, 2000]. Al respecto, De Farías & Van Roy [De Farías, 2003] propusieron una técnica para aproximar programación dinámica con programación lineal para resolver problemas de control estocástico, reportando cierto éxito.

2.2.3. Aprendizaje por Refuerzo

En caso de que no se cuente con la función de transición de estados ni con la función recompensa del problema, el planificador deberá explorar su entorno para aprender qué hacer y cómo relacionar situaciones (estados) con acciones, de manera que la recompensa sea maximizada.

El aprendizaje por refuerzo está basado en la interacción constante de un agente con su entorno incierto, por lo que existe un **fuerte compromiso exploración-explotación**: la información de entrada es la retroalimentación que obtiene del mundo exterior como respuesta a sus acciones. De ahí este aprendizaje sea cercano a la teoría óptima de control y la aproximación estadística.

Mediante la función de aprendizaje por refuerzo, el agente determina cuáles estados son más provechosos a corto plazo. A diferencia de la técnica anterior, la función evaluación es hecha a largo plazo, considerando a los estados a los que puede conducir una acción. De esta manera, sus programas de cómputo buscan ir aprendiendo por sí solos conforme van adquiriendo experiencia, tal y como sucede con el ser humano. En consecuencia, los métodos de solución que utilizan aprendizaje por refuerzo son **inductivos**.

La idea central es hacer uso de la estructura inherente de los MDP, cuyos estados y acciones pueden ser abstraídos en subpolíticas [Van Otterlo, 2005]. Finalmente, una vez que explora y obtiene información de su entorno, este método se torna informado, pudiendo ser resuelto mediante los métodos de programación dinámica o programación lineal antes descritos.

2.3. Complejidad

Un aspecto muy importante a cuidar en todo algoritmo es su grado de complejidad. Es deseable que el algoritmo sea lo más simple posible, pues a mayor grado de complejidad, tendrá mayor dificultad para la ejecución del programa, especialmente frente a problemas altamente dimensionales. Al respecto, la **complejidad temporal** [Aho, 1988] de un programa, que también se le conoce como **tiempo de ejecución**, depende de factores tales como:

- los datos de entrada al programa,
- la calidad del código generado por el compilador utilizado para crear el programa objeto,
- la naturaleza y rapidez de las instrucciones de máquina empleadas en la ejecución del programa,
- la complejidad de tiempo del algoritmo base del programa.

El hecho de que el tiempo de ejecución dependa de la entrada indica que dicho tiempo debe definirse como una función de entrada y generalmente se refiere a su "tamaño". La medida natural del tamaño de la entrada a un programa es su longitud.

Por otro lado, se acostumbra denominar al tiempo de ejecución de un programa como $T(n)$, con una entrada de tamaño n. Las unidades de $T(n)$ se dejan sin especificar, pero se puede considerar a $T(n)$ como el número de instrucciones ejecutadas en un ordenador ideal.

Para muchos programas, el tiempo de ejecución es en realidad una función de la entrada específica y no sólo del tamaño de ella. En este caso se define $T(n)$ como el tiempo de ejecución del peor caso, es decir, el máximo valor del tiempo

de ejecución para entradas de tamaño n.

También suele considerarse $T_{prom}(n)$ el valor promedio del tiempo de ejecución de todas las entradas de tamaño n. Aunque $T_{prom}(n)$ parece una medida más razonable, a menudo es engañoso suponer que todas las entradas son igualmente probables.

En la práctica casi siempre es más difícil determinar el tiempo de ejecución promedio que el del peor caso, pues el análisis matemático se hace intratable y la noción de entrada "promedio" puede carecer de un significado claro. Así pues, se utiliza el tiempo de ejecución del peor caso como medida principal de la complejidad de tiempo. En cuanto al segundo y al tercer factor, el hecho de que el tiempo de ejecución dependa del compilador y del ordenador utilizados implica que no es posible expresar en unidades estándar de tiempo, como son los segundos. Sólo se pueden hacer observaciones tales como "el tiempo de ejecución del algoritmo es proporcional a n^2", sin especificar la constante de proporcionalidad, pues depende en gran medida del compilador, del ordenador y de otros factores. Por ejemplo, un programa con tiempo de ejecución $O(n^2)$ es mejor que uno con tiempo de ejecución $O(n^3)$ [Aho, 1988].

Por otro lado, la **complejidad computacional** se refiere a la velocidad de crecimiento de los valores de una función y se representa por la notación asintótica ("O grande"). Por ejemplo, decir que el tiempo de ejecución $T(n)$ de un programa es $O(n^2)$, que se lee "O grande de n al cuadrado", significa que existen c y n_o constantes enteras positivas tales que, para n mayor o igual que n_o, se tiene que $T(n) \leqslant cn^2$.

Cuando el tiempo de ejecución de un programa es $O(f(n))$, se dice que tiene una *velocidad de crecimiento* de $f(n)$. Entonces $f(n)$ es una cota superior para la velocidad de crecimiento de $T(n)$. A continuación se presenta una lista de tiempos de ejecución en orden ascendente de velocidad de crecimiento: $O(n)$, $O(n^2)$, $O(n^3)$, $O(c^n)$ [Aho, 1988].

Sin embargo, calcular el tiempo de ejecución de un programa arbitrario puede ser un problema matemático complejo. En la práctica esto suele ser más sencillo: la regla es que el tiempo de ejecución de una secuencia fija de pasos es igual al tiempo de ejecución del paso con mayor tiempo de ejecución, tanto para suma como para multiplicación de pasos [Aho, 1988].

Al respecto, de acuerdo con la literatura acerca de MDP, la complejidad temporal del algoritmo de iteración de valor está en función del número de iteraciones (n_{it}), del número de estados (n_s) y del número de acciones (n_a) y se calcula de la siguiente manera:

$$T\left(n_{it}, n_s, n_a\right) = \left(n_{it}\right)\left(n_s\right)\left(n_a\right) \tag{2.11}$$

Por otro lado $P(a, s, s')$, la función de transición de estados, es una matriz tridimensional con complejidad cuadrática en función del número de estados y complejidad lineal en función del número de acciones. Para problemas altamente dimensionales ($n_s \to \infty$), el número de acciones y el número de iteraciones son relativamente pequeños y constantes, por lo que resulta:

$$T\left(n_{it}, n_s, n_a\right) = n_s^2 \tag{2.12}$$

En resumen, los MDP exhiben una **complejidad de orden cuadrática** con respecto al tamaño del espacio de estados $|S|$ y de orden lineal con respecto al tamaño del espacio de acciones $|A|$ y con máxima precisión en *bits* para representar valores de recompensa de $logmax_{s,a}|T(s, a)|$.

Tanto en problemas de horizonte finito como de horizonte infinito descontado, el tiempo computacional requerido por iteración resulta de $O(|S|^2|A|)$ y de $O(|S|^2|A| + |S|^3)$, respectivamente y convergen en un número polinomial de iteraciones.

Cabe señalar que en esta tesis **la complejidad temporal se calcula en forma experimental** mediante la línea de tendencia de la gráfica del tiempo de solución en función del número de estados de cada algoritmo implementado.

Finalmente, es bien conocido que **la complejidad computacional** del algoritmo de iteración de valor puede ser calculada como **la complejidad de la actualización de un estado multiplicada por el número de actualizaciones requeridas para obtener la convergencia**. Como la complejidad de una actualización de estado es igual al número de acciones, entonces es posible hacer una comparación usando solamente el número de actualizaciones ejecutadas por cada algoritmo cuando converge a la política óptima. Por lo que, en esta tesis, **la complejidad computacional se calcula en forma experimental** mediante la línea de tendencia de la gráfica del número de actualizaciones en función del número de estados, de cada algoritmo implementado.

2.4. Estado del Arte en Técnicas de Solución de los Procesos de Decisión de Markov

A continuación se presenta una inspección del estado del arte en las técnicas de solución de MDP encaminadas a eliminar la indeseable intratabilidad que ha presentado este formalismo con el curso de la dimensionalidad.

2.4.1. En Programación Dinámica

La búsqueda heurística en los MDP es el proceso que se sigue para encontrar un plan (no necesariamente el óptimo) asignándole a cada estado una estima-

ción del coste óptimo a la solución. Los métodos basados en búsqueda heurística se propusieron con el objeto de minimizar el número de estados relevantes y el número de expansiones durante la búsqueda, pero el coste ha sido usualmente alto. Barto *et al.* [Barto, 1995] propusieron RTDP, un programa dinámico en tiempo real, donde una acción voraz es seleccionada basándose en el conocimiento que tiene el estado en ejecución. Como el estado cambia estocásticamente, entonces visita todos los estados posibles y retorna. Esta búsqueda termina cuando se ha calculado un cierto número de pruebas. Tiene la ventaja de encontrar políticas subóptimas rápidamente, pero su convergencia ha resultado lenta. Hansen *et al.* [Hansen, 2001] en LAO* consideraron solo una parte del espacio de estados para construir un grafo de soluciones parciales, hacen búsqueda implícita partiendo del estado inicial hacia el estado meta y expandiendo a la rama más prometedora del MDP, de acuerdo con una función heurística. Bhuma *et al.* [Bhuma, 2003] extiendieron al enfoque anterior usando el primer algoritmo de búsqueda heurística bidireccional.

Posteriormente Dai *et al.* [Dai, 2007a,b] hicieron una inspección acerca de técnicas heurísticas del estado del arte y compararon a los dos anteriores enfoques. Dai *et al.* [Dai, 2007a] propusieron extender la idea de Bhuma *et al.* [Bhuma, 2003], usando concurrentemente diferentes puntos de inicio para llegar al estado meta más rápidamente. Ellos combinaron estrategias de búsqueda heurística con programación dinámica para obtener la convergencia al plan óptimo. Sin embargo ellos reportan que su algoritmo fue solamente diez veces más rápido que el clásico algoritmo de iteración de valor debido al alto coste de inicio y al mantenimiento de cola de prioridad. También Bonet *et al.* [Bonet, 2003a,b] en LRTDP propusieron dos algoritmos heurísticos que usan una técnica de etiquetado inteligente para marcar estados irrelevantes.

Por otro lado, Peng *et al.* [Peng, 2007] propusieron un algoritmo de iteración de valor topológico, que retorna a estados basándose en secuencias estructurales, evitando retornos innecesarios a todo el espacio de estados. Este método presentó mejor desempeño comparado con LRTDP, LAO*, VI y HDP, detectando secuencias óptimas. Menciona que garantiza la convergencia en la función de valor óptima y que aborda perfectamente la complejidad resultante del incremento de capas estrucurales, aunque solamente fue probado en un ejemplo académico. Bonet *et al.* [Bonet, 2006] combinaron los beneficios de la programación dinámica con la potencia de la búsqueda heurística mediante el algoritmo *Learning Depth-First Search*, que ejecuta búsqueda iterativa "primero en profundidad" acompañada con aprendizaje. Reportan buenos resultados al ser comparados con el algoritmo de iteración de valor y con sus algoritmos previamente propuestos, debido a que la heurística lo hace óptimo. Sin embargo, no han tenido éxito con los grafos AND/OR aditivos. Además los algoritmos de iteración dinámica no han soportado el manejo de números no enteros, como pueden ser los costes reales.

Guestrin *et al.* [Guestrin, 2003] propusieron un enfoque de aproximación en

programación dinámica para MDP factorizados, cuyo modelo de transición es manejado por una red Bayesiana dinámica [Dean, 1993]. El elemento central de su propuesta es una técnica de descomposición lineal, que explota dos estructuras: la aditiva (que captura características del sistema global que pueden combinar a los componentes que interactúan localmente, por ejemplo en una producción en serie, la calidad que se obtenga en uno de los pasos afectará a la calidad de los demás) y la de contexto específico (que codifica a los diferentes tipos de influencia, por ejemplo siguiendo con la producción en serie, una pieza solo se trabajará en la celda que la pueda contener por su forma y volumen). Su algoritmo aplica una aproximación basada en *max-norm*, una técnica que minimiza directamente los términos del margen de error de la aproximación en MDP. Reportan haber trabajado con problemas del orden de 10^{40} estados y, comparándolo con los enfoques de Boutilier *et al.* [Boutilier, 2000] y de Koller *et al.* [Koller, 2000], ellos obtuvieron mejores tiempos de solución, aunque no lograron superar la escalabilidad de los problemas por la presencia de árboles con considerable anchura.

Hoey *et al.* [Hoey, 2005] propusieron un método para resolver POMDP continuos o de grandes dimensiones imponiendo una discretización *a priori* al espacio de observación, que puede ser subóptima para la toma de decisiones, mediante partición dinámica. Para su demostración ellos utilizaron un ejemplo de juguete y sobre una tarea de vida asistida (personas con capacidades disminuidas). Solamente abordaron estados discretos, pero no han logrado la integración total de la programación dinámica con la regresión basada en puntos. Spaan *et al.* [Spaan, 2004] propusieron la misma regresión pero, dada la naturaleza continua del espacio de estados, generaron α-funciones en vez de α-vectores, dándole mayor complejidad al proceso de resolución de los MDP. Givan *et al.* [Givan, 2003] introdujeron la medida de bisimulación estocástica a grupos de estados, modelo de transición y distribuciones de recompensa, tratando de minimizar los MDP. Reyes *et al.* [Reyes, 2006a] presentaron una aproximación para resolverlos, tanto para dominios continuos como híbridos, en un horizonte infinito mediante partición del espacio de estados basándose en la función recompensa, aunque solamente probaron en problemas relativamente pequeños.

Zhang *et al.* [Zhang, 2006] restringieron subconjuntos o estados de creencia (cuyos componentes son similares) para acercarse a la optimización debido a la buena reducción del espacio de búsqueda. Mencionan que estos subconjuntos son cerrados (ninguna acción puede llevar al agente a otro estado de creencia), que son suficientes (es decir, que su función de valor puede ser extendida por todo el estado de creencia) y que son mínimos (porque su iteración de valor necesita considerar al menos al mismo subconjunto para obtener la calidad de las funciones de valor), aunque la restricción le adicionó complejidad al proceso. Chenggang *et al.* [Chenggang, 2007] usaron diagramas de decisión de primer orden, que son grafos acíclicos *booleanos* y que usan operadores STRIPS probabilísticos al estilo de *ReBell* [Van Otterlo, 2004], prometiendo mayor flexibilidad a su representación. Sin embargo, ellos no pudieron utilizar la cuantificación uni-

versal en sus parámetros y funciones.

En otro sentido, la priorización aplicada a MDP se desarrolló con el objeto de decrecer la cantidad de actualizaciones ejecutadas, debido a que algunas de ellas son inútiles pues no todos los estados cambian entre dos iteraciones sucesivas. De esta manera se logra disminuir la mayor porción del tiempo consumido por la programación dinámica. Los métodos basados en priorización de estados usan la información del estado inicial para eliminar estados inalcanzables (e inútiles) y ejecutar las actualizaciones en orden inteligente en cada iteración. Al respecto, Moore *et al.* [Moore, 1993] propusieron al barrido priorizado, una técnica de programación dinámica que usa cola de prioridad, la cual es actualizada mediante un barrido a través del espacio de estados. Inicia insertando el estado meta en cola de prioridad (que es del tipo *offline* por ser externa), tal como lo hace iteración de valor [Puterman, 1994], de ahí que se considera una variante de este último. A cada paso, barrido priorizado arroja de cola al estado con más alta prioridad y ejecuta una actualización de la función de valor de ese estado. Sin embargo, el uso de colas de prioridad para todos los estados ha resultado en un excesivo tiempo de inicio requerido por la propia estructura.

También se encontró a Ferguson *et al.* [Ferguson, 2004], que propusieron la programación dinámica enfocada, que se reporta lenta debido a que tiene que ejecutar varias actualizaciones, pues considera a todos los predecesores de cada estado, no simplemente a los predecesores correspondientes al grafo de políticas (acciones). Por otro lado, los enfoques de ordenamiento topológico han sido capaces de encontrar buenos ordenamientos, pero su coste de inicio también ha sido considerablemente alto. Dai *et al.* [Dai, 2007b] propusieron a iteración de valor topológico, que agrupa estados que están mutua y causalmente relacionados en un *metaestado*, pero que está restringido a estados fuertemente conectados (acíclicos), no para MDP (que son grafos cíclicos). Más adelante ellos mismos presentaron un método de priorización que no requiere de cola de prioridad [Dai, 2007c], mediante una estrategia FIFO (primero en entrar, primero en salir, por sus siglas en inglés), al utilizar una búsqueda del tipo "primero en anchura" sobre la matriz transpuesta del grafo correspondiente al MDP. Este método se reporta rápido debido a que sólo considera a los estados predecesores en el grafo de políticas. Sin embargo, solamente es aplicable para MDP acíclicos que no corresponden al mundo real, por lo que ellos lo reportan subóptimo. Posteriormente Wingate *et al.* [Wingate, 2005] presentaron un ordenamiento topológico que encuentra buenos ordenamientos de actualizaciones, aunque con alto coste computacional.

Por otro lado, Jamali *et al.* [Jamali, 2006] consideraron un caso especial de problemas de ruta estocástica más corta, en el que el coste de las aristas (caminos entre nodos) tiene una distribución probabilista, de donde resulta un grafo estocástico ponderado. Ellos trataron de aprender el coste esperado para cada arista, usando el algoritmo de Dijkstra para encontrar la ruta más corta mediante conocimiento local. Dibangoye *et al.* [Dibangoye, 2008] con su iteración de

valor topológico mejorado propusieron una nueva manera de ordenar actualizaciones sin utilizar al algoritmo de Dijkstra, basándose en un mapeo del espacio de estados, balanceando el número de actualizaciones ejecutadas y el esfuerzo requerido para priorizar estas actualizaciones, mostrando un mejoramiento del orden de magnitud en el tiempo de solución sobre el número de parámetros de referencia. Sin embargo, ellos reportan que el orden de actualizaciones provisto no resulta óptimo, debido a que no minimiza su número. Otro enfoque es el de Shani *et al.* [Shani, 2008] que extiende el uso de priorización en MDP parcialmente observables.

Por último, McMahan *et al.* [McMahan, 2005a,b] mejoraron al barrido priorizado usando una extensión del clásico algoritmo de Dijkstra (basado en colas de prioridad) para resolver problemas de MDP deterministas, reportando una rapidez considerable. Ellos aplican priorización en aquellos estados cuya función de valor cambió entre dos iteraciones sucesivas. En consecuencia su algoritmo calcula sobre un reducido conjunto de estados, a los que acomoda en orden decreciente de máxima recompensa. Sin embargo, su método no ha logrado la convergencia a la política óptima en MDP no deterministas [Li, 2009].

2.4.2. En Programación Lineal

En esta técnica se encontró el enfoque de Paschalidis *et al.* [Paschalidis, 2000], quienes aplicaron su método a problemas bidimensionales, presentando entornos de error a la iteración de valor. También Guestrin *et al.* [Guestrin, 2002], Schuurmans *et al.* [Schuurmans, 2002] y Morrison *et al.* [Morrison, 1999], desarrollaron eficientes implementaciones del algoritmo *Simplex* para MDP factorizados, permitiendo que las restricciones en el programa lineal sean representadas compactamente. En este sentido, Zhang *et al.* [Zhang, 1996] habían fragmentado al problema en una serie de subconjuntos para simplificar su cálculo, reportando ser eficiente en el tiempo para problemas estructurados, aunque no lograron automatizarlo. En esta línea, Koller *et al.* [Koller, 2000] propusieron un nuevo enfoque para la determinación de la función de valor mediante una aproximación lineal de descomposición de mínimos cuadrados, donde el algoritmo de determinación de valor es una subrutina del proceso de iteración de política. Usaron un algoritmo de eliminación de variables para la función de optimización. La complejidad de sus algoritmos dependió de la factorización del sistema dinámico y de la aproximación de su función de valor.

Por otro lado, Guestrin *et al.* [Guestrin, 2001] propusieron la aplicación de multiagentes de planificación en los MDP factorizados, mediante una arquitectura de la función de valor y programación lineal, utilizando restricciones *a priori* en la estructura de comunicación entre agentes. Reportan que se les presentó una complejidad computacional dependiente de la anchura del árbol inducido de la coordinación del grafo utilizado por los agentes al negociar su selección de acción. Después, Hauskrecht *et al.* [Hauskrecht, 2004] presentaron

un modelo jerárquico con macroacciones o *macros*, que son políticas locales, ofreciendo soluciones de aproximación de programación lineal a los MDP con dominios continuos o híbridos. Propusieron un modelo de transición factorizado identificando funciones características y sus densidades, que tiene un enfoque de muestreo aleatorio de restricciones. Argumentaron que resulta una aproximación robusta para problemas con dominios altamente dimensionales (resuelve tres ejemplos), comparándola con algoritmos de rejilla. Además, evitan la inestabilidad del *método del mínimo cuadrado* [Koller, 2000]. Comentan que tiene la propiedad de reutilizar sus macros, dándole mayor flexibilidad al sistema. No obstante, mencionan que su método tiene dos limitaciones: *i*) la calidad de la solución, pues no siempre es la política óptima debido al ensamble de las *macros*, donde se pierden ciertos comportamientos, *ii*) que debido a que las regiones abstractas (correspondientes a las *macros*) cubren al espacio de estados y deberían de capturar y controlar una variedad de comportamientos, por lo que su generación mediante valores periféricos hace más compleja la solución del MDP original.

Después De Farías *et al.* [De Farías, 2003] le dieron un enfoque de programación lineal a la programación dinámica para resolver problemas de control estocáticos a gran escala. Propusieron un método que combina las funciones de peso relevante a los estados (lineales) con la función de valor (dinámica), aliviando de esta manera el curso de la dimensionalidad mediante el beneficio de la simplicidad inherente a la programación lineal, aunque reporta un error por su uso. Por otro lado, Feng *et al.* [Feng, 2004] trataron de combinar a la programación lineal con la programación dinámica en la solución de MDP continuos y estructurados, haciendo que a cada paso de la programación dinámica, el espacio sea partido en regiones rectangulares con funciones de valor constantes. Para cada acción, el conjunto de salidas y la distribución de probabilidad son idénticos para todos los estados dentro de la misma región segmentada, denominando cada segmento como "estado abstracto". Utilizaron árboles de Feldman *et al.* [Feldman, 1977] para almacenar y manipular las particiones que, una vez obtenidas las representaciones constantes finitas, se transforman en representaciones finitas lineales usando técnicas de POMDP. Estas representaciones sirven para razonar eficientemente en superficies lineales, mediante la suma cruzada con el modelo de recompensa. Mencionan que este enfoque explota a la estructura natural llegando a soluciones óptimas. Ellos reportan que, a pesar de que esta técnica no garantiza una reducción exponencial de la complejidad al discretizar a los problemas en dominios continuos, pudieron resolver en pocos minutos problemas de simulación del dominio *rover* planetario, que antes requerían de un día de procesamiento computacional. Dicen que una línea a seguir con su técnica combinada es utilizar acciones con parámetros continuos. Como trabajo relacionado mencionan a Munos *et al.* [Munos, 2002], quienes propusieron un modelo formal de MDP continuos y algoritmos para discretizarlos en forma adaptiva. Sin embargo, su modelo es un MDP determinístico, donde sus acciones pueden tener duraciones continuas.

En contraste, Guestrin *et al.* [Guestrin, 2003] utilizaron la aproximación de programación lineal, mediante una técnica de descomposición lineal, explotando las propiedades aditivas y de contexto específico en los estados abstractos. Su algoritmo resultó más simple y más fácil de implementar que la mayoría de las aproximaciones de programación dinámica. Desafortunadamente, ellos reportan el manejo de problemas cuyas acciones solamente afectan a pocas variables de estados, para los que obtuvieron tiempo de solución polinomial cercano al valor óptimo. Más adelante, ellos mismos [Guestrin, 2004] hicieron una reflexión acerca de los problemas del mundo real, que se acercan más a los modelos híbridos, pues manejan variables tanto discretas como continuas. Mencionan que pueden resolverlos mediante representaciones factorizadas que usan redes Bayesianas dinámicas [Dean, 1993], del tipo híbridas. Para abordar a estas redes, propusieron un método de aproximación de programación lineal, al cual denominaron HALP, que permite calcular funciones de valor aproximadas más eficientemente, con un enfoque hacia los sistemas multiagente. Esto lo logran haciendo una discretización factorizada de las variables continuas, evitando con esto el crecimiento cuadrático con el número de estados de los enfoques tradicionales. Ellos demostraron la eficiencia de su método en problemas de control con 28 dimensiones de espacio de estados y 22 dimensiones de espacio de acciones. Con la misma aproximación HALP, Kveton *et al.* [Kveton, 2006] presentaron un método para abordar dominios híbridos, que permite una representación compacta de los MDP continuos y factorizados. Ellos trataron de aproximar la función de valor óptima mediante una combinación lineal de funciones base y optimizar sus pesos mediante programación lineal. De esta manera ellos lograron que su espacio de restricciones tuviera la misma estructura que la aproximación de iteración de política de Guestrin *et al.* [Guestrin, 2001], con esto trataron de demostrar su potencial escalamiento sobre problemas, aunque su aprendizaje automático se tornó crítico ante problemas complejos.

2.4.3. En Aprendizaje por Refuerzo

En esta técnica se encontró el enfoque de Barto *et al.* [Barto, 2003], que utiliza una estructura de jerarquía de tareas para aprender subpolíticas en su sistema SMDP, aplicando duraciones a las tareas. Otros enfoques de abstracción jerárquica son MAXQ de Dietterich *et al.* [Dietterich, 2000] y OPTIONS de Sutton *et al.* [Sutton, 1999]. Sin embargo, estos métodos construyen su jerarquía de tareas manualmente. Sallans *et al.* [Sallans, 2002] y Guestrin *et al.* [Guestrin, 2003] hacen representaciones factorizadas pero no han logrado la automatización. Por otro lado, Guestrin *et al.* [Guestrin, 2003] probaron la representación de funciones de valor usando el marco del modelo relacional probabilista de Getoor *et al.* [Getoor, 2001], cuyo punto crucial es que considera estáticas las relaciones entre objetos, lo cual no siempre resulta ser cierto en el mundo real. Por otro lado Roncagliolo *et al.* [Roncagliolo, 2004] usaron el aprendizaje por lotes en una jerarquía relacional, cuyo resultado es una lista de decisiones sobre las tareas y subtareas a realizar. Se tiene también la representación predictiva de estado de

Littman *et al.* [Littman, 2002], cuyos estados están en términos de predicciones sobre el futuro. Ryan *et al.* [Ryan, 2004] construyeron tareas jerárquicas para moverse de un estado abstracto a otro, aplicando el comportamiento aprendido durante su exploración. Dzeroski *et al.* [Dzeroski, 2001] combinaron aprendizaje cualitativo con regresión relacional, usando un árbol-Q [Quinlan, 1993] y lógica de primer orden. Lecoeuche *et al.* [Lecoeuche, 2001] manejan listas de decisión de primer orden en lugar de árboles. Sin embargo, los dos últimos sistemas trabajan de manera semiautomatizada. Driessens *et al.* [Driessens, 2001] propusieron un sistema de aprendizaje totalmente incremental, TG, que construye árboles, pero presentó problema al obtener la medida de la muestra mínima para dividir un nodo, pues si es considerablemente grande entonces es lenta su convergencia. Luego Driessens *et al.* [Driessens, 2003] propusieron el método de regresión basado en instancias relacionales, RIB, que usa aprendizaje de estados en crecimiento, con el concepto de mínima distancia entre dos interpretaciones. Después Driessens *et al.* [Driessens, 2005] combinaron las fortalezas de TG y de RIB, resultando TRENDI, que construye un árbol pero con representación basada en instancias, con cierta mejoría en su desempeño.

En otro sentido, Sanner *et al.* [Sanner, 2005] presentaron un aprendizaje libre de modelo en dominios con horizonte finito, discontinuo, que proporciona una recompensa terminal simple en cada éxito o falla. Sin embargo solamente fue probado en algunos juegos mediante codificación manual. Fern *et al.* [Fern, 2003] se enfocaron en la inducción de políticas en dominios estocásticos, explorando al conjunto de instancias aleatorias del dominio. Yoon *et al.* [Yoon, 2005] midieron el progreso habido en el aprendizaje heurístico sobre dominios de planificación, tanto estocásticos y como deterministas. Kersting *et al.* [Kersting, 2003] introdujeron un lenguaje tipo STRIPS probabilista de primer orden en dominios relacionales, a través de reglas tipo: si <estado(i)> entonces <acción(j)>, resultando para un mismo estado varias acciones posibles. Sin embargo obtuvieron una significativa complejidad computacional. Mausam *et al.* [Mausam, 2003] calcularon situaciones para MDP relacionales de una forma más simple que Boutilier *et al.* [Boutilier, 1999], mediante técnicas de regresión de primer orden para aprender la función de valor del MDP relacional completo, aunque tuvieron problemas con la escalabidad y de convergencia. En este sentido, Morales [Morales, 2003] usó la representación relacional en el aprendizaje por refuerzo en robótica con cierto éxito. Sin embargo reporta haber obtenido políticas subóptimas.

Por otro lado, Van Otterlo *et al.* [Van Otterlo, 2005] definieron a CARCASS, que usa pares de estados-acciones, ambos abstractos, que aprende sobre un modelo de MDP relacional y que puede llevar a una extensión priorizada para resolverlo. Su abstracción consiste en un listado de reglas, sólo que deben ser codificadas manualmente. Anteriormente ellos habían propuesto la iteración de valor relacional, *ReBell* [Van Otterlo, 2004]. Sin embargo no lograron la convergencia. Feng *et al.* [Feng, 2004] usaron estados continuos mediante aprendizaje por refuerzo basado en explicaciones, siguiendo a [Dietterich, 1997]. Kochender-

fer *et al.* [Kochenderfer, 2005] integraron aprendizaje incremental del modelo y construcción de un plan reactivo para tiempo real, mediante un grafo de decisión donde el agente gana experiencia y refina su plan, aunque solo usaron dominios cuyo estado en ejecución es parcialmente observable. Por otro lado, Sanner *et al.* [Sanner, 2005] usaron funciones base que generan restricciones y una función de valor global. Sin embargo, como las restricciones las resuelve con programación lineal, entonces acumula error debido a las aproximaciones, por lo que no han podido automatizarlo. Gretton *et al.* [Gretton, 2004] usaron regresión basada en teoría en decisiones [Boutilier, 1999], calculándola parcialmente, con aprendizaje de árboles inductivos.

En otro sentido, Jonsson *et al.* [Jonsson, 2006] presentaron el método de análisis de estructura de influencia variable, VISA, el cual ejecuta descomposición jerárquica de MDP, usando una red Bayesiana dinámica [Dean, 1993] para modelar sus probabilidades de transición de estados debida a las acciones. Construye un grafo causal acíclico que captura las relaciones habidas entre las variables de estado, obteniendo una jerarquía de acciones llamada macroacciones. Lo comparan con SPUDD de Hoey *et al.* [Hoey, 1999], aunque este no requiere aprender el modelo. Sin embargo, VISA está limitado a trabajar con tareas que involucran relaciones entre variables de estado. Mannor *et al.* [Mannor, 2004] usan teoría de grafos para construir opciones automáticamente en un ambiente dinámico. Construyen un mapa topológico del ambiente donde se efectúan las transiciones de estado y usan agrupación para segmentar a dicho mapa en regiones. Las políticas aprendidas se guardan como macroacciones y se alimentan a un acelerador de aprendizaje con opciones [McGovern, 1997], etiquetando "regiones interesantes". Reportan mayor velocidad en la etapa de aprendizaje, aunque hicieron experimentos en dominios de poca dimensionalidad. Wierstra *et al.* [Wierstra, 2004] presentaron una modificación del algoritmo de *Baum-Welch* que maneja los modelos ocultos de Markov para crear memoria de planificación a partir de la percepción y utilidad en problemas no deterministas. Sin embargo ellos reportan intratabilidad con la escalabilidad.

En contraste, Pasula *et al.* [Pasula, 2004] suponen que las acciones solamente afectarán a un pequeño número de propiedades del mundo, por lo que hacen aprendizaje supervisado del conjunto de reglas resultantes y que el número de salidas de una acción es pequeño, lo cual no se apega a la realidad. Ellos mismos [Pasula, 2005] extendieron su enfoque para manejar efectos indeseables, etiquetándolos e ignorándolos como salida ruidosa, por lo que trabaja con estructuras parciales. Degris *et al.* [Degris, 2006] propusieron un marco de prueba y error con el entorno, adicionándole técnicas de aprendizaje supervisado. Usan un árbol de decisión inductivo e incremental, combinándolo con iteración de valor estructurado [Boutilier, 2000], aunque no lograron la política óptima. Driessens *et al.* [Driessens, 2005] propusieron un sistema que utiliza aprendizaje cualitativo y prueban tres algoritmos de regresión relacional, capaces de hacer predicciones y sugerencias después de haber explorado a sus vecinos más próximos. No obstante sólo reportan pruebas en ejemplos de juguete y juegos

sencillos de cómputo *Digger* y *Tetris*. Rafols *et al.* [Rafols, 2005] utilizaron representaciones predictivas acerca de observaciones futuras y las aplicaron en redes de Littman *et al.* [Littman, 2002], pero no pudieron involucrar recompensas en estados y codificadron manualmente el mapeo de clases, resultando por ello ineficiente.

Por otro lado, Epshteyn *et al.* [Epshteyn, 2006] consideraron un marco con restricciones de dominios estocásticos para el clásico problema de ascenso de un carro en una montaña. Kersting *et al.* [Kersting, 2006] desarrollaron un marco general para aprendizaje relacional estadístico, incorporando conceptos lógicos de objetos y sus relaciones a través de redes bayesianas. Jong *et al.* [Jong, 2006] combinaron las fortalezas del aprendizaje por refuerzo basado con la representación de estados basada en instancias en el dominio *Mountain Car*. Reportan que empíricamente convergen en buenas políticas sobre dominios continuos, sólo que con poca cantidad de datos.

En contraste, Kalyanakrishnan *et al.* [Kalyanakrishnan, 2007] compararon aprendizaje por refuerzo con algoritmos de tiempo real basados en un dominio multiagente, ruidoso y continuo. Aplicaron reutilización de la experiencia e iteración compuesta. Trabajaron en lotes, reportando cierta economía computacional comparando con algoritmos de tiempo real. Safaei *et al.* [Safaei, 2007] presentan inducción incremental utilizando operadores de planificación probabilistas para actuar en dominios estocásticos, aunque no demuestran estabilidad y exactitud. Por último, Fernandez *et al.* [Fernandez, 2008] propusieron un método de aproximación de funciones de valor basadas en aprendizaje supervisado (algoritmo CMAC Q-Learning), basado en el controlador robótico CMAC, propuesto en 1975 por James Albus.

2.5. Conclusiones del Capítulo

En este capítulo se estudió a los MDP y se hizo una inspección en sus técnicas de solución. Al respecto las hay informadas y no informadas. El aprendizaje por refuerzo es una técnica no informada debido a que carece de información de su entorno inicialmente, por lo que una vez que el agente lo ha explorado se transforma en técnica informada, con la complejidad adicional que este procedimiento le reporta. En cuanto a las técnicas informadas para resolver MDP, se encontró que el algoritmo de iteración de política (de programación dinámica) y programación lineal han resultado computacionalmente más costosas que el algoritmo de iteración de valor. Esto es debido a que aquellas técnicas requieren resolver, en cada iteración, un sistema de ecuaciones lineales de la misma medida que el espacio de estados. En contraste, el algoritmo de iteración de valor evita este problema usando un enfoque recursivo de programación dinámica. Sin embargo, dado que el requerimiento típico del algoritmo de iteración de valor es **enumerar enteramente el espacio de estados y de acciones**, entonces ha

resultado impráctico, ineficiente e intratable ante la escalabilidad de problemas. Por lo que la solución de MDP complejos ha sido una interesante área de investigación por largo tiempo. Por tales motivos, en el siguiente capítulo se estudiarán dos enfoques del estado del arte que prometen reducir el espacio de búsqueda de los MDP, con el objeto aplicar su filosofía en el algoritmo de iteración de valor.

Capítulo 3

Reducción del Espacio de Búsqueda

Como se señaló en el capítulo anterior, la limitación que han presentado los procesos de decisión de Markov (MDP por sus siglas en inglés), al intentar resolver problemas de considerables dimensiones, ha sido la intratabilidad. Con el objeto de zanjar esta limitación, se han propuesto técnicas de reducción del espacio de búsqueda, entre las que se encuentran las basadas en minería de datos, en priorización y en búsqueda heurística. De las anteriores técnicas se seleccionaron dos estrategias que han sobresalido por su eficiencia: minería de datos y priorización, las cuales entregan respectivamente:

- una reglamentación de acciones y,
- una priorización de estados.

En el siguiente capítulo se probará aplicar la filosofía de estas estrategias en el algoritmo de iteración de valor, con el objeto de darle tratabilidad ante la escalabilidad. A continuación se describen estas estrategias.

3.1. Reglamentación de Acciones

Del análisis del estado del arte se desprende que es posible calcular acciones en función del estado en evaluación mediante un método de minería de datos: el algoritmo *Apriori* [Agrawal, 1993, 2002]. Con este método se puede reducir el **espacio de búsqueda de acciones**. Para ello es necesario que el autómata recolecte datos durante la exploración de su entorno y con esta estrategia obtenga las reglas de asociación. Estas reglas relacionan un estado inicial con una acción requerida para alcanzar un estado final. A continuación se describen estas reglas y el algoritmo que las obtiene.

3.1.1. Reglas de Asociación

Considere un ejemplo sencillo, representado en el *modelo de Markov* que se presenta en la Figura 3.1, donde existen tres acciones probables $(a1, a2, a3)$ y cuatro estados $(s1, s2, s3, s4)$.

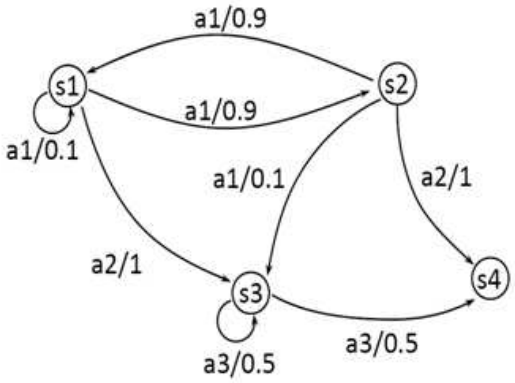

Figura 3.1: Un modelo de Markov.

En esta figura se puede observar que, al ejecutar la acción $a1$ sobre $s1$, existe el 90 % de probabilidad de alcanzar el estado $s2$, así como el 10 % de probabilidad de permanecer en el estado $s1$. Siguiendo este diagrama es fácil obtener las acciones que son ejecutables en cada estado (vea Tabla 3.1).

estado	*acciones*
$s1$	$a1, a2$
$s2$	$a1, a2$
$s3$	$a3$

Tabla 3.1: Conjunto de acciones ejecutables en cada estado, correspondiente al modelo de Markov de la Figura 3.1.

Con las reglas de asociación se pueden describir relaciones entre atributos, si los estados y las acciones se describen como atributos. Estas reglas son del tipo: *Si un autómata se encuentra en el estado(i) y ejecuta la acción(j), entonces es probable que alcance el estado(k).*

Una forma sencilla de visualizar estas reglas es mediante un árbol de decisión, en el que cada rama corresponde a una regla de asociación. Por ejemplo, en la Figura 3.2 se puede apreciar un árbol de decisión derivado del problema *Jugar Golf* [Quinlan, 1993], de acuerdo con la observación del estado del tiempo.

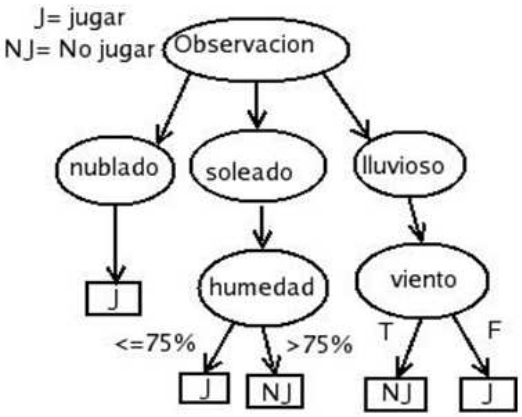

Figura 3.2: Un árbol de decisión.

Si la observación contiene a "lluvioso Y viento", entonces la acción que hace verdadera a esta expresión debe ser "NJ" y se establece como:

Si observación = (lluvioso Y viento) = T entonces Clase = No Jugar

Cada rama del árbol contiene a la acción (hoja) que corresponde a ciertos atributos (estados o nodos) y a ciertas relaciones habidas entre los atributos en una base de datos.

Cabe destacar que **una regla de asociación corresponde a la representación compacta de la matriz tridimensional de la función de transición de estados, debido a que solamente considera a las relaciones estado-acción con probabilidad diferente de cero** [Garcia-Hernandez, 2009]. De ahí que el conjunto de reglas de asociación resulte ser una **lista de adyacencia**.

Las reglas de asociación representan una importante herramienta en aplicaciones de minería de datos. En la regla: si $X \Rightarrow Y$, X e Y son conjuntos disjuntos de atributos (*itemsets*), X e Y se refieren a pares atributo-valor, no solamente a atributos.

En este sentido, una típica aplicación de minería de datos es descubrir asociaciones entre artículos. Por ejemplo, en el análisis del contenido de una canasta de supermercado [Hashler, 2005]. Estas asociaciones pueden ofrecer información útil para conocer detalles de usos y costumbres y tomar óptimas decisiones en el manejo de colecciones de artículos [Brijs, 2004], recomendaciones personalizadas de artículos [Lawrence, 2001] y para implementar actividades promocionales [Van den Poel, 2004].

Dado que cada regla de asociación consiste de una acción condicionada al atributo "estado inicial" (que es el estado en evaluación), en esta tesis se concluyó que **si el MDP calcula únicamente sobre reglas de asociación (es decir, solamente sobre las acciones ejecutables en el estado en evaluación), entonces requerirá de menor esfuerzo computacional dado que no tendrá que calcular sobre todas las acciones del dominio**. A continuación se describe al algoritmo que es capaz de calcular estas reglas.

3.1.2. Algoritmo Apriori

Las reglas de asociación usualmente son seleccionadas desde el conjunto de todas las posibles reglas, usando medidas de significancia estadística y de interés, como lo hace el algoritmo Apriori [Agrawal, 1993]. Este algoritmo parte de una serie de transacciones observadas para elaborar las reglas que asocian a los atributos participantes. Estas reglas se basan en medidas de confianza y soporte de cualquier conjunto de atributos relacionado con otros conjuntos de atributos. Por lo que este algoritmo es capaz de encontrar las reglas que predicen la ocurrencia de un atributo basándose en la ocurrencia de otros atributos en el registro [Agrawal, 1993].

Dado un conjunto de instancias o transacciones, Apriori trata de encontrar lo que tienen en común, generando candidatos en forma de árbol de decisión, sobre el que hace "búsqueda en anchura". Posteriormente somete a poda a dicho árbol para obtener los conjuntos que presentan mayor frecuencia. A menudo las bases de datos contienen miles o incluso millones de registros. Por lo que, para seleccionar reglas interesantes del conjunto de todas las reglas que se pueden derivar de un conjunto de datos, se utilizan restricciones en la medida del soporte y de la confianza (vea ecuaciones (3.1) y (3.2)).

El **soporte** es una medida de significancia estadística y es definido como el porcentaje de instancias que contienen a todos los *items* o atributos de una regla. Esto es, para un conjunto de *items* X e Y en una base de datos D, se define como la proporción de transacciones en la base de datos que contiene a dicho conjunto de atributos:

$$sop(X \Rightarrow Y) = sop(X \smile Y) = \frac{|X \smile Y|}{|D|} \tag{3.1}$$

donde $|X \smile Y|$ representa el número de instancias que contienen todos los *items* en X e Y, y $|D|$ es el número de instancias contenidas en una base de datos. En consecuencia, se utiliza un umbral de soporte mínimo para seleccionar a los *itemsets* más frecuentes (y por lo tanto más prometedores) [Hashler, 2005].

Por otro lado, la **confianza** es una medida de interés y representa el máximo

porcentaje de instancias cubiertas por una regla, la cual se define como:

$$conf(X \Rightarrow Y) = \frac{sop|X \smile Y|}{sop|X|} = P(Y|X) \tag{3.2}$$

donde $sop|X|$ es el porcentaje de instancias en la base de datos que contienen solamente al *item* X, y $P(Y|X)$ representa la probabilidad de que se presente Y habiéndose presentado X.

Por lo anteriormente expuesto, en esta tesis se concluyó que **la confianza de una relación o regla de asociación viene a ser un estimador de la probabilidad de transición entre dos estados, dada una acción**. Por esta razón se decidió utilizar al algoritmo Apriori para obtener aquellas reglas cuya probabilidad sea diferente de cero.

Lo anterior es sumamente importante, pues el algoritmo de iteración de valor solamente tendrá que calcular sobre estas reglas, es decir **sobre las acciones que realmente son ejecutables en el estado en evaluación, no sobre todas las acciones del dominio**. En consecuencia, el algoritmo de iteración de valor resolverá más rápidamente el MDP.

Por otro lado, las reglas de asociación deben satisfacer las especificaciones del usuario en cuanto a umbrales mínimos de confianza. Para conseguir esto el proceso de generación de reglas de asociación se realiza en dos pasos. Primero, se aplica el soporte mínimo (*minSup*) para encontrar a los conjuntos de atributos más frecuentes en la base de datos. Segundo, se forman las reglas partiendo de estos conjuntos frecuentes de atributos y de la restricción de confianza mínima (*minConf*). A continuación se detallan ambos pasos:

Paso 1

- generar todos los conjuntos (*itemsets*) con un solo elemento y con ellos generar conjuntos con dos elementos y así sucesivamente.
- tomar todos los posibles pares cuyo soporte sea mayor que *minSup* (permite ir eliminando posibles combinaciones considerando todas las combinaciones).

Paso 2 Por cada *itemset* frecuente L' encontrado y por cada uno de sus subconjuntos:

- determinar todas las reglas de asociación,
- seleccionar aquellas reglas cuya confianza sea mayor que *minConf*.

De esta manera se forma una matriz con la que se calcula el soporte y la confianza de las reglas de asociación. Una observación es que si la unión de conjuntos de una regla cumple al menos con los niveles requeridos de soporte y

confianza, entonces sus subconjuntos también los cumplirán. Por el contrario, si algún atributo no los cumple, no tiene caso considerar a sus superconjuntos [Agrawal, 1993].

En resumen, este algoritmo obtiene las reglas de asociación de la siguiente manera:

- busca las posibles combinaciones de todos los atributos,
- usa esas combinaciones para generar reglas de asociación,
- selecciona las reglas que ofrezcan el soporte y la confianza mínimos,
- entrega las mejores reglas encontradas (es decir, con máximos soporte y confidencia).

Como se puede observar en el Algoritmo 3.1, Apriori inicia contando el número de veces en que se presenta un *item*, con el objetivo de encontrar los conjuntos de *items* L_1 más frecuentes. Los siguientes k pasos consisten en dos fases.

En la primera fase, los conjuntos de *items* frecuentes $L-k-1$ encontrados en el paso previo son usados para generar a los conjuntos de *items* candidatos C_k. Después se revisa la base de datos y se calcula el soporte de los candidatos en C_k.

En la segunda fase, el conjunto de *items* candidatos es sujeto a un proceso de poda para garantizar que todos los subconjuntos de los conjuntos candidatos sean siempre frecuentes.

Al respecto, el proceso de generación de candidatos se ejecuta de la siguiente manera: si un conjunto de *items* X tiene mínimo soporte, entonces lo segmenta en todos sus subconjuntos posibles. El paso de podado elimina las extensiones de los conjuntos de $(k-1)$-*items* que no fueron encontrados como frecuentes. Así continúa este algoritmo hasta que no queda algún candidato después de la poda [Agrawal, 1993, 2002].

Dado un conjunto de instancias o transacciones T y un error de aproximación ε, se tendrán L_k combinaciones de atributos o *itemsets* conteniendo k atributos y C_k combinaciones generadas con L_k reglas de asociación.

El esfuerzo computacional depende principalmente de la confianza mínima requerida y se lleva prácticamente todo el recurso computacional en el primer paso. Al proceso de iteración del primer paso se le denomina "de nivel discreto" *level wise*, esto significa que va considerando a los superconjuntos nivel por nivel. Se le aplica la *propiedad antimonótona*: si un conjunto de atributos no pasa la prueba de soporte, ninguno de sus superconjuntos la pasarán. Esto se aprovecha en la construcción de candidatos de tal manera que no se consideren a todos ellos, sino más bien haciendo un barrido por la base de datos para cada conjunto de atributos de diferente tamaño [Agrawal, 1993, 2002].

Algoritmo 3. 1 Apriori [Agrawal, 1993]

(T, ε)
$L_1 \leftarrow \{1 - itemsets\}$
$k \leftarrow 2$
mientras $L_{k-1} \neq \emptyset$ **hacer**
  $C_k \leftarrow genera(L_{k-1})$
  para transacciones $t \in T$
  $C_t \leftarrow C_{k,t}$
fin mientras
para candidatos $c \in C_t$ **hacer**
  $cont\,[c] \leftarrow cont\,[c] + 1$
  $L_k \leftarrow \{\, c \in C_k | \, cont\,[c] \geq \varepsilon\}$
  $k \leftarrow k + 1$
fin para
regresa $\cup_k L_k$

Transacción	*Leche*	*Pan*	*Mantequilla*	*Cerveza*
1	1	1	0	0
2	0	1	1	0
3	0	0	0	1
4	1	1	1	0
5	0	1	0	0

Tabla 3.2: Matriz de *items* de cada transacción.

Cabe destacar que **Apriori presenta la ventaja** de ser de fácil implementación y de fácil paralelización con otros procesos. De ahí el interés en este algoritmo para aplicarlo en el enfoque que se propone en esta tesis.

Por último, se tienen algunas versiones mejoradas de este algoritmo, tales como *AprioriHybrid* [Agrawal, 1994] que ha mostrado escalar linealmente con el número de instancias para casos específicos y *FP-growth* [Gupta, 2006], que ha reportado cierta eficiencia.

Para ilustrar estos conceptos, se presenta el siguiente ejemplo de ventas en un supermercado. El conjunto de *items* es: $I = \{Leche,\ Pan,\ Mantequilla,\ Cerveza\}$.

En la anterior tabla se muestra una pequeña base de datos, donde el dígito 1 (uno) se interpreta como que el producto (que es un *item*) está presente en la transacción y en su defecto se usa el dígito 0 (cero). Una regla de asociación para este ejemplo podría ser:

$$\{Leche, Pan\} \rightarrow \{Mantequilla\}$$

lo cual significa que si el cliente compra $\{Leche$ y $Pan\}$, entonces por costum-

bre también comprará $\{Mantequilla\}$ y según la especificación formal anterior se tiene que:

$$X=\{Leche,\ Pan\}\ \ Y=\{Mantequilla\}$$

El ejemplo anterior es considerablemente pequeño, pues en la práctica una regla necesita tener soporte de varios cientos de registros (transacciones) antes de que esta pueda considerarse significativa desde un punto de vista estadístico, donde el conjunto $\{Leche$ y $Pan\}$ tiene un soporte de

$$sop(X) = \frac{2}{5} = 0{,}4$$

Es decir, el soporte es del 40 % (esto es, dos de cada cinco transacciones). Por ejemplo, para la regla arriba indicada la medida de la confianza es:

$$conf\left(\{Leche, Pan\} \Rightarrow \{Mantequilla\}\right) = \frac{sop\,|\{Leche, Pan\} \cup \{Mantequilla\}|}{sop\,|\{Leche, Pan\}|}$$

$$conf\left(\{Leche, Pan\} \Rightarrow \{Mantequilla\}\right) = \frac{0{,}2}{0{,}4} = 0{,}5$$

Lo anterior significa que la regla resulta cierta en el 50 % de los casos. Por otro lado, encontrar todos los subconjuntos frecuentes de la base de datos es difícil, ya que esto implica considerar a todos los posibles subconjuntos de atributos de tamaño $(2^n - 1)$, donde n es el número de *items*. Obviamente se excluye al conjunto vacío debido a que no es válido como conjunto de atributos.

Cabe destacar lo siguiente: aunque el tamaño del conjunto de *itemsets* crece exponencialmente con el número de *items*, es posible hacer una búsqueda eficiente utilizando la *propiedad de clausura descendente* del soporte (también llamada *antimonótona*) que garantiza que para un conjunto de atributos frecuentes (es decir, con alto valor de soporte), todos sus subconjuntos también son frecuentes y, del mismo modo, para un conjunto de atributos no frecuentes, todos sus superconjuntos deben ser no frecuentes [Agrawal, 1993].

En resumen, **el algoritmo Apriori es capaz de generar reglas de asociación porque se basa en el cálculo y selección del conjunto de atributos que presentan la combinación bien predicha y el porcentaje de aciertos de la regla máximos, para la aplicación de cada acción**.

Por ejemplo, para el árbol de decisión de la Figura 3.2, este algoritmo entrega las siguientes reglas de asociación:

1. outlook=overcast 4 ==> play=yes 4 conf:(1)
2. temperature=cool 4 ==> humidity=normal 4 conf:(1)
3. humidity=normal windy=FALSE 4 ==> play=yes 4 conf:(1)
4. outlook=sunny play=no 3 ==> humidity=high 3 conf:(1)

5. outlook=sunny humidity=high 3 ==> play=no 3 conf:(1)

donde los dígitos 3 (tres) y 4 (cuatro) son la medida del soporte de la regla correspondiente y el valor 1 (uno) significa el 100 % de confianza. Una vez estudiado cómo reducir el espacio de búsqueda de acciones, a continuación se estudia cómo reducir el espacio de búsqueda de estados en los MDP.

3.2. Priorización de Estados

En el estado del arte se encontró una estrategia que promete reducir, en buena medida, el **espacio de búsqueda de estados** en los MDP y este es el barrido priorizado de McMahan *et al.* [McMahan, 2005a,b]. En esta tesis se decidió estudiarla debido a que ha obtenido resultados alentadores al resolver MDP deterministas, pero que ante MDP estocásticos ha obtenido planes subóptimos [Li, 2009].

A partir del enfoque de barrido priorizado de Moore *et al.* [Moore, 1993], McMahan *et al.* [McMahan, 2005a,b] propusieron un enfoque mejorado (vea Algoritmo 3.2). Este algoritmo fue originalmente concebido como una extensión del clásico algoritmo de Dijkstra para la solución de problemas de ruta más corta, con recompensas positivas y con un estado absorbente, que es el estado meta. A continuación se describen estos algoritmos.

3.2.1. Algoritmo de Dijkstra

El clásico algoritmo de Dijkstra calcula simultáneamente las rutas más cortas a cada nodo del grafo desde el punto de partida. Cabe señalar que este algoritmo no tiene orientación a meta porque no utiliza heurística. Sin embargo, se ha demostrado que este algoritmo es un método de aproximación sucesiva que resuelve la ecuación de programación dinámica para el problema de ruta más corta [Sniedovich, 2006, 2010].

A continuación se describe al clásico algoritmo de Dijkstra (1956):

Condiciones iniciales

- R es el conjunto de estados pertenecientes a la ruta más corta, que inicia conteniendo solamente al estado inicial.
- S el conjunto de estados candidatos a pertenecer a la ruta más corta, que inicia con todos los estados del problema, excepto el estado inicial.
- D contiene la distancia acumulada entre el estado en evaluación (en este caso inicial) y cada estado adyacente. La distancia a un estado no adyacente es infinita.

Mejorando sucesivamente la ruta hasta entregar la ruta óptima

1. Saca un estado no visitado de S con menor distancia acumulada al estado en evaluación y lo inserta en R.
2. Suma la distancia acumulada al estado en evaluación con la distancia a los estados adyacentes a él.
3. Compara la nueva distancia con la anterior distancia acumulada e inserta en D a la menor, regresando a S a los estados no ganadores.
4. Marca al estado ganador como estado visitado, lo inserta en R y vuelve al paso 1.
5. Cuando se agota S, entonces entrega la ruta óptima R.

Es importante señalar que el algoritmo de Dijkstra está basado en el *principio de optimalidad de Bellman* [Puterman, 2005].

La principal diferencia entre este algoritmo y otros métodos de programación dinámica al calcular problemas de ruta más corta es el orden particular en que procesa estados, pues lo hace de acuerdo a la regla "primero el mejor", usando colas de prioridad. Este algoritmo escoge como el siguiente estado en ser procesado al que tenga la más pequeña función de valor debido a que está buscando la ruta más corta. Sin embargo, este ordenamiento **no ha resultado óptimo al resolver MDP no deterministas** [Li, 2009].

3.2.2. Algoritmo de Barrido Priorizado

Como ya se señaló, McMahan *et al.* [McMahan, 2005a,b] propusieron a IPS (*Improved Prioritized Sweeping*), un enfoque mejorado del barrido priorizado (PS) de Moore *et al.* [Moore, 1993]. En esta tesis se decidió estudiar al algoritmo IPS dado que es capaz de **reducir al algoritmo de Dijkstra** con el objeto de resolver a los MDP deterministas y, como se trata de un **algoritmo de un solo paso**, es uno de los métodos más rápidos del estado del arte para resolverlos.

La idea clave de IPS es mantener una cola de prioridad, conformada por estados que prometen alcanzar al estado meta, expandiendo a los estados más prometedores. Desafortunadamente la convergencia a la solución óptima sólo está garantizada para el caso determinista de MDP de ruta más corta (que no corresponden al mundo real) [Li, 2009].

Una característica del algoritmo IPS es que considera a todos los estados predecesores del estado en evaluación, no solamente a sus predecesores en el grafo de políticas. Este algoritmo mantiene una cola de prioridad acomodando en orden decreciente a los estados en las actualizaciones, iniciando por el estado

meta e insertando a los estados adyacentes con probabilidad de transición diferente de cero, y así sucesivamente en forma regresiva hasta alcanzar al estado inicial. De esta manera IPS actualiza cada estado una sola vez en cada iteración.

Por otro lado, este algoritmo usa el concepto de **actualización múltiple**, es decir que actualiza al estado predecesor sin verificar su función de valor, debido a que este ya había sido actualizado anteriormente. Logra la convergencia cuando queda vacía la cola de prioridad. Sin embargo, una desventaja de este algoritmo es que puede procesar actualizaciones innecesarias.

Es importante enfatizar que la principal diferencia entre los algoritmos PS e IPS radica en la forma en que los estados son actualizados. Por ejemplo, después de que PS arroja a un estado fuera de cola de prioridad, este algoritmo **actualiza a dicho estado** y cuando el error de Bellman está sobre el umbral establecido, entonces empuja dentro de cola a todos sus estados predecesores.

En contraste, cuando IPS arroja al estado con mayor prioridad fuera de cola, entonces **actualiza a todos los predecesores de dicho estado** y cuando el error de Bellman de un estado predecesor está por encima del valor de dicho umbral, entonces lo empuja dentro de cola o cambia su prioridad si ese estado aún permanece en cola. Sin embargo IPS no garantiza la convergencia en MDP no deterministas [Dai, 2007a,b] [Li, 2009]. Además solo resuelve problemas con un estado meta y un estado de inicio, lo cual no corresponde al mundo real.

Sea $U(s)$ el coste esperado para que un autómata alcance la meta cuando se encuentra en el estado $s \in S$, $Q(s, a)$ significa lo mismo que $U(s)$ excepto que la primera acción debe ser $a \in A$ y $\pi(s)$ es la política voraz o acción a ejecutar a partir del estado s para alcanzar al estado meta. Básicamente IPS ejecuta los siguientes pasos: inicio, remoción priorizada de cada estado habido en cola iniciando desde el estado meta y actualizando a sus predecesores, repitiendo el proceso hasta que la cola queda vacía. A continuación se describen los pasos de este algoritmo.

Primer paso: ejecuta un pésimo inicio asignando un valor constante positivo suficientemente grande M a $U(s)$ y a $Q(s, a)$, excepto cuando se trata del único estado meta ($meta \in S$). En este caso, IPS le asigna valor cero a la variable $Q(meta, a)$ para cada acción a y el valor de -1 a la política del estado meta $\pi(meta)$.

Segundo paso: expande al estado meta y actualiza la función de valor (coste esperado) de todos sus predecesores. Como resultado de esto, algunos estados predecesores podrían ser empujados dentro de cola de prioridad.

Tercer paso: el estado con mayor prioridad es empujado afuera de cola y sus predecesores son actualizados. Como en el segundo paso, algunos predecesores del estado actual podrían ser empujados dentro de cola de prioridad o podría

ser cambiada su prioridad. Este paso es repetido hasta que la cola de prioridad queda vacía, en cuyo caso se declara la convergencia del algoritmo.

Al respecto, cada vez que IPS **actualiza** a los estados predecesores de un estado prometedor s, entonces ejecuta los siguientes pasos:

(a) guarda el valor óptimo de la función de valor del estado en evaluación,

$$U(s) \leftarrow Q(s, \pi(s)) \tag{3.3}$$

(b) para cada estado predecesor $y \in pred(s)$ del estado en evaluación s, recalcula los valores de Q solamente para acciones b que podrían alcanzar s,

$$Q(y, b) \leftarrow R(y, b) + \sum_{s' \in suc(y,b)} P(s' \,|y, b) Q(s', \pi(s'))) \tag{3.4}$$

donde $R(y, b)$ es el coste esperado de la ejecución de la acción b desde el estado y, $P(s' \,|y, b))$ es la probabilidad de alcanzar el estado s' cuando la acción b es aplicada sobre el estado y. Por otro lado, $suc(y, b)$ es el conjunto de todos los posibles sucesores del estado y bajo la acción b, excepto en el caso del estado meta que siempre es excluído.

(c) siempre que un nuevo valor de Q sea más bajo que su valor previo, IPS calcula la prioridad de expandir al estado y desde la acción b,

$$pri(y, b) = \frac{Q(y, b) - U(y)}{Q(y, b)} \tag{3.5}$$

(d) entonces la acción b es tomada como la política que se debe aplicar sobre el estado y,

$$\pi(y) \leftarrow b. \tag{3.6}$$

(e) si $|U(y) - Q(y, b)| > \varepsilon$, donde ε es el máximo error permitido (ecuación (2.6)), entonces el estado y es empujado dentro de cola de prioridad o su prioridad es cambiada si este permanece aún en cola.

Al final entrega la política requerida para que un autómata alcance la meta mediante la ruta más corta.

Como ya se mencionó, IPS **converge en un solo paso** a la política óptima solo para el caso de MDP deterministas. Esto lo logra en cada paso al iniciar desde la función de valor calculada en el paso previo (de ahí que se le llame **algoritmo de barrido**). En consecuencia, IPS es una de las más rápidas alternativas que permite varias actualizaciones de un estado (también se le conoce como **algoritmo de actualización múltiple**) y esto lo logra en una sola pasada.

Algoritmo 3. 2 IPS [McMahan, 2005a,b].

$(R, S, A, P, \varepsilon)$
$(\forall s \in S)U(s) \leftarrow M$ // M es un número arbitrario positivo grande
$(\forall s \in S, a \in A)Q(s, a) \leftarrow M$
$(\forall a \in A)Q(meta, a) \leftarrow 0$ // donde $meta \in S$
$(\forall s \in S)\pi(s) \leftarrow$ indefinido
$\pi(meta) =$ arbitrario
actualiza($meta$)
mientras (¬colaVacia()) **hacer**
  $s \leftarrow$ colaArroja()
  actualiza(s)
fin mientras
regresa π

actualiza(s)
$U(s) \leftarrow Q(s, \pi(s))$
para todo $(y, b) \in pred(s)$ **hacer**
  $Q_{\text{anterior}} \leftarrow Q(y, \pi(y))$ (o M si $\pi(y)$ es indefinido)
  $Q(y, b) \leftarrow R(y, b) + \sum_{s' \in succ(y,b)} P(s'|y, b)Q(s', \pi(s'))$
  si $(Q(y, b) < Q_{\text{anterior}})$ **entonces**
    $pri \leftarrow \frac{Q(y,b)-U(y)}{Q(y,b)}$
    $\pi(y) \leftarrow b$
    si $(|U(y) - Q(y, b)| > \varepsilon)$ **entonces**
      decreceColaPrioridad(y, pri)
    fin si
  fin si
fin para
regresa

Sin embargo, para el caso de problemas de ruta estocástica más corta son necesarios múltiples pasos para la convergencia a la política óptima. En este sentido, si se acelera la convergencia de IPS por algún método del estado del arte, como es un típico método basado en la función de valor, entonces se declararía convergencia cuando el máximo *error de Bellman* quedara por debajo del umbral previamente establecido ε.

Para ello **en esta tesis se modificó a IPS de modo que pudiese tratar con problemas estocásticos de considerables dimensiones**, así como con múltiples estados meta y múltiples estados de inicio. En primera instancia se le adaptó a IPS el uso de **acciones reglamentadas**. Cabe recordar que estas acciones corresponden a la representación compacta (*sparse*) de la función de transición, contenidas en una lista de adyacencia de las transiciones de estados con probabilidad diferente de cero.

A la anterior modificación se le denominó **SIPS** (*Sparse Improved Prioritized Sweeping*), que ejecuta básicamente los mismos pasos que IPS, excepto que SIPS utiliza una lista de adyacencia $L = \{\ell_k | \ell_k = (s_k, s'_k, a_k, p_k), p_k = P(s'_k|s_k, a_k) \neq 0\}$ donde el k-ésimo elemento, que es una regla de asociación, relaciona a un estado inicial s_k con la acción aplicada a_k en dicho estado, al estado final resultante s'_k de aplicar dicha acción y a la probabilidad de transición de estados p_k, que es la **confidencia de la regla**. Considerando lo anterior es claro que **SIPS presentará las mismas ventajas y desventajas que IPS**, por lo que SIPS será capaz de introducir varios estados meta en cola de prioridad al inicio del proceso, de acuerdo con Dai *et al.* [Dai, 2007c].

Por último, es importante señalar el esfuerzo que se ha hecho en cuanto al aprendizaje de la función de valor y de la función de política mediante redes neuronales [Cervellera, 2007]. También el realizado por Fernandez *et al.* [Fernandez, 2008], que propusieron un método de aproximación de funciones de valor basadas en aprendizaje supervisado (algoritmo CMAC Q-Learning), basado en el controlador robótico CMAC propuesto en 1975 por James Albus. Su método está basado en segmentación uniforme. Sin embargo estas estrategias están diseñadas para resolver MDP parcialmente observables (POMDP) [Amstron, 1965]. En cambio en esta tesis solamente se estudian y resuelven **MDP totalmente observables**. Para ello se analizó el comportamiento de las variantes del algoritmo de iteración de valor pertenecientes al estado del arte que han tenido mejor desempeño frente a problemas altamente dimensionales, especialmente aquellas basadas en priorización.

3.3. Conclusiones del Capítulo

Entre las técnicas de reducción del espacio de búsqueda, encontradas en el estado del arte, están las basadas en minería de datos, en priorización y en búsqueda heurística. De ellas se seleccionó en esta tesis los enfoques que han reportado una considerable reducción del espacio de búsqueda, tanto de acciones como de estados. El primero de ellos obtiene acciones en función de estado mediante minería de datos. El segundo es el barrido priorizado de estados, que entrega la política óptima en MDP deterministas. Esta última tiene la virtud de entregar al subconjunto de estados útiles durante la búsqueda de la ruta más corta entre dos puntos, aplicando una extensión del clásico algoritmo de Dijkstra. En el siguiente capítulo se aplicarán acciones en función de estado durante el proceso de inferencia del algoritmo de iteración de valor y se le aplicará el ordenamiento dinámico del barrido priorizado, pero con otro criterio de prioridad, para que sea capaz de encontrar el plan óptimo en tiempo tratable.

Capítulo 4

Simplificación de los Procesos de Decisión de Markov aplicados a la búsqueda de la ruta más corta mediante Reglamentación de Acciones y Priorización de Estados

Con el objeto de darle tratabilidad a los procesos de decisión de Markov (MDP por sus siglas en inglés) de considerables dimensiones, específicamente en problemas de ruta estocástica más corta, en este capítulo se propone la reducción de su espacio de búsqueda mediante un nuevo enfoque del algoritmo de iteración de valor priorizado con acciones reglamentadas, basado en el clásico algoritmo de Dijkstra.

En primera instancia se propone **reducir el espacio de búsqueda de acciones** del algoritmo de iteración de valor, mediante la aplicación de reglas de asociación durante su proceso de inferencia. Cada regla consiste en una acción en función del estado en evaluación, simplificando en buena medida el dominio sobre el que calcula el MDP.

En segunda instancia se propone **reducir el espacio de búsqueda de estados** del algoritmo de iteración de valor mediante un ordenamiento dinámico de los estados más prometedores. Esta estrategia utiliza al clásico algoritmo de Dijkstra, eficiente método que resuelve problemas de ruta más corta en un grafo acíclico ponderado, pero modificado para que sea capaz de resolver grafos ponderados altamente cíclicos, como son los MDP. De aquí resulta el enfoque que se propone en esta tesis.

Por último, es importante señalar que el enfoque aquí propuesto solamente funciona para MDP planos, es decir no factorizados, y completamente observables. También cabe destacar que el análisis de la complejidad de los algoritmos propuestos en este capítulo se presentará al final del Capítulo 6.

4.1. Iteración de Valor con Reglamentación de Acciones

En esta sección se presenta la aplicación de reglas de asociación al algoritmo de iteración de valor, de la que resultó **ARVI** por sus siglas en inglés (*Association-Rule based Value Iteration*) [García-Hernández, 2009] (vea Algoritmo 4.1).

Con respecto a las técnicas de solución de los MDP, es importante señalar que el aprendizaje por refuerzo es una variante del control óptimo. Sin embargo, dicho control solamente involucra a la planificación, mientras que el aprendizaje por refuerzo involucra tanto al aprendizaje (del modelo de transición y de la función recompensa) como a la planificación. Lo anterior es debido a que el aprendizaje por refuerzo es un método no informado que debe explorar el entorno para transformarse en un método informado, como lo es iteración de valor, por ejemplo.

Por otro lado, los métodos de aprendizaje por refuerzo pueden ser ampliamente clasificados como: directos (**libres de modelo**) e indirectos (**basados en el modelo**) [Sutton, 1998]. En contraste con los métodos libres de modelo -que calculan una política óptima directamente de los datos experimentales- los métodos basados en el modelo primero estiman el MDP subyacente y luego usan técnicas estándares, tales como el algoritmo de iteración de valor [Puterman, 1994], para calcular una política óptima.

En consecuencia, los métodos basados en el modelo son más eficientes que los métodos libres de modelo. Un método usado para la estimación de parámetros del MDP (probabilidades de transición de estados y la función recompensa) está basado en la **máxima verosimilitud**. Este estimador es el valor del parámetro que maximiza la verosimilitud de los datos.

Sea $n(s, a)$ el número de veces que el agente ha aplicado la acción a en el estado s. Sea $n(s, a, s')$ el número de veces en que este alcanzó el estado s' después de haber ejecutado la acción a en el estado s. Sea $\sum R(s, a)$ la cantidad acumulada de recompensas recibidas por el agente al aplicar la acción a en el estado s. Los estimadores de máxima verosimilitud [Scherrer, 2006] de las probabilidades de transición y recompensas están dados por:

$$\hat{T}(s, a, s') = \frac{n(s, a, s')}{n(s, a)} \tag{4.1}$$

y

$$\hat{R}(s, a) = \frac{\sum R(s, a)}{n(s, a)} \tag{4.2}$$

respectivamente.

Como se estudió en el anterior capítulo, del estado del arte se extrajo Apriori [Agrawal, 1993, 2002] (vea algoritmo 3.1), el cual es un algoritmo de minería de datos que calcula, desde los datos de exploración de su entorno obtenidos por un autómata, al conjunto de reglas de asociación de la forma $\{s, a\} \Rightarrow \{s'\}$, donde s es el estado en evaluación y s' es el estado resultante después de ejecutar la acción a.

En este trabajo se seleccionó al algoritmo Apriori con el objeto de aplicarlo en el algoritmo de iteración de valor (enfoque de programación dinámica que resuelve a los MDP) para reducir, en buena medida, su complejidad y por ende su tiempo de solución.

Por otro lado, es importante señalar las **ventajas del uso de reglas de asociación en los MDP**:

- que estas reglas pueden ser calculadas paralelamente al proceso de inferencia del MDP mediante Apriori,
- que la medida de confidencia de cada regla es su probabilidad de transición.

También las recompensas pueden ser calculadas al mismo tiempo usando la ecuación (4.2). Para conjuntos de datos considerablemente grandes, el número de pasadas a través del conjunto de datos puede hacer que resulten inviables algunos algoritmos de minería de datos. Al respecto, un algoritmo basado en muestreo (por ejemplo FPMax [Ceglar, 2006]) puede ser usado para obtener aproximaciones sucesivas del conjunto de reglas de asociación. Derivado de esas aproximaciones, un MDP aproximado puede fácilmente ser obtenido y resuelto al mismo tiempo en que el agente extrae datos de su entorno. A continuación se presenta el detalle de pasos contenidos en el algoritmo ARVI.

Sea L una lista de adyacencia calculada por Apriori, donde $L = \{\ell_k|\ell_k = (s_k, s'_k, a_k, p_k), p_k = P(s'_k|s_k, a_k) \neq 0\}$, en la que el k-ésimo elemento, que es una regla de asociación, relaciona un estado inicial s_k con la acción aplicada a_k en dicho estado y al estado final resultante s'_k y su probabilidad de transición de estados p_k, que es la **confidencia de la regla**. Este conjunto devuelve soporte y confidencia máximos (que corresponde a la lista de adyacencia de la representación *sparse* de la matriz de probabilidad de transición de estados).

Sea $R = \{R(s,a)\,|s = 1, \ldots, n, a = 1, \ldots, m\}$ el conjunto de recompensas de cada par estado-acción y sea $P = \{p_k\,|p_k = (s_k, s'_k, a_k)\}$ el conjunto de probabilidades de transición de cada regla, donde n es el número de estados, γ es el factor de descuento, ε es el máximo error y π es la política (o acción) óptima.

Primer paso: El agente inicia explorando el entorno y entrega la base de datos a ARVI para que, mediante Apriori, obtenga el conjunto $L = \{\ell_k|\ell_k = (s_k, s'_k, a_k, p_k)\}$ de acciones reglamentadas.

Segundo paso: Asigna la recompensa de mayor valor para cada par estado-acción (s, a) -correspondiente a los n estados del problema- a la variable $U^0(s)$:

$$U^0(s) \leftarrow \max_a R(s,a) \tag{4.3}$$

Tercer paso: Calcula la ecuación de Bellman para los estados del problema y las acciones del dominio que correspondan al conjunto L de reglas de asociación (proveniente de Apriori), previamente obtenido durante la fase de exploración del entorno por un autómata. Para ello efectúa lo siguiente:

(a) guarda el valor de la recompensa estado-acción $R(s, a)$ en una variable local para n estados y m acciones:

$$J(s,a) \leftarrow R(s,a) \tag{4.4}$$

(b) para la k-ésima regla de asociación del conjunto L, asigna a la variable s el valor del estado inicial, así como el valor de la acción ejecutable, a, en dicho estado y el valor del estado final alcanzado, s':

$$a \leftarrow \text{accion}\,(\ell_k),\ s \leftarrow \text{estadoinicial}\,(\ell_k),\ s' \leftarrow \text{estadofinal}\,(\ell_k),$$

y recalcula la variable $J(s, a)$ mediante la siguiente ecuación:

$$J(s,a) \leftarrow J(s,a) + \gamma p_k U^{t-1}(s') \tag{4.5}$$

(c) calcula la ecuación de Bellman para cada variable recalculada $J(s, a)$ y selecciona a la que entregue mayor valor, para n estados:

$$U^t(s) \leftarrow \max_a J(s,a) \tag{4.6}$$

(d) repite iteraciones hasta que el error de Bellman (vea ecuación (2.6)) sea menor o igual que el umbral predeterminado de paro ε:

$$\left\|U^t - U^{t-1}\right\|^2 > \varepsilon \tag{4.7}$$

Cuarto paso: en cuanto obtiene la convergencia del paso anterior, asigna el valor de la acción que entregó la máxima utilidad a la variable $\pi(s)$, que es la acción que entrega mayor utilidad al estado s, para n estados:

$$\pi(s) \leftarrow \underset{a}{\operatorname{argmax}}\, J(s,a) \tag{4.8}$$

y al final entrega la política óptima π, que es la secuencia de acciones que deberá ejecutar un autómata para alcanzar la meta.

Algoritmo 4. 1 ARVI (iteración de valor basado en reglamentación de acciones [Garcia-Hernandez, 2009]).

función ARVI$(R, L, P, \gamma, \varepsilon)$
$L = \{\ell_k | \ell_k = (s_k, s'_k, a_k, p_k)\}$ /es el conjunto de reglas entregadas por Apriori

$U^0(s) \leftarrow \underset{a}{\text{máx}}\, R(s,a)$ **para** $s = 1, 2, ..., n$
$t \leftarrow 1$
hacer
  $J(s,a) \leftarrow R(s,a)$ **para** $s = 1, 2, ..., n$ y $a = 1, 2, ..., m$
  para $k = 1$ **a** $|L|$ **hacer**
    $a \leftarrow \text{accion}\,(\ell_k)$
    $s \leftarrow \text{estadoinicial}\,(\ell_k)$
    $s' \leftarrow \text{estadofinal}\,(\ell_k)$
    $J(s,a) \leftarrow J(s,a) + \gamma p_k U^{t-1}(s')$
  fin para
  $U^t(s) \leftarrow \underset{a}{\text{máx}}\, J(s,a)$ **para** $s = 1, 2, \ldots, n$
  $t \leftarrow t + 1$
  mientras $\left\|U^t - U^{t-1}\right\|^2 > \varepsilon$ **hacer**
    $\pi(s) \leftarrow \underset{a}{\operatorname{argmax}}\, J(s,a)$ **para** $s = 1, 2, ..., n$
  fin mientras
fin hacer
regresa π

En resumen, ARVI inicia calculando la utilidad esperada $J(s,a)$ para cada regla ℓ_k y su correspondiente probabilidad de transición p_k. Después calcula la máxima utilidad esperada y la política óptima de cada estado, que corresponde a la acción de $J(s,a)$. Después de varias iteraciones, este algoritmo entrega la política óptima que otorga la máxima utilidad (presente y futura) del estado en

evaluación.

Por último, aunque no se aplicaron técnicas de aceleración a este algoritmo, se espera que ARVI sea más rápido que el clásico iteración de valor debido a que solo utilizará una lista de reglas, no la costosa matriz tridimensional de probabilidades de transición que utiliza este último algoritmo.

4.2. Enfoque Propuesto: Iteración de Valor Priorizado con Reglamentación de Acciones

En esta sección se presenta el enfoque que se propone en esta tesis: **IPVI** (*Improved Prioritized Value Iteration*) [Garcia-Hernandez, 2012a] (vea Algoritmo 4.2). Este es un nuevo algoritmo de iteración de valor priorizado que usa acciones reglamentadas, basado en el clásico algoritmo de Dijkstra, para resolver MDP de ruta estocástica más corta.

A diferencia de otros enfoques priorizados tales como el barrido priorizado de [McMahan, 2005a,b], el enfoque propuesto actualiza sucesivamente a cada estado relevante utilizando la ecuación de Bellman completa, por lo que garantiza la convergencia a la solución óptima. La **métrica de prioridad es la función de valor actual**, pues el algoritmo de Dijkstra sugiere que un orden de actualización más adecuado está dado por el valor de la programación dinámica funcional. Por último, este enfoque es capaz de tratar con múltiples estados meta y de inicio.

Cabe recordar que una de las ventajas del algoritmo de iteración de valor y de sus variantes aceleradas (en particular barrido priorizado), es que la convergencia a la función de valor óptima está garantizada para el caso de MDP descontados (con $0 < \gamma < 1$) y para el caso de MDP aditivos con estados absorbentes [Bertsekas, 1995], donde $\gamma = 1$. Esto es debido a que las aplicaciones sucesivas de la ecuación de Bellman garantizan dicha convergencia. Por esta razón en IPVI se actualizan los estados predecesores del mejor estado (el que tiene la más alta función de valor) usando dicha ecuación.

En resumen, el algoritmo IPVI trabaja de la siguiente manera: inicia introduciendo en cola de prioridad a los estados meta. Luego va sacando cada estado habido en cola y actualizando la función de valor de sus estados predecesores. Cuando el error de Bellman de un estado predecesor es mayor que el umbral establecido ε, entonces devuelve dicho estado a cola de prioridad. En caso contrario, lo entrega como política óptima. Esto significa que hace **ordenamiento dinámico de estados**. También es importante señalar que este nuevo algoritmo usa operaciones de expansión y calcula toda la ecuación de Bellman para cada estado predecesor al estado en evaluación, que corresponde a la **métrica**

de prioridad.

Sea $U^t(s)$ el coste esperado en el tiempo t para que un autómata alcance un estado meta iniciando desde el estado $s \in S$, $\pi^t(s)$ es la política o acción a ejecutar en el tiempo t y estado s, $R(s,a)$ es la recompensa para la acción a en el estado s, $L = \{\ell_k | \ell_k = (s_k, s'_k, a_k, p_k), p_k = P(s'_k|s_k, a_k) \neq 0\}$ es la lista de adyacencia conteniendo las probables transiciones de estado, G es el conjunto de estados meta, el cual es un subconjunto de S, γ es el factor de descuento y ε es el mínimo error permitido (es decir, el umbral predeterminado de paro).

Básicamente IPVI ejecuta los siguientes pasos: inicio seguido de una sucesiva remoción priorizada de cada estado habido en cola con actualización de sus predecesores (ordenamiento dinámico), hasta que cola queda vacía.

Primer paso: El agente inicia explorando el entorno y entrega la base de datos a ARVI para que, mediante Apriori, obtenga el conjunto $L = \{\ell_k | \ell_k = (s_k, s'_k, a_k, p_k)\}$ de todas las acciones reglamentadas.

Segundo paso: para cada estado $s \in S$, inicia con $\pi^0(s) = -1$, si $s \notin G$ entonces se le asigna un valor grande y positivo a $U^0(s)$, y en caso contrario se le asigna valor de cero.

Tercer paso: empuja cada estado meta $s \in G$ dentro de cola de prioridad. Después repite lo siguiente hasta que queda vacía la cola de prioridad: saca de cola al estado s con la más alta prioridad y después actualiza a sus predecesores $y \in pred(s)$, regresando a cola de prioridad a aquellos estados que cumplen con el umbral prestablecido en la diferencia de función de valor. En cuanto se vacía la cola de prioridad se declara la convergencia del algoritmo.

En cada **actualización** de un predecesor del estado en evaluación s, se calcula completa la ecuación de Bellman (vea ecuación (2.3)) para cada acción $a \in pred(s)$:

$$U^{t+1}(y) = \max_a \left\{ R(y,a) + \gamma \sum_{\forall (s_k = y, s'_k, a_k = a, p_k) \in L} p_k U^t(s'_k) \right\} \tag{4.9}$$

y se actualiza la política actual:

$$\pi^{t+1}(y) \leftarrow \arg\max_a \left\{ R(y,a) + \gamma \sum_{\forall (s_k = y, s'_k, a_k = a, p_k) \in L} p_k U^t(s'_k) \right\} \tag{4.10}$$

y si $(\left|U^{t+1}(y) - U^t(y)\right| > \varepsilon)$, entonces empuja dentro de cola al estado y de acuerdo con su prioridad $U^{t+1}(y)$. En caso de que el estado y permanezca en

cola, su prioridad solamente se actualiza.

Algoritmo 4. 2 IPVI (Improved Prioritized Value Iteration [Garcia-Hernandez, 2012a]).

$(R, S, A, L, P, \gamma, \varepsilon)$
$L = \{\ell_k | \ell_k = (s_k, s'_k, a_k, p_k), p_k = P(s'_k|s_k, a_k) \neq 0\}$ //es el conjunto de reglas de Apriori
$(\forall s \in S)U(s) \leftarrow M$ //M es un número arbitrario positivo y suficientemente grande
$(\forall s \in G)U(s) \leftarrow 0$ //G es el conjunto de estados meta y es un subconjunto de S
$(\forall s \in S)\pi(s) \leftarrow$ indefinido
$(\forall s \in G)\pi(s) =$ arbitrario
$(\forall s \in G)$entraEnCola$(s, U(s))$
mientras $(\neg$colaVacia$())$ **hacer**
  $s \leftarrow$ colaArroja()
  actualizaPred(s)
fin mientras
regresa π

actualizaPred(s)
para todo $(y, a) \in$ pred(s) **hacer**
  $U'(y) \leftarrow U(y)$
  $$U(y) = \max_a \left\{ R(y,a) + \gamma \sum_{\forall (s_k = y, s'_k, a_k = a, p_k) \in L} p_k U'(s'_k) \right\}$$
  $$\pi(y) \leftarrow \arg\max_a \left\{ R(y,a) + \gamma \sum_{\forall (s_k = y, s'_k, a_k = a, p_k) \in L} p_k U'(s'_k) \right\}$$
  si $(|U(y) - U'(y)| > \varepsilon)$ **entonces**
    decreceColaPrioridad$(y, U(y))$
  fin si
fin para
regresa

Es importante señalar que el algoritmo aquí propuesto difiere de IPS, así como de las otras variantes de iteración de valor aquí descritas, en los siguientes aspectos:

(i) IPVI ejecuta varias actualizaciones de cada estado predecesor del estado en evaluación calculando la ecuación de Bellman completa, obteniendo entonces la política óptima. En cambio, IPS solamente calcula algunos valores de Q, pero no actualiza inmediatamente la función de valor, no logrando la convergencia al plan óptimo.

(ii) en IPVI el criterio de prioridad de un estado s es igual a la nueva función

de valor disponible $U^{t+1}(s)$, mientras en IPS es igual a $\frac{Q^{t+1}(s,b)-U^t(s)}{Q^{t+1}(s,b)}$, donde b es la acción para el que $Q^{t+1}(s,b) < Q^t(s,b)$.

(*iii*) IPVI puede tratar con múltiples estados meta y múltiples estados de inicio, mientras que IPS sólo puede tratar con un estado meta y un estado de inicio.

Es notorio que estas diferencias reducen significativamente al espacio de búsqueda de iteración de valor, por lo que el enfoque aquí propuesto (IPVI) promete exhibir mejor desempeño que demás algoritmos estudiados, en una tarea altamente dimensional de ruta estocástica más corta, la cual se presenta en el siguiente capítulo.

Por otro lado, como se señaló anteriormente, la velocidad de convergencia del algoritmo de iteración valor depende de un buen ordenamiento de estados, por lo que se estudiaron diversas formas de actualización de los estados predecesores en IPVI, el cual es asíncrono y presenta repeticiones en la lista de transiciones de estados predecesores y no ordena a los estados predecesores antes de actualizarlos.

La primera de estas variantes se presenta en el Algoritmo 4.3 (*Jacobi Improved Prioritized Value Iteration*, JIPVI), donde se actualiza de forma asíncrona a todos los predecesores de cada estado prometedor por el método de Jacobi [Shlakhter, 2005], sin tener en cuenta su orden.

La segunda de estas variantes se presenta en el Algoritmo 4.4 (*Prioritized Predecessors Improved Prioritized Value Iteration*, PPIPVI), donde se actualiza de forma asíncrona a todos los predecesores de cada estado prometedor, pero estas actualizaciones son priorizadas de acuerdo a la función de valor de cada predecesor.

Sin embargo, con este ordenamiento adicional se espera que PPIPVI presente un coste de inicio adicional a JIPVI y a IPVI. Es decir, se espera que actualizar en forma priorizada de acuerdo a la función de valor de cada predecesor sea más costoso que actualizar de forma asíncrona a todos los predecesores de cada estado prometedor por el método de Jacobi [Shlakhter, 2005], sin tener en cuenta su orden.

4.3. Conclusiones del Capítulo

En este capítulo se presentó, en dos secciones, el enfoque propuesto para resolver MDP altamente dimensionales, con múltiples estados meta y de inicio. Cabe destacar que el enfoque aquí propuesto solamente funciona para MDP planos, es decir no factorizados, y completamente observables.

En la primera sección se estudió la aplicación de acciones en función del estado en evaluación en el clásico algoritmo de iteración de valor (que resuelve a

Algoritmo 4. 3 JIPVI (Jacobi Improved Prioritized Value Iteration [Garcia-Hernandez, 2012b]).

$(R, S, A, L, P, \gamma, \varepsilon)$
$L = \{\ell_k | \ell_k = (s_k, s'_k, a_k, p_k), p_k = P(s'_k|s_k, a_k) \neq 0\}$ //es el conjunto de reglas de Apriori
$(\forall s \in S)U(s) \leftarrow M$ //M es un número arbitrario positivo y suficientemente grande
$(\forall s \in G)U(s) \leftarrow 0$ //G es el conjunto de estados meta y es un subconjunto de S
$(\forall s \in S)\pi(s) \leftarrow$ indefinido
$(\forall s \in G)\pi(s) =$ arbitrario
$(\forall s \in G)$entraEnCola$(s, U(s))$
mientras $(\neg$colaVacia$())$ **hacer**
  $s \leftarrow$ colaArroja()
  actualizaPred(s)
fin mientras
regresa π

actualizaPred(s)
para todo $(y, a) \in$ pred(s) **hacer**
  $U'(y) \leftarrow U(y)$
  $$U(y) = \max_a \left\{ (1 - \gamma p_{yya})^{-1} R(y, a) + \gamma \sum_{\forall (s_k = y, s'_k, a_k = a, p_k) \in L} p_k U'(s'_k) \right\}$$
  $$\pi(y) \leftarrow \arg\max_a \left\{ R(y, a) + \gamma \sum_{\forall (s_k = y, s'_k, a_k = a, p_k) \in L} p_k U'(s'_k) \right\}$$
  si $(|U(y) - U'(y)| > \varepsilon)$ **entonces**
    decreceColaPrioridad$(y, U(y))$
  fin si
fin para
regresa

los MDP). Lo anterior se hizo con el objeto de reducir el espacio de búsqueda de acciones de este formalismo. Para obtener estas acciones se utilizó una eficiente técnica de minería de datos. Con esta modificación se obtuvo iteración de valor basado en reglas de asociación.

En la segunda sección se presentó la propuesta de tesis: un nuevo algoritmo de iteración de valor priorizado que usa acciones reglamentadas, basado en el clásico algoritmo de Dijkstra. Este enfoque utiliza la función de valor actual como métrica de prioridad, ordenando dinámicamente a los estados más prometedores, reduciendo así drásticamente el espacio de búsqueda del algoritmo de iteración de valor. Por ello es capaz de obtener la política óptima en MDP de ruta estocástica más corta, con múltiples estados meta y de inicio.

El algoritmo propuesto va un paso más adelante que los demás enfoques del estado del arte debido a que solamente calcula la prioridad de los predecesores de cada estado relevante, mientras que los demás calculan la prioridad del propio estado relevante.

Por otro lado, como este enfoque actualiza de forma asíncrona a todos los predecesores de cada estado prometedor, entonces en este mismo capítulo se propusieron dos variantes de él: una que actualiza a estos predecesores por el método de Jacobi sin tener en cuenta su orden; y otra que prioriza a dichas actualizaciones.

Con esta nueva estrategia se espera que este enfoque presente una reducción considerable de la complejidad del algoritmo de iteración de valor (que es el objetivo de esta tesis) para resolver, en tiempo tratable, problemas de considerable dimensionalidad.

Para ello, en el siguiente capítulo se preparará el ambiente en el que se probarán el enfoque propuesto y otros enfoques del estado del arte. Luego, en el Capítulo 6, se presentarán los resultados experimentales obtenidos en dicho ambiente, en una tarea compleja de rutas de navegación. Ahí mismo se calculará la complejidad de los algoritmos presentados en este capítulo y de los enfoques del estado del arte con los que se compararán.

Algoritmo 4. 4 PPIPVI (Prioritized Predecessors Improved Prioritized Value Iteration [Garcia-Hernandez, 2012b]).

$(R, S, A, L, P, \gamma, \varepsilon)$
$L = \{\ell_k | \ell_k = (s_k, s'_k, a_k, p_k), p_k = P(s'_k|s_k, a_k) \neq 0\}$ //es el conjunto de reglas de Apriori
$(\forall s \in S) U(s) \leftarrow M$ //M es un número arbitrario positivo y suficientemente grande
$(\forall s \in G) U(s) \leftarrow 0$ //G es el conjunto de estados meta y es un subconjunto de S
$(\forall s \in S) \pi(s) \leftarrow$ indefinido
$(\forall s \in G) \pi(s) =$ arbitrario
$(\forall s \in G)$entraEnCola$(s, U(s))$
mientras $(\neg$colaVacia$())$ **hacer**
  $s \leftarrow$ colaArroja() //prioriza a los predecesores del estado s por su función de valor
  actualizaPred(s)
fin mientras
regresa π

actualizaPred(s)
para todo $(y, a) \in$ pred(s) **hacer**
  $U'(y) \leftarrow U(y)$
  $$U(y) = \max_a \left\{ R(y,a) + \gamma \sum_{\forall (s_k = y, s'_k, a_k = a, p_k) \in L} p_k U'(s'_k) \right\}$$
  $$\pi(y) \leftarrow \arg\max_a \left\{ R(y,a) + \gamma \sum_{\forall (s_k = y, s'_k, a_k = a, p_k) \in L} p_k U'(s'_k) \right\}$$
  si $(|U(y) - U'(y)| > \varepsilon)$ **entonces**
    decreceColaPrioridad$(y, U(y))$
  fin si
fin para
regresa

Capítulo 5

Preparación del Ambiente de Prueba

En el anterior capítulo se presentó el enfoque propuesto en esta tesis para simplificar a los procesos de decisión de Markov (MDP por sus siglas en inglés). Este enfoque aplica una reglamentación de acciones y un nuevo enfoque de priorización de estados en el algoritmo de iteración de valor. Al enfoque propuesto se le denominó IPVI (*Improved Prioritized Value Iteration*) [Garcia-Hernandez, 2012a]. Para verificar la robustez del enfoque propuesto, en este capítulo se presenta su implementación en el **simulador de planificación de movimientos robóticos (SPRM)** de Reyes *et al.* [Reyes, 2006b], en una tarea compleja de ruta estocástica más corta, la cual a continuación se describe.

5.1. El Dominio de Navegación Marítima

Para validar al enfoque propuesto se seleccionó un problema de ruta estocástica más corta con bote de vela: el dominio de navegación marítima denominado *Sailing* [Vanderbei, 1996], debido a que este es un caso especial de MDP de **ruta estocástica más corta, con costes positivos y estados terminales absorbentes** [Bertsekas, 1995]. Este problema es modelado en forma estocástica para la búsqueda de la trayectoria más corta (la óptima), tomando en cuenta fluctuaciones aleatorias de la dirección del viento. Para ello se considera que un marinero desea navegar desde un punto A a otro B en un lago donde sopla viento desde diferente dirección en cada punto del mismo, en el tiempo más corto posible (por ejemplo en una competencia).

Se considera un **proceso finito** debido a que el marinero debe alcanzar cierto número de etapas en su recorrido. Para crear un modelo simple de tal situación, se supone que el lago está representado por un espacio bidimensional, donde los puntos a los que se puede trasladar el bote de vela son nodos en una

malla (tablero) de n dimensiones.

Cada ruta desde un estado inicial A hasta un estado meta B, consiste de una secuencia de tramos en la malla. Se supone que el velero puede moverse hacia sus ocho puntos vecinos contiguos: $N, S, E, W, SE, NE, SW, NW$ que corresponden a los puntos cardinales y sus puntos intermedios. Por otro lado, la velocidad del velero depende del ángulo relativo entre la **dirección** (*tack*) de la "proa" (adelante) del velero y la dirección desde donde sopla el viento.

Por ejemplo, el velero no puede navegar directamente en contra del viento (*into the wind*), es decir que requerirá de un tiempo infinito. Sin embargo, si el viento golpea a 45° la "proa" del velero, se dice que navega "viento arriba" (*upwind*). En tal caso, le llevará al velero 4 minutos en navegar hasta un punto contiguo. Pero si la "proa" esta dirigida a 90° de la dirección del viento, entonces el velero se moverá más rápidamente y podrá alcanzar al punto vecino en solo 3 minutos. Aquí se dice que el velero se encuentra en dirección "viento cruzado" (*crosswind*).

Por otro lado, si el viento azota al velero en forma inclinada a la "popa" (detrás), entonces se dice que el velero está navegando en dirección "viento abajo" (*downwind*), lo cual le tomará tan solo 2 minutos llegar al siguiente vecino. Por último, si el velero recibe al viento directamente por la "popa", entonces el velero permanecerá en la misma dirección (*away*) que el viento y en solamente 1 minuto se podrá cambiar a un punto vecino en la misma dirección.

También se debe considerar que cuando el viento azota al velero por su derecha, sea en forma inclinada o perpendicular, se dice que está en dirección "a estribor" (*starboard*). En cambio, si el viento azota por la izquierda al velero, entonces se dice que está en dirección "a babor" (*port*). Por ejemplo, si el velero está enfilado hacia el N y el viento sopla desde el NE, E o SE, entonces se dice que lo azota "a estribor". Mientras que si el viento lo golpea desde el NW, W o SW, entonces lo azota "a babor".

Cambiar de dirección de "babor" a "estribor" o *vice versa* es una tarea complicada. Los marineros expertos pueden ejecutarla con gran pericia, pero a los novatos se les puede voltear el velero con lamentables consecuencias. Aquí se supone que el marinero es experto y que le lleva 3 minutos de retardo (*delay*) cada cambio de dirección.

En cuanto al viento, se supone que tiene la tendencia a cambiar de dirección en cada punto alcanzado por el velero. Por otro lado, el viento solo puede cambiar de dirección lateralmente a 45° (izquierda o derecha) o puede continuar con su anterior dirección, respondiendo al conjunto de probabilidades para cada uno de estos escenarios. Por lo que la estrategia óptima dependerá de cuatro factores independientes en cada punto del lago, que son los siguientes:

(a) latitud,

(b) longitud,

(c) dirección del viento en el siguiente punto a alcanzar,

(d) dirección relativa velero-viento.

Por último, sólo por simplicidad este dominio supone que la intensidad o velocidad del viento es constante.

5.2. Modelado del Dominio como Proceso de Decisión de Markov

Para modelar el dominio de navegación marítima como un MDP, se utilizó un espacio bidimensional de puntos (x, y), donde $x, y = 0, 1, \ldots, n$ que representa el lago donde se simula la navegación de un velero. Cabe señalar que se puede usar un espacio rectangular simplemente variando los límites a $x = 0, 1, \ldots, n, y = 0, 1, \ldots, m$. La parte superior del espacio corresponderá al N, la inferior al S, la derecha al E y la izquierda al W. Las coordenadas de los bordes inferior, superior, izquierdo y derecho (que corresponden a las playas) del lago son (x, 0), (x, n), (0, y) y (n, y), respectivamente.

Por otro lado, si el velero llega a la playa quedará atracado (debido a que son **estados absorbentes** [Blackwell, 1965] por tener recompensa cero), por lo que los únicos puntos donde podrá navegar el velero tendrán por coordenadas en $x, y = 1, \ldots, (n-1)$. Así, para una posición (x, y) en el lago, se supone que el velero puede navegar a un punto vecino en una de las siguientes ocho direcciones:

- N a $(x, y + 1)$
- S a $(x, y - 1)$
- E a $(x + 1, y)$
- W a $(x - 1, y)$
- NE a $(x + 1, y + 1)$
- SE a $(x + 1, y - 1)$
- NW a $(x - 1, y + 1)$
- SW a $(x - 1, y - 1)$

También se supone que el viento sopla desde una de las ocho direcciones arriba enlistadas y que su dirección es constante mientras el velero se mueve de un punto a otro en el lago, pero una vez que el velero ha alcanzado la

de w a w'	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			
SE				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

Tabla 5.1: Probabilidades de cambio de dirección del viento, $p(w, w')$. Los espacios en blanco corresponden a probabilidad cero.

valor	dirección (tack)
0	N
1	NE
2	E
3	SE
4	S
5	SW
6	W
7	NW

Tabla 5.2: Valores correspondientes a la dirección del viento o del velero.

nueva posición, ahí mismo el viento cambiará de dirección en forma aleatoria, de acuerdo con la tabla de probabilidades, $p(w, w')$ (vea Tabla 5.1).

Por otro lado, el marinero puede ver (por las pequeñas olas que provoca el viento en la superficie del agua) la nueva dirección con que el viento azotará al velero al llegar al punto vecino. Ahora bien, como en cada punto de la malla cuadrada (de lado con magnitud n) el velero puede recibir al viento desde cualquiera de las ocho direcciones de los puntos cardinales y sus combinaciones y puede estar en tres direcciones relativas velero-viento, entonces el número total de estados, n_s, se puede calcular de la siguiente manera:

$$n_s = (8)(3)n^2 = 24n^2 \tag{5.1}$$

Para su codificación, tanto la dirección de la "proa" del velero como la dirección desde donde sopla el viento, se usaron los valores de la Tabla 5.2.

Si w denota la dirección desde donde sopla el viento y h denota la dirección actual del velero, entonces la dirección por donde golpea el viento al velero se calcula de la siguiente manera:

si $tack(w,h)es$	entonces $time(w,h)$, minutos:
0	*into the wind* $= \infty$
1 ó 7	*upwind* $= 4$
2 ó 6	*crosswind* $= 3$
3 ó 5	*downwind* $= 2$
4	*away from wind* $= 1$

Tabla 5.3: Valores del tiempo requerido por el velero para navegar un tramo en la malla.

$$tack(w,h) = (w-h) mod 8 \tag{5.2}$$

Si este parámetro toma el valor de $1, 2$ o de 3, entonces corresponde "a babor". Si toma el valor de $5, 6$ o de 7, entonces corresponde "a estribor". En cambio, si toma valor de 0 o de 4, entonces el velero está siendo golpeado por el viento por la "proa" o por la "popa".

Cabe aclarar que, como la **función de valor es el tiempo** (pues en la competencia el marinero desea alcanzar el estado meta en el menor tiempo posible), entonces dicha función se minimiza en el cálculo de la política óptima.

Por otro lado, el tiempo es discretizado de modo que el marinero solamente tenga que tomar una decisión al arribar a una nueva posición. Ahí el marinero conocerá sus coordenadas, la dirección relativa t viento-velero ("a babor" $t = 0$, "a estribor" $t = 1$, por la "proa" o por la "popa" $t = 2$) y la nueva posición del viento w. Con esta información el marinero decidirá hacia dónde deberá enfilar su velero para lograr la ruta óptima.

La función $delay(t, t')$ denota el retardo correspondiente al cambio de dirección del velero (desde "estribor" hacia "babor" o *viceversa*. Si $t = t'$ ó $t' = 2$ (viento por detrás o por delante del velero), entonces se tiene que $delay(t, t') = 0$ y si existe un cambio de dirección relativa viento-velero, entonces se tiene que $delay(t, t') = 3$. Ahí mismo, $time(w, h)$ es el tiempo que toma alcanzar un punto vecino al estado en evaluación, dado que el velero está siendo enfilado en dirección h y el viento está golpeando al velero en dirección w.

El tiempo requerido para navegar una unidad de longitud (desde un punto dado hasta otro contiguo en el mismo eje) tomará uno de los cinco valores siguientes, dependiendo del valor de $tack(w, h)$, como se puede observar en la Tabla 5.3.

Generalmente *into the wind* es infinito, pues el velero no puede navegar directamente en contra del viento. Para fines de implementación basta establecer un valor suficientemente alto en comparación con los otros valores de la segunda

columna de la tabla anterior. Con estos valores se encuentra que:

$$time(w, h) = tack(w, h) \tag{5.3}$$

Esos valores corresponden a direcciones N, S, W o E de la "proa" del velero, h. Pero si el bote se enfila en movimientos diagonales en la malla (NE, SE, SW o NW), entonces $time$ deberá se multiplicado por $\sqrt{2}$ para entregar la hipotenusa o diagonal entre nodos. Por otro lado, de acuerdo con el nodo que ocupe el velero en el tablero, el estado actual del velero puede ser calculado de la siguiente manera:

$$s = (8nx + 8y + w)(3) + t \tag{5.4}$$

La anterior ecuación resulta de la combinación de las ocho direcciones del viento w (vea Tabla 5.2) con los tres valores de t. Entonces $p(w, w')$ es la probabilidad de que la nueva dirección del viento sea w' dado que la dirección del viento actual es w. Por último, $\triangle x_k$ es el cambio en la coordenada x dado que la dirección del velero es h y $\triangle y_k$ es el cambio en la coordenada y dado que la dirección del velero es h.

Por otro lado, el dominio descrito utiliza un enfoque de programación dinámica que aquí se denominará **VDP** *(Vanderbei's Dynamic Programming)*, vea Figura 5.1. Este enfoque iterativo minimiza al tiempo esperado para alcanzar al estado meta. Para ello a partir de una aplicación que presenta el dominio descrito se creó la clase *Sail.java* y se implementó en *SailingStrategies*, que requirió la librería *myutil*.

También se estableció que la probabilidad de transición del viento es cero si se encuentra en alguna playa (es decir que quedó atracado) o en el punto final de la trayectoria (estado meta). Este algoritmo es asíncrono y se observó que esto lo fortalece ante problemas hasta con 10^{20} estados.

Para la ejecución del algoritmo VDP, con *Sail.java* se abre la interfaz correspondiente para *Optimal Sailing Strategies*, que con el botón "Calculate" ejecuta una corrida del mismo. En dicha interfaz se presentan los campos de la matriz de probabilidades de transición de la dirección del viento con sus valores predeterminados, para su posible modificación. Además, se presentan los campos con valores predeterminados del tamaño del lago, las coordenadas del punto inicial y del punto meta del velero, que de igual manera pueden ser modificadas.

Cabe señalar que para indicar al programa *Sail.java* el tamaño del lago, se debe establecer un punto menos del requerido por el problema, por ejemplo, para el tamaño 16 del tablero (en este caso el lago) se debe escribir como 15 y como $(14, 14)$ las coordenadas del punto meta. Lo anterior es debido a que este algoritmo sólo cuenta los segmentos requeridos en el lago para conseguir

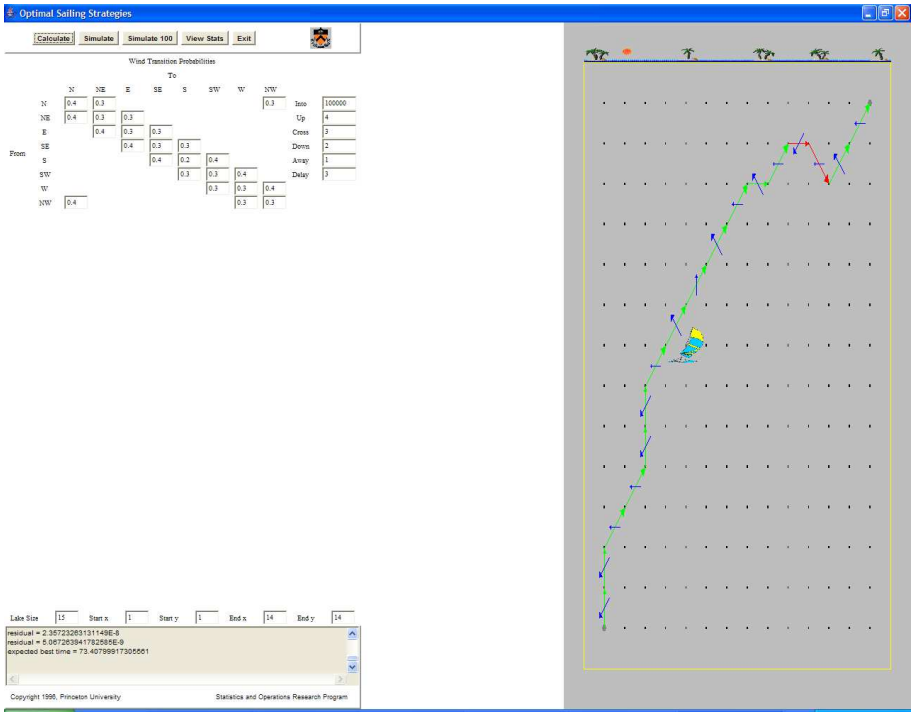

Figura 5.1: Interfaz del dominio de navegación marítima *Sailing* (Cortesía [Vanderbei, 2008]).

el número de nodos deseados. Por último, esta interfaz utiliza un generador aleatorio de números para la dirección del viento.

El algoritmo VDP se basa en la siguiente ecuación para calcular el tiempo mínimo esperado para que el velero se desplace de un punto a otro dentro del tablero:

$$v(x, y, t, w') = delay(t, t') + time(w, h) + \sum_{w' \in \{0,\ldots,7\}} p(w, w') v(x + \Delta x_h, y + \Delta y_h, t', w') \quad (5.5)$$

donde t' es la nueva dirección relativa viento-velero asociada con la nueva dirección del viento w' y con la nueva dirección del velero h'. Este enfoque iterativa inicia dando un valor muy grande a cada variable, excepto al estado meta, al cual le otorga valor cero. Como este enfoque minimiza el tiempo esperado entonces tenderá a acercar al bote de vela al estado meta. Durante el proceso iterativo, este enfqoue va mejorando el tiempo al reemplazar su valor actual con el calculado en la anterior iteración, como se puede observar en la anterior ecuación.

De modo que, para establecer el tiempo requerido para que el velero navegue *into the wind* el cual es infinito, se utilizó un valor considerablemente grande (por ejemplo 100000 minutos) comparado con el resto de los valores, que van desde uno hasta cuatro minutos. También se pudo determinar que en este enfoque el número de estados, n_s, se relaciona con el tamaño del tablero, n, de la siguiente manera:

$$n_s = 24\,(n-2)^2 \tag{5.6}$$

Este enfoque es capaz de resolver rápidamente problemas con una considerable cantidad de estados. Por lo que, en el Capítulo 5, se comparará su desempeño con el tenido por el enfoque que se propone en esta tesis y por otros enfoques del estado del arte.

5.3. Obtención de la Reglamentación de Acciones

De la herramienta de **código libre** WEKA versión 3.5.6 [Witten, 2005] se obtuvo la clase *Apriori.java*, con la que se calcula la reglamentación de acciones correspondiente al entorno de un autómata. Para probar esta herramienta se creó la clase *EjemplodeApriori.java* en el SPRM (simulador de planificación citado al inicio de este capítulo). De la misma herramienta se obtuvo la clase *AssociationRules.java* y se implementó en el simulador SPRM. Después se probó con el problema *JugarGolf* [Quinlan, 1993]

Por otro lado, se observó que WEKA solo entrega reglas hasta $(n-1)$ atributos, por lo que se diseñó otra clase: *AprioriAllRules.java*, la cual desciende o hereda *de Apriori.java*, basándose en la clase *BuildAssociation.java* de la misma caja de herramientas WEKA. A esta última clase se le volvió a codificar de modo que incluya al último conjunto de datos en el barrido que efectúa, esto se logró modificando al método *ToString*.

Posteriormente se diseñó la clase *ItemSetAll.java* para que sustituyera a *ItemSet.java*, de modo que incluya a aquellas reglas con soporte diferente de cero. Así, la clase *AprioriAllRules.java* pudo ser probada en *EjemplodeApriori.java*, la cual entregó el 100 % de reglas con acción posicionada a la derecha. Posteriormente, la clase modificada *Apriori.java* se insertó en la ecuación de Bellman de la superclase *MarkovDecisionProcesses.java* y de ahí resultó *RulesRelationalActions.java*, que desciende de dicha superclase. En la clase heredada *modelo.lenght* es el número de estados y *modelo[0][0].lenght* es el número de acciones.

Cabe mencionar que los datos que proporciona SPRM corresponden a las coordenadas de un autómata en el espacio bidimensional (vea **Anexo 1)**. Se

modificó para que entregue los atributos en el siguiente orden: estado inicial, estado alcanzado y acción ejecutada. Por otro lado, los estados se enumeran iniciando con $s = 0$ en la esquina inferior izquierda del espacio bidimensional, siguiendo la línea vertical y de acuerdo a la segmentación establecida.

Cabe señalar que se tiene la siguiente restricción: **el robot no podrá efectuar saltos de estado, en consecuencia sólo se podrá mover a estados contiguos**. Una vez obtenidas las reglas, se procedió a insertarlas en el algoritmo de iteración de valor contenido en el simulador mencionado, con objeto de que solamente calcule sobre esas reglas (que son las únicas acciones ejecutables en cada estado), como se estudió en el Capítulo 3.

5.4. Implementación de Algoritmos

En esta sección se presenta la implementación de los algoritmos estudiados en el Capítulo 4, en el ambiente de prueba descrito al inicio de este capítulo, con el objeto de resolver problemas complejos de ruta estocástica más corta. Cabe señalar que para ello se utilizó el lenguaje *Java* por ser independiente de la plataforma en que se encuentre, que ofrece máxima portabilidad y seguridad y que tiene mínimas dependencias de la implementación.

5.4.1. Iteración de Valor con Reglamentación de Acciones

Con el objeto de implementar a ARVI y variantes aceleradas, se hicieron adaptaciones a la interfaz "Ambiente de Navegación" del citado simulador SPRM y en la clase *MarkovDecisionProcess.java* (aquí denominado VI), así como en la clase *RulesRelationalActions.java*, de modo que la función de valor fuera el tiempo requerido para que el bote de vela se traslade entre dos puntos del lago en el mínimo tiempo posible (por ejemplo en una competencia). Para ello se generó el paquete *SailingStrategies*, en el que se añadió el paquete *robotica*, se le cambió el nombre y se le eliminó la exploración aleatoria del espacio de estados. Posteriormente, se le añadió la clase *Estado.java*, derogándole "sensores".

Otra clase añadida al paquete en cuestión es *AreaNavegacion.java*, la cual tuvo que ser modificada para que el velero se mueva entre nodos, no entre casillas. Se tuvo que asignar nodo inicial y nodo final del velero. De igual manera se añadieron las clases *Celda.java*, *Graficas.java*, *Movimiento.java* (que se modificó a *Movimientoc.java*), así como *Punto.java*, todo con el objeto de adaptarlo a los requerimientos del movimiento del velero a través de **nodos** en lugar del movimiento a través de **casillas**.

También se le añadió la clase *GridFrame.java* para modificar la interfaz de usuario "Ambiente de Navegación" del simulador SPRM. Se reconfiguró dicha

interfaz de tal manera que manejara información pertinente al dominio que nos ocupa. Ahí se puede apreciar que el tamaño del tablero corresponde a un punto menos del requerido por el problema, esto es debido a que estaba diseñado para contar casillas, no para contar nodos, por lo que se le hizo la adaptación.

Por ejemplo, para un tablero de tamaño 16, se debe establecer como medida 1500 cm de ancho y de alto, para casillas de 100 cm de medida. De esta manera resultan 15 casillas, pero como lo que interesa conocer son los nodos, entonces estos resultan 16 (contando a ambos lados de cada casilla). El **tamaño del tablero** se calculó como nodos efectivos (estados) más dos unidades, que corresponden a los nodos de las indeseables playas. Es decir, si el tamaño de tablero es 200, significa que los nodos laterales son 198 al restarle los dos nodos correspondientes a las playas. Los 198 nodos corresponden a estados donde efectivamente se puede mover el velero. Por lo que el número de estados posibles resulta entonces de 940896.

Por otro lado, en la misma interfaz se presentan campos con valores predeterminados de los datos de entrada como son el factor de descuento γ, el máximo número de iteraciones (*numit*), el margen de error permitido ε y los tiempos requeridos por el velero para moverse de un punto a otro teniendo al viento en cinco posiciones posibles: *into, up, cross, down, away*. También se presenta el retardo *delay* requerido para moverse desde babor a estribor o *vice versa*.

Otro dato de entrada requerido en la misma interfaz es la matriz de probabilidades de transición de la dirección del viento. Esta interfaz trae la opción de planificador que se desea utilizar: botón "RulesMDP" (ARVI y sus variantes) o botón "MDP" (que corresponde al VI previamente contenido en el simulador SPRM). Cabe aclarar que estos valores predeterminados pueden ser modificados en la misma interfaz.

Para su ejecución, basta pinchar "Run" para que muestre la interfaz "Sistema de Planificación con Incertidumbre", de ahí se selecciona el menú *Simulation* y posteriormente se pincha *Sailing Strategies*. Por último, para una corrida solamente se debe pinchar el botón "Simular".

En la clase *GridFrame.java* se le otorgó la capacidad de que llamara y leyera a dos clases de archivos: *data* y *text*. Se derogó "ruido" y se insertó la matriz de probabilidades de transición del viento *p(w,w')*, desde la dirección w hasta la nueva dirección w'.

También se modificó al estado en que se encuentra el velero, de modo que sea dado en función de sus coordenadas, de la dirección del viento y de la dirección relativa viento-velero, resultando $s[x][y][w][t]$. Se observó que si se usa $0 < \gamma < 1$, entonces no se logra la convergencia, en cambio si se aplica $\gamma = 1$ entonces sí se obtiene. Esto es debido a que las orillas del lago (playas) y el nodo meta del velero se consideran **estados absorbentes** (con recompensa ce-

ro [Blackwell, 1965] y, por lo tanto, con 100 % de probabilidad de permanecer atracado en el mismo estado), pues es un **proceso finito**.

De igual forma se añadió la superclase *MarkovDecisionProcess.java* y se le modificó la función recompensa (*reward*) requerida por la *ecuación de Bellman*, la cual se calcula de la siguiente manera:

$$R(s,a) = delay(t(s), tackt(w(s),a)) + time(w(s),a) \tag{5.7}$$

debido a que en este trabajo se requiere que la recompensa esté en función de dos variables: del estado en que se encuentra el velero s y de la acción ejecutada a sobre dicho estado. Esta última corresponde a la dirección de la "proa" del velero, h, resultando así $R[s][a]$.

Además, dado que la ecuación de Bellman maximiza a la utilidad esperada, entonces se estableció la recompensa con valor negativo, con el objeto de que entregue el **mínimo tiempo esperado**. Entonces la ecuación de Bellman se transforma en:

$$v_{t+1}(x,y,w,t) = min_{h=0,\ldots,7} \sum_{w'} \{p(w,w')v(x+\Delta x_h, y+\Delta y_h, w', tackt(w,h)) + delay(t, tackt(w,h)) + time(w,h)\} \tag{5.8}$$

donde *tackt* se refiere a la dirección relativa velero-viento en el nuevo nodo alcanzado. Por otro lado, ARVI y sus variantes corresponden a la clase *Rules-RelationalActions.java* con las opciones que presenta la interfaz "Ambiente de Navegación".

De ellas, ARVI y ARVI3 procesan en forma **síncrona** debido a que actualizan la función valor (utilidad) hasta que termina el barrido por el espacio de estados completo. Esto también sucede con el VI contenido en el mismo simulador.

En cuanto a las otras variantes, para darle mayor velocidad al proceso se modificó a la superclase *MarkovDecisionProcess.java* para que también procese en forma **asíncrona**, es decir, que cuando encuentre a la función de valor óptima la actualice inmediatamente, sin esperar a que termine el barrido antes mencionado, que son los algoritmos ARVI2, ARVI4, ARVI5 y ARVI6.

Por otro lado, la asincronía se obtiene calculando la nueva utilidad del estado en evaluación mediante la ecuación de Bellman y al valor obtenido se le resta la utilidad anterior del mismo estado. Si la diferencia habida entre ambas es mayor a la máxima permitida (maxError), entonces la utilidad previa del mismo estado adquirirá el valor de la nueva utilidad.

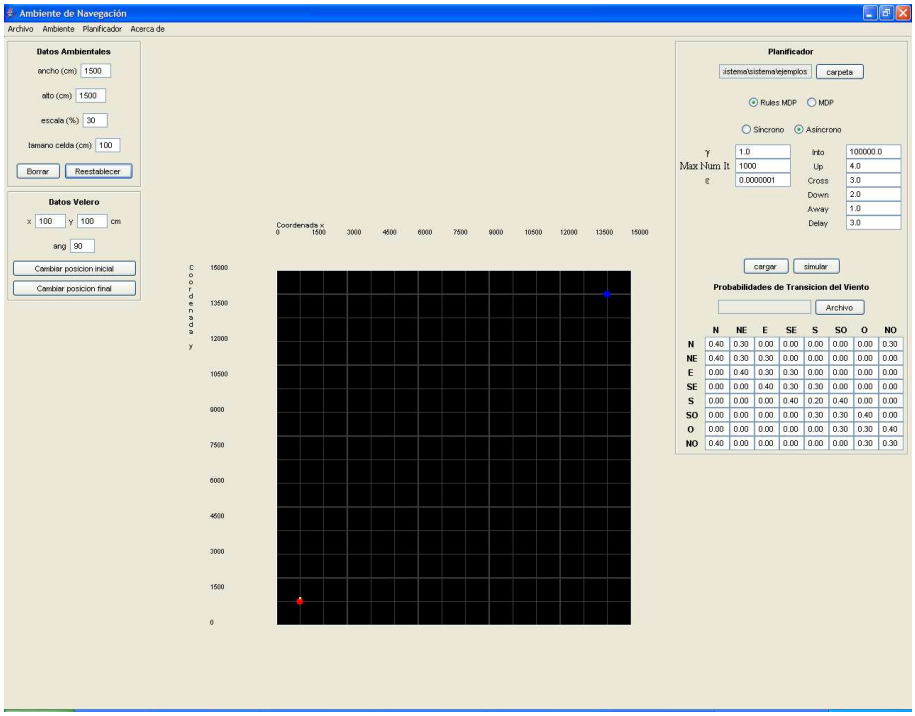

Figura 5.2: Interfaz "Ambiente de Navegación" para VI clásico, ARVI y ARVI2.

En la Figura 5.2 se presenta la interfaz "Ambiente de Navegación" que entrega el simulador SPRM, donde se puede seleccionar a "RulesMDP" (para opciones "Síncrono" o "Asíncrono", pero sin priorización de estados: **ARVI** y **ARVI2**, respectivamente) o "MDP" (para el **VI** clásico). Ahí mismo se capturan el modelo de transición de estados y demás valores requeridos. Para seleccionar las diferentes modalidades se tuvo que modificar la clase *GridFrame.java*.

En la Figura 5.3 se presenta la interfaz "Sailing Strategies", donde se puede seleccionar "RulesMDP" (para opciones "Síncrono" o "Asíncrono", con la opción "Priorizar" estados y con la opción "Reordenamiento estático" de estados: **ARVI3**, **ARVI4** y **ARVI5**, respectivamente. También incluye la opción "MDP" (para el VI clásico). Ahí mismo se capturan el modelo de transición de estados y demás valores requeridos. De igual manera, para seleccionar las diferentes modalidades se modificó la clase *GridFrame.java.*.

En la misma figura, la opción "Priorizar" utilizó "No" para aquellos experimentos en que no existe métrica, es decir, que no selecciona a la función de valor máxima en la primera iteración (**H1**) ni al máximo valor absoluto del error de Bellman (**H2**) (vea ecuación (2.6)). **No tener métrica** significa que solo se actualizan los estados cuya función de valor cambia entre iteraciones sucesivas

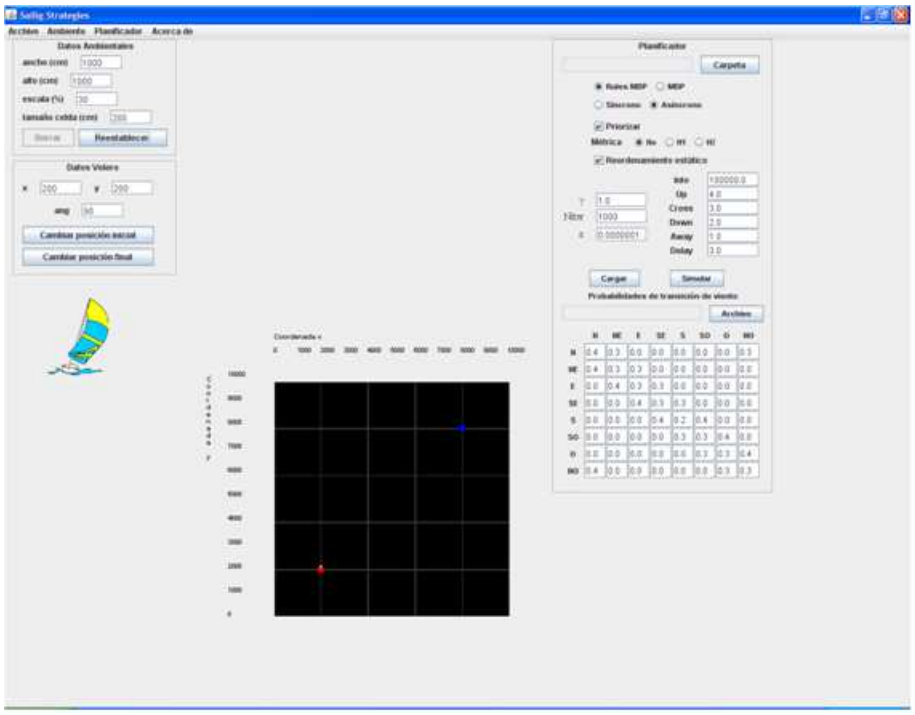

Figura 5.3: Interfaz "Sailing Strategies" para VI clásico y para ARVI y variantes).

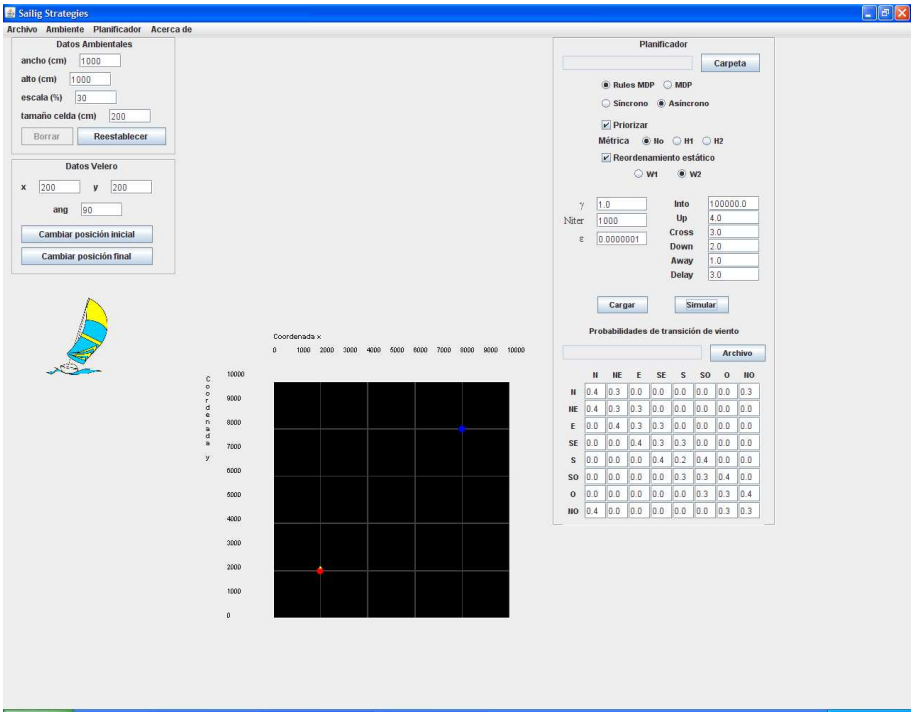

Figura 5.4: Interfaz "Sailing Strategies" para los algoritmos con reordenamiento de estados: ARVI5 y ARVI6.

y es mayor que el máximo error permitido.

Cabe aclarar que se decidió no usar las métricas **H1** y **H2** debido a que ambas presentaron, en experimentos previos, alto coste de inicio.

En la Figura 5.4 se presenta otra interfaz "Sailing Strategies" en la que, además de seleccionar "RulesMDP", asincronía y priorización, se puede seleccionar el tipo de reordenamiento de estados.

Este reordenamiento puede ser dado en forma decreciente de su máxima recompensa (**W1**, es decir ARVI5), o por su topología (W2, es decir ARVI6).

5.4.2. Iteración de Valor Priorizado con Reglamentación de Acciones

En el ambiente de planificación de movimientos robóticos descrito también se implementó a IPS [McMahan,2005] y a SIPS (que es IPS con acciones reglamentadas), respectivamente. Como la literatura reporta que IPS solamente converge

a la política óptima ante MDP deterministas, entonces en el siguiente capítulo se le adicionarán a SIPS técnicas de aceleración del estado del arte para obtener su convergencia [Garcia-Hernandez, 2011]. Cada técnica corresponderá a un método distinto dentro de la clase *McMahanRules.java* para evaluar y mejorar la política encontrada.

Para ello se modificó a *GridFrame2.java* de modo que ahora despliegue todas las versiones convergentes de SIPS, con el objeto de ser aplicadas sobre MDP basados en regla de asociación (opción "RulesMDP" en la esquina superior derecha de la interfaz).

Cabe aclarar que una vez que se ha seleccionado cualquier opción convergente de SIPS, simultáneamente quedan inhabilitadas las opciones "Actualización", "Priorizar", "Métrica" y "Reordenamiento estático" en esta interfaz de usuario. Estas variantes aceleradas de IPS se compararán en el siguiente capítulo con el enfoque propuesto en esta tesis.

También se implementó al enfoque propuesto (IPVI), y sus variantes JIPVI y PPIPVI (vea final del Capítulo 4), en el ambiente de planificación señalado al inicio de este capítulo. Se utilizó la misma interfaz "Sailing Strategies" que usan SIPS y sus variantes (vea Figura 5.5), donde *GridFrame2.java* despliega la opción "IPVI, Improved Prioritized Value Iteration". En dicha interfaz se observa que tiene la opción de cambiar los parámetros predeterminados como son la matriz de transición de estados, el umbral de paro del algoritmo, los valores del tiempo de ejecución de las diferentes tareas de navegación marítima, entre otros.

De igual manera, al seleccionar la opción "IPVI, Improved Prioritized Value Iteration" quedan inhabilitadas las opciones "Actualización", "Priorizar", "Métrica" y "Reordenamiento estático" en esta interfaz de usuario.

Por último, con el objeto de comparar el desempeño del enfoque propuesto y el de los demás enfoques implementados, en la siguiente sección se comparará experimentalmente el tiempo de solución y el número de actualizaciones requeridas por cada uno. Esto se hará en el ambiente de planificación robótica con la tarea compleja de ruta estocástica más corta descrita al inicio de este capítulo.

5.5. Conclusiones del Capítulo

En este capítulo se presentó la preparación del ambiente de prueba y para ello se seleccionó un problema de navegación marítima con bote de vela, debido a que este es un caso especial de MDP de ruta estocástica más corta, con costes positivos y estados terminales absorbentes. Este problema se modeló en forma

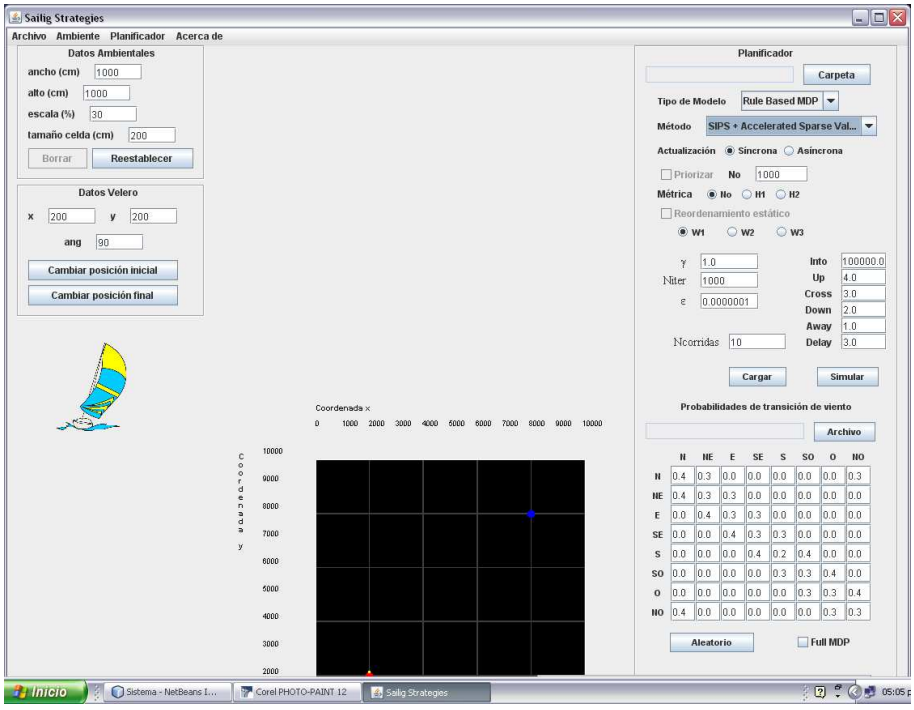

Figura 5.5: Interfaz "Optimal Sailing Strategies" para los algoritmos con priorización.

estocástica para la búsqueda de la trayectoria óptima (que se recorrerá en el tiempo más corto), tomando en cuenta fluctuaciones aleatorias de la dirección del viento. Por otro lado, es importante señalar que este caso se considera un proceso finito dado que el marinero debe alcanzar cierto número de etapas en su recorrido. Este ambiente de prueba se estableció en un simulador de planificación de movimientos robóticos, con objeto de comparar el desempeño del enfoque propuesto y de otros enfoques del estado del arte. Sus resultados experimentales se presentarán en el siguiente capítulo.

Capítulo 6

Resultados Experimentales

En el anterior capítulo se llevó a cabo la preparación del ambiente de prueba con el objeto de verificar la robustez del enfoque propuesto. En este capítulo se presentan los resultados experimentales obtenidos por este enfoque y por los otros enfoques estudiados en el Capítulo 4, al ejecutar una tarea compleja de ruta estocástica más corta de navegación marítima.

Para ello, en el simulador de planificación de movimientos robóticos (SPRM) de [Reyes, 2006b] se probarán los algoritmos que a continuación se enlistan:

- el clásico algoritmo de iteración de valor (VI) [Puterman, 1994],
- el algoritmo de iteración de valor con reglamentación de acciones (ARVI) [Garcia-Hernandez, 2009],
- variantes de ARVI aceleradas con estrategias de Wingate *et al.* [Wingate, 2005],
- el algoritmo de programación dinámica del dominio utilizado (VDP) [Vanderbei, 2008],
- el algoritmo IPS de McMahan *et al.* [McMahan, 2005a,b] con reglamentación de acciones (SIPS) [Garcia-Hernandez, 2011],
- variantes de SIPS aceleradas con estrategias de Wingate *et al.* [Wingate, 2005],
- el algoritmo iteración de valor topológico mejorado de Dibangoye *et al.* (iTVI) [Dibangoye, 2008],
- el algoritmo propuesto en esta tesis (IPVI) [Garcia-Hernandez, 2012a].
- IPVI con el método de Jacobi (JIPVI) [Garcia-Hernandez, 2012b].
- IPVI con priorización de predecesores (PPIPVI) [Garcia-Hernandez, 2012b].

Las variantes de ARVI y de SIPS que se probarán en este capítulo se describirán en las siguientes dos secciones.

Cabe destacar que del estado del arte se seleccionó al algoritmo iTVI de Dibangoye *et al.* [Dibangoye, 2008] debido a que este reporta mejor desempeño que los bien conocidos algoritmos LAO* de Hansen *et al.* [Hansen, 2001], LRTDP de Bonet *et al.* [Bonet, 2003b] y TVI de Dai *et al.* [Dai, 2007b].

Por último, se resolverán problemas de ruta estocástica más corta en el dominio de navegación marítima de Vanderbei [Vanderbei, 2008] descrito en el anterior capítulo. En el **Anexo 2** se presenta la metodología de uso de este simulador.

Posteriormente se calculará la reducción del espacio de búsqueda de acciones y de estados de cada uno de estos algoritmos. Cabe señalar que todos los algoritmos fueron implementados en *Java* por ser este un lenguaje de programación independiente de la plataforma en que se encuentre, que ofrece máxima portabilidad y seguridad y que tiene mínimas dependencias de la implementación.

También es importante mencionar que las pruebas se hicieron en un ordenador con procesador Intel Pentium D, con velocidad de 2.66 GHz con 2 GB de RAM. A continuación se presentan los resultados obtenidos por estos algoritmos, enfatizando en la reducción del espacio de búsqueda logrado por cada uno de ellos.

6.1. Resultados de Iteración de Valor basado en Reglamentación de Acciones

Como datos de entrada se establecieron los siguientes valores: $\gamma = 1$, $\varepsilon = 1 \times 10^{-7}$ y $numit = 1000$, que son el factor de descuento, el umbral predeterminado de paro en las aproximaciones y el número máximo de iteraciones, respectivamente.

Cabe señalar que **se utilizó $\gamma = 1$ debido a que se está trabajando con MDP descontados**, donde la convergencia no está garantizada por el clásico **teorema de Banach del Punto Fijo** [Kirk, 2001] y el rango del número de iteraciones (vea ecuación (2.7)) nunca es rebasado [Blackwell, 1965]. Afortunadamente, la presencia de **estados absorbentes** (estados con nula recompensa y con 100 % de probabilidad de permanecer atracado en el mismo estado) como son las playas, pudo llevar al algoritmo a la convergencia [Hinderer, 2003].

Es importante señalar que en los problemas planteados, el tamaño del lago varió desde 6×6 hasta 260×260 correspondiendo a 384 estados y 1597536

estados, respectivamente. También que se repitió cada experimento diez veces y que se calculó su media aritmética, así como su desviación estándar.

Por otro lado, en todos los problemas se supuso que el marinero deseaba mover el velero entre dos puntos inmediatos a las orillas y opuestos en diagonal en el lago, partiendo desde el punto $(1, 1)$ y terminando, en consecuencia, en el punto $(n-2, n-2)$. También se supuso que el velero inicia con la "proa" enfilada hacia el N y que la dirección desde donde sopla el viento inicialmente es el NE, resultando de esta manera $tack(w, h) = 1$.

Cabe aclarar que durante la ejecución del clásico algoritmo de iteración de valor después de 2904 estados se agotó la memoria asignada en el equipo descrito. Un aspecto muy importante a enfatizar es que **todos los algoritmos aquí probados entregaron el plan óptimo**, lo que se puede verificar observando el tiempo esperado para alcanzar el punto B (meta), que todos reportan en sus ejecuciones.

Por otro lado, del estado del arte se extrajeron cinco variantes de iteración de valor: con actualización asíncrona [Puterman, 2005]; con priorización de estados que cambian su función de valor entre iteraciones sucesivas, con ordenamiento estático de estados priorizados por su máxima recompensa y con ordenamiento topológico de [Wingate, 2005]. **A estas variantes se les aplicó reglamentación de acciones** con el objeto de compararlas posteriormente con el enfoque propuesto.

A la primer variante se le llamó **ARVI2**, la cual aplica sólo una técnica de aceleración: actualización asíncrona de Gauss-Seidel [Puterman, 2005] sobre la función de valor.

A la segunda variante se le estableció como **ARVI3**, la cual también aplica sólo una técnica de aceleración: priorización de estados [Wingate, 2005] que actualiza a aquellos estados (así como a sus estados vecinos) cuyas funciones de valor cambiaron en la anterior iteración. Esto tiene el efecto de enfocar los cálculos en las regiones del problema que se visualizan como de máxima utilidad y, simultáneamente, evita actualizaciones de información inútil. En este caso, el orden de evaluación de los estados no es modificado.

A la tercer variante se le denominó **ARVI4**, la cual es una combinación de ARVI2 y ARVI3, por lo que actualiza asíncronamente sólo a aquellos estados (así como a sus vecinos) cuya función de valor cambió en iteraciones sucesivas [Wingate, 2005].

A la cuarta variante se le llamó **ARVI5**, la cual utiliza los mismas técnicas de aceleración que ARVI4, pero además aplica el reordenamiento estático de estados de Wingate *et al.* [Wingate, 2005] en forma decreciente de máxima recompensa. Cabe señalar que se utilizó este tipo de reordenamiento debido a que

el uso de una cola de prioridad para todos los estados del modelo puede resultar en un gasto excesivo, por lo que en lugar de un buen ordenamiento dinámico es más recomendable usar un buen ordenamiento estático. Este concepto consiste en reordenar los estados una sola vez (durante el inicio) de modo que, para cada pasada (barrida), estos estados sean actualizados en un orden aproximadamente óptimo. En este caso el ordenamiento es efectuado usando el popular algoritmo *quicksort* (método *sort* de la clase *Array* de Java), cuya complejidad es $O(n \log n)$. Cabe destacar que el reordenamiento de variables es solamente efectivo cuando se usa actualización asíncrona [Wingate, 2005].

A la quinta variante se le denominó **ARVI6**, la cual utiliza las mismas técnicas de aceleración que ARVI5 pero, en lugar de aplicar el reordenamiento estático, ejecuta el ordenamiento topológico modificado de estados de Wingate *et al.* [Wingate, 2005]. Sin embargo, como en ARVI5, el uso de una cola de prioridad para todos los estados del modelo puede resultar un gasto excesivo.

Para casos especiales de MDP acíclicos, el uso de un arreglo topológico sobre los estados da paso a un óptimo ordenamiento de estados, con tiempo de solución lineal en el número de nodos y en el número de aristas del grafo. Desafortunadamente los problemas correspondientes al mundo real involucran MDP no deterministas, por lo que un ordenamiento topológico es prácticamente imposible por su alto coste de inicio. Sin embargo, para estos casos se ha reportado efectivo el enfoque de Wingate *et al.* [Wingate, 2005].

En la Figura 6.1 se muestran los tiempos de solución de ARVI y de sus variantes aceleradas, en función del número de estados. Ahí se puede apreciar que ARVI5 (que usa actualización asíncrona de estados que cambian su función de valor entre iteraciones sucesivas, así como un reordenamiento estático por su máxima recompensa de Wingate *et al.* [Wingate, 2005]) resulta significativamente más rápida que las demás variantes de ARVI [Garcia-Hernandez, 2011].

En la Tabla 6.1 se muestran los resultados obtenidos por los algoritmos presentados en la Figura 6.1, para problemas con 940896 estados. En la misma tabla el tiempo de solución relativo obtenido por cada algoritmo se calculó con respecto al obtenido por ARVI5 (por ser la variante más rápida de ARVI).

Ahí mismo se puede observar que ARVI5 resulta 2.5 veces más rápido que ARVI simple, 1.76 veces más rápido que ARVI2 (que usa actualización asíncrona de Gauss-Seidel) y 3.16 veces más rápido que VDP. También se puede observar que ARVI simple (que no incluye proceso de aceleración alguno) resulta aproximadamente 1.2 veces más rápido que VDP.

En la figura anterior se puede observar que las técnicas de actualización asíncrona y de priorización de estados que cambian su función de valor entre iteraciones sucesivas (ARVI2 y ARVI3 respectivamente), entregan resultados similares en tiempo de solución.

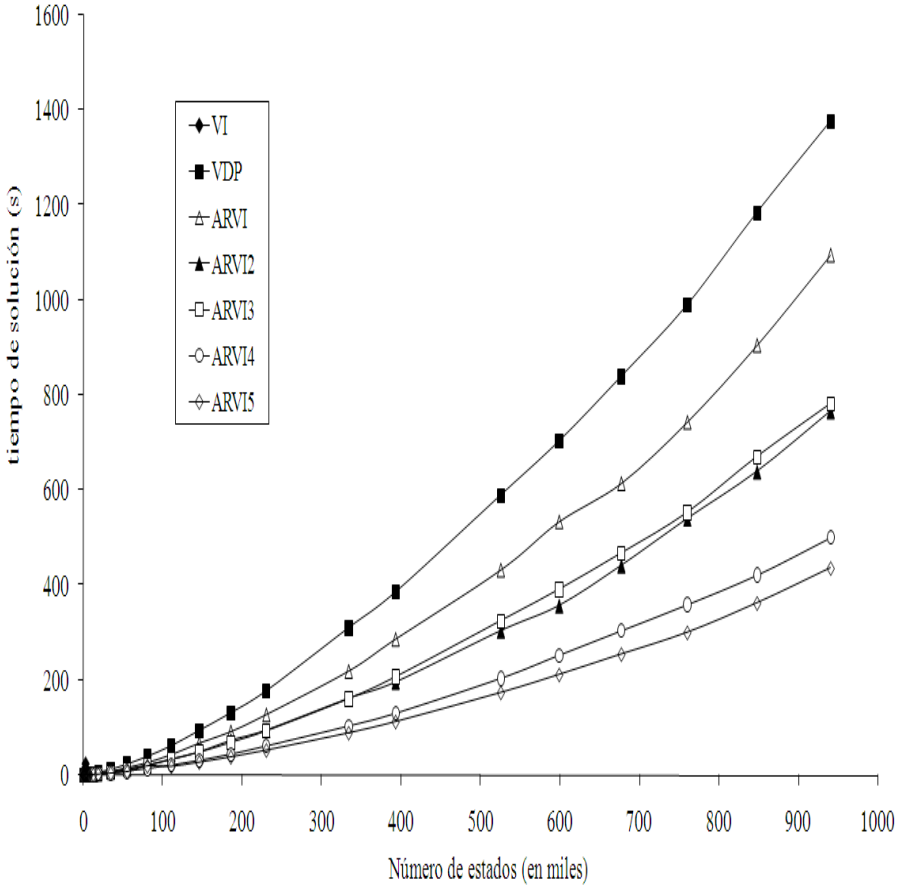

Figura 6.1: Tiempo de solución en función del número de estados de VDP, de VI, de ARVI y variantes aceleradas, excepto ARVI6 (VI agotó la memoria en 2904 estados).

Algoritmo	tiempo de solución (s)	tiempo de solución relativo
VDP	1376.6	3.16
ARVI	1095.7	2.50
ARVI2	766.1	1.76
ARVI3	782.3	1.79
ARVI4	499.2	1.14
ARVI5	436.3	1.00

Tabla 6.1: Resumen de resultados en términos del tiempo de solución de VDP, de ARVI y variantes aceleradas (excepto ARVI6) para 940896 estados.

Algoritmo	tiempo de solución (ms)	tiempo de solución relativo
VI	25112.4	262.1
VDP	385.8	4.0
ARVI	171.8	1.8
ARVI2	154.8	1.6
ARVI3	142.2	1.5
ARVI4	106.3	1.1
ARVI5	95.8	1.0

Tabla 6.2: Resumen de resultados en términos del tiempo de solución obtenido por VDP, ARVI y variantes aceleradas (excepto ARVI6) para 2904 estados.

En la Figura 6.2 se presenta un acercamiento a las curvas de la Figura 6.1, para 2904 estados, con el objeto de comparar el desempeño de VI con los demás algoritmos probados. Se puede observar que el tiempo de solución de VI muestra un crecimiento cuadrático con respecto al número de estados (de acuerdo con la ecuación (2.5)), mientras que los demás algoritmos probados exhiben lento crecimiento en el mismo. La Tabla 6.2 muestra que ARVI5 resulta 262 veces más rápido que VI.

En la misma figura se muestra la comparación del desempeño de los algoritmos que aplican reordenamiento de estados: ARVI5 (con máxima recompensa mediante el clásico algoritmo *quicksort*) y ARVI6 (con ordenamiento topológico modificado de Wingate *et al.* [Wingate, 2005].

Se puede observar que ARVI5 resulta significativamente más rápido que ARVI6, aunque ellos difieren solamente en la forma como son ordenados los estados. De esta manera se concluye que el algoritmo de ordenamiento topológico modificado presenta el peor tiempo de solución debido a su alto coste de inicio, requerido para encontrar el propio ordenamiento topológico.

Una alternativa a este algoritmo de ordenamiento topológico modificado es transformar el MDP (estocástico, por tanto cíclico) en uno acíclico mediante la resolución del "problema del conjunto de arcos de retroalimentación" (*feedback arc set problem*), usando algoritmos de complejidad lineal basados en búsqueda tipo "primero en profundidad". Desafortunadamente, el problema del conjunto de arcos de retroalimentación ha sido reportado como "NP-completo" por Garey *et al.* [Garey, 1990], esto significa que este problema es intratable computacionalmente por su magnitud.

Otra posibilidad para encontrar un ordenamiento topológico con bajo coste de inicio pudiera ser mediante un algoritmo que encuentre estados fuertemente conectados, como lo reportan Dai *et al.* [Dai, 2007a,b].

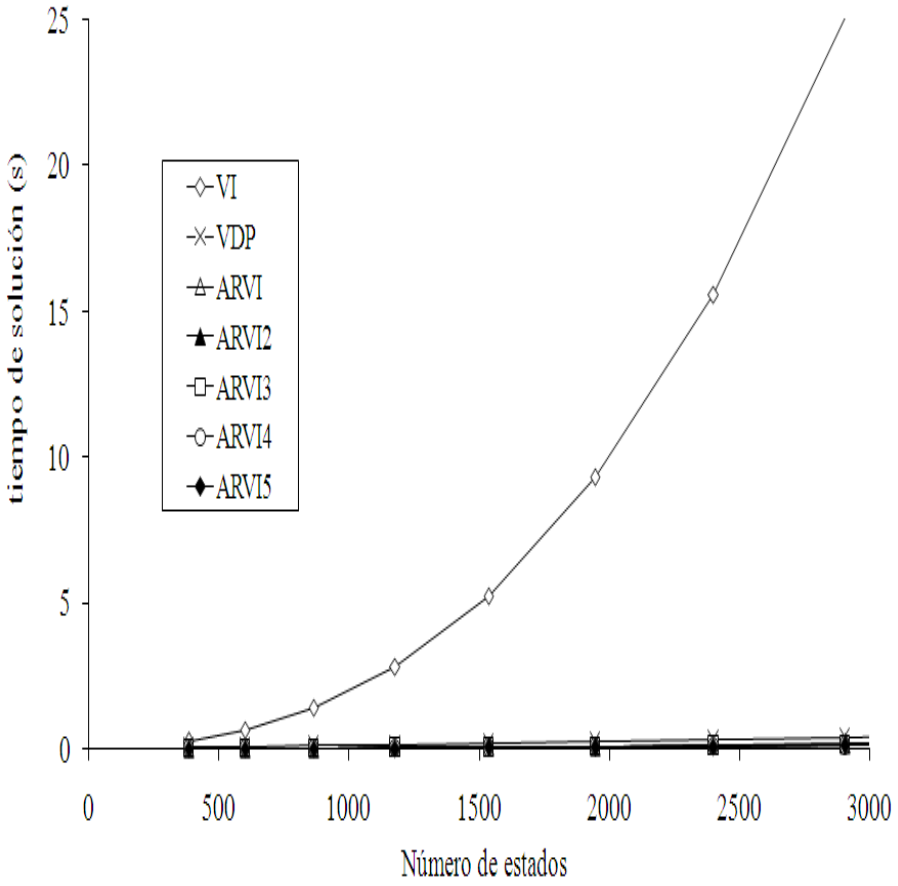

Figura 6.2: Un acercamiento de la Figura 6.1 con el objeto de comparar el desempeño de ARVI y variantes aceleradas con el algoritmo VI (todos empalman con el eje x, excepto VI).

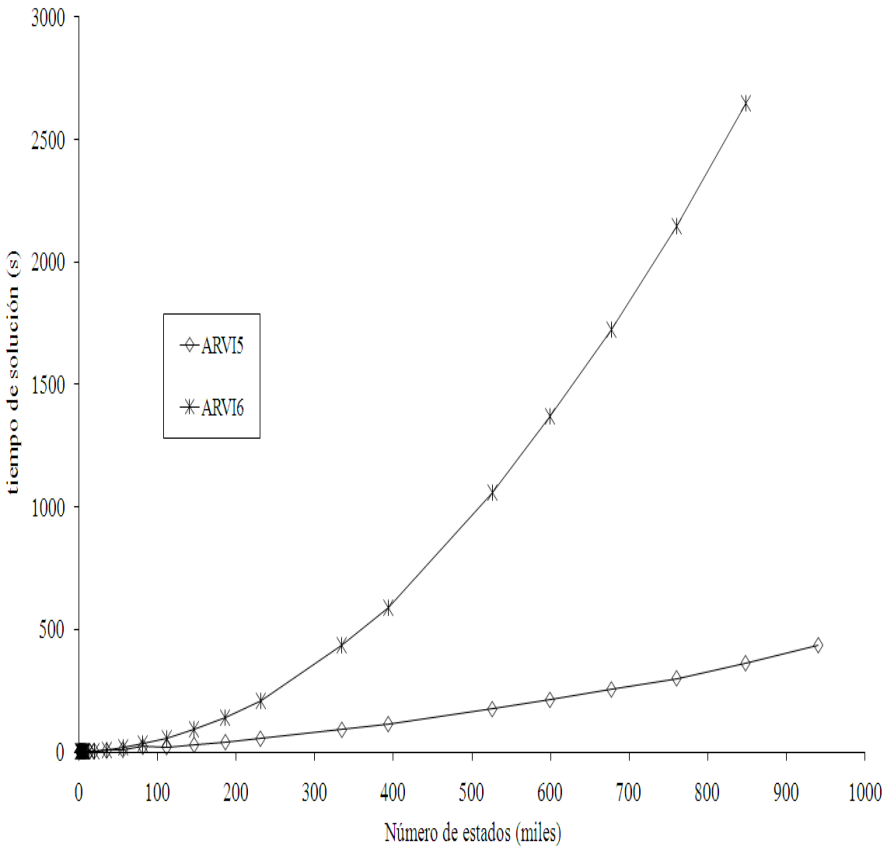

Figura 6.3: Tiempo de solución en función del número de estados de los algoritmos con ordenamiento de estados: ARVI5 y ARVI6.

Para verificar lo anterior se hicieron pruebas preliminares aplicándole a ARVI5 el clásico **algoritmo de Tarjan** que encuentra estados fuertemente conectados. Posteriormente se comparó su desempeño con ARVI5 puro y se encontró que este último continuó siendo más rápido en resolver MDP no deterministas.

Por otro lado se observó en las diferentes corridas que ARVI6 requirió de menos iteraciones que ARVI5. Es decir, ARVI6 requirió de mayor tiempo pero con menos actualizaciones que ARVI5. Esto se explica por el alto coste de inicio del ordenamiento topológico modificado usado por ARVI6, debido a que el dominio utilizado es un problema estocástico (no determinista). Por lo que se concluyó que el ordenamiento topológico modificado de Wingate *et al.* [Wingate, 2005] es recomendable para resolver MDP siempre y cuando se logre disminuir

Algoritmo	tiempo de solución (s)	tiempo de solución relativo
ARVI6	2650.6	7.3
ARVI5	362.5	1.0

Tabla 6.3: Resumen de resultados en términos del tiempo de solución de ARVI5 y ARVI6, para 848256 estados.

considerablemente su alto coste de inicio.

En la Tabla 6.3 se muestran los resultados observados en la Figura 6.2, para 848256 estados.

Por último, en el **Anexo 3** se presenta una ejecución de las variantes síncronas: ARVI y ARVI3. En el **Anexo 4** se muestra una ejecución de las variantes asíncronas: ARVI2, ARVI4, ARVI5 y VDP. En el **Anexo 5** se muestra una ejecución de las variantes con ordenamiento de estados: ARVI5 y ARVI6.

Se concluye que, con respecto a las variantes aceleradas del algoritmo de iteración de valor con reglamentación de acciones (ARVI) que se probaron en esta sección, ARVI5 fue la que requirió del menor tiempo de solución. Por lo que, hasta este punto de la experimentación, se puede concluir que la combinación de técnicas de aceleración del estado del arte, tales como actualización asíncrona de Gauss-Seidel y reordenamiento estático de estados priorizados por su máxima recompensa, resulta eficiente en resolver MDP estocásticos de considerables dimensiones.

6.2. Resultados de Iteración de Valor con Barrido Priorizado

Del estado del arte se seleccionó al enfoque de McMahan *et al.* [McMahan, 2005a,b] debido a que reportaron haber alcanzado la convergencia en tiempos de solución considerablemente cortos, aunque esto sucedió con MDP deterministas (acíclicos). Por esta razón en una primera instancia se probó la convergencia de su enfoque (IPS). Esto se hizo debido a que el dominio probado en esta tesis es del tipo estocástico. Para ello se le adicionó a IPS una reglamentación de acciones, obteniendo de esta manera al algoritmo **SIPS**. Por lo que SIPS debe presentar las mismas ventajas y desventajas que IPS.

Para probar su convergencia, se calculó el error como la norma infinita de la diferencia habida entre la función de valor entregada por SIPS y la función de valor obtenida por VI (vea Algoritmo 2.1), es decir $\|U - U^*\|_\infty$. También se utilizó el umbral $\varepsilon = 1 \times 10^{-7}$ como criterio de paro de los algoritmos.

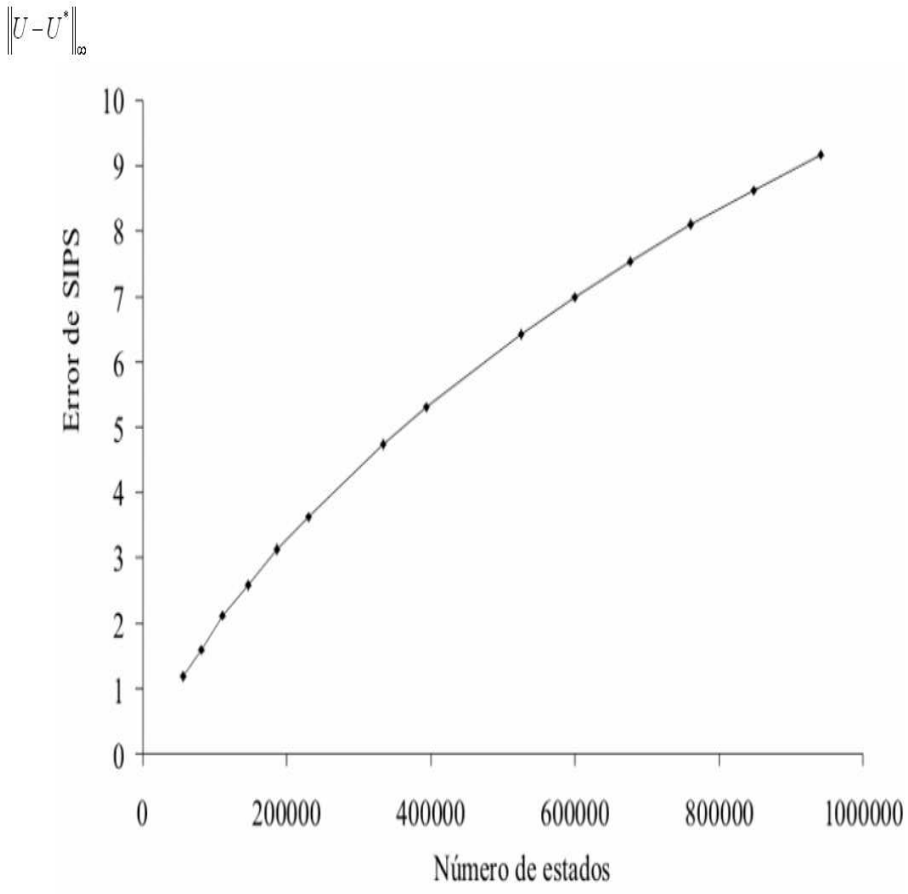

Figura 6.4: Error de SIPS en función del número de estados.

La Figura 6.4 muestra el error de la política obtenida por SIPS en comparación con la política óptima obtenida por VI, en función del número de estados. En esta figura se observa claramente una **tendencia incremental del error de SIPS cuando el número de estados aumenta**.

Cabe recordar que VI garantiza la convergencia a la política óptima. De ahí se infiere que, al menos en los problemas de ruta estocástica más corta (dominio utilizado), **SIPS entrega una solución subóptima**. Esto no es sorpresivo debido a que SIPS está basado en IPS, el cual fue diseñado para resolver problemas casi deterministas (es decir, con pocos ciclos), en cambio **el problema de navegación marítima *Sailing* es altamente cíclico**.

En una segunda instancia se investigó el tiempo que podría tomarse para ob-

tener la función de valor óptima **iniciando desde la política subóptima que entrega SIPS**, en un problema altamente cíclico como lo es el dominio mencionado [Garcia-Hernandez, 2011]. Para esto se le adicionaron a SIPS técnicas de aceleración de la convergencia tales como: actualización asíncrona de Gauss-Seidel [Puterman, 2005] obteniendo **SIPS+ARVI2**, actualización asíncrona de Gauss-Seidel de aquellos estados cuya función de valor cambia en la iteración previa [Wingate, 2005] obteniendo **SIPS+ARVI4**, ordenamiento estático de estados en forma decreciente de máxima recompensa [Wingate, 2005] obteniendo **SIPS+ARVI5**, iteración de política, obteniendo **SIPS+PI** e iteración de política modificada, obteniendo **SIPS+MPI**, ambos de [Puterman, 2005].

La Figura 6.5 muestra el tiempo de solución requerido por las variantes aceleradas de SIPS como una función del número de estados. En esta figura se puede observar que SIPS+ARVI2 y SIPS+ARVI4 requirieron aproximadamente del mismo tiempo de solución. Como ya se esperaba, SIPS+PI fue el algoritmo más lento, mientras que SIPS+ARVI5 resultó ser la variante más rápida, por razones obvias. El comportamiento de esta última combinación se debe a que utiliza tres tipos de técnicas de aceleración simultáneamente.

Por ejemplo, para 525696 estados SIPS+PI requirió 395.7 segundos, mientras que SIPS+ARVI5 necesitó 97.4 segundos; y para 1359456 estados, SIPS+PI requirió de 1580.6 segundos, mientras que SIPS+ARVI5 necesitó de 346.1 segundos para resolverlo. De esta manera SIPS+ARVI5 resultó 4.1 veces más rápido para 525696 estados y 4.6 veces más rápido para 1359456 estados, que el otro algoritmo.

En conclusión, al menos en los experimentos realizados, el algoritmo SIPS+ARVI5 que combina reordenamiento estático de estados en forma decreciente de máxima recompensa con barrido priorizado resultó ser el más rápido frente a los algoritmos hasta este punto probados.

6.3. Resultados del Enfoque Propuesto

Se probó el desempeño del enfoque propuesto (**IPVI**) con el de los enfoques que mostraron mejor desempeño en las anteriores secciones. En la Figura 6.6 se muestra el tiempo de solución en función del número de estados de estos algoritmos.

Cabe señalar que IPVI no utiliza ordenamiento estático, sino que explora en forma recursiva actualizando solamente a los estados predecesores de un estado prometedor. En cada actualización cambia el ordenamiento de estados si la diferencia habida en la función de valor es mayor que el umbral de error predeterminado, ε, es decir que **hace ordenamiento dinámico**.

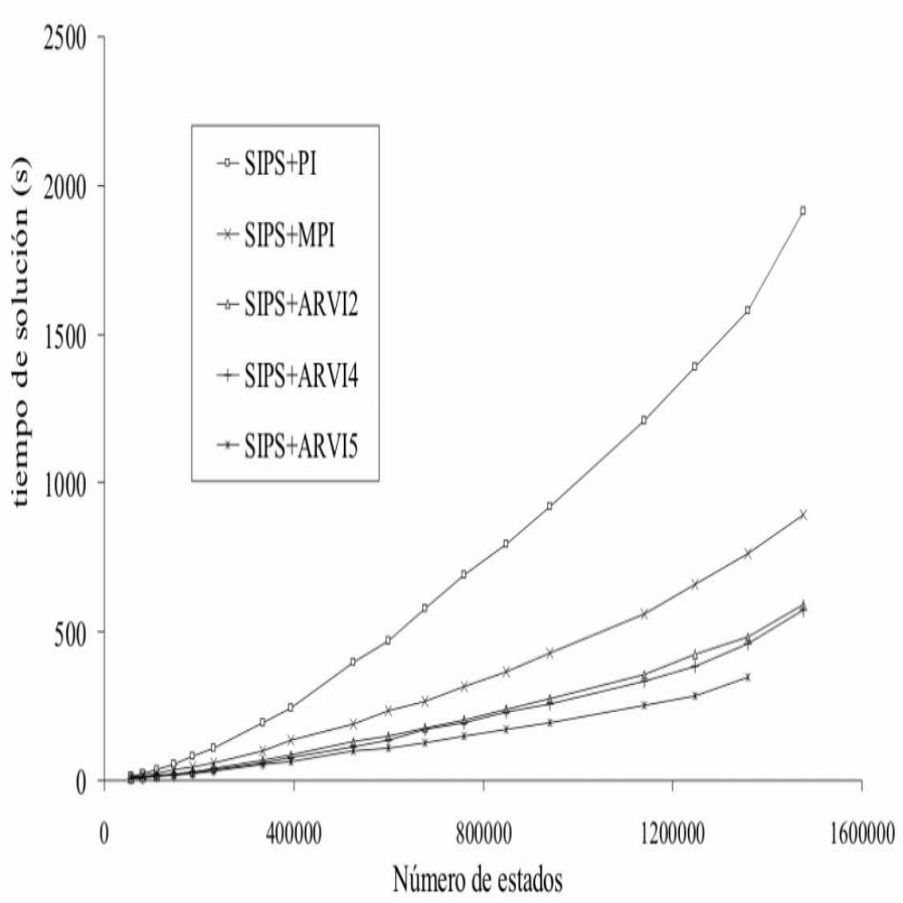

Figura 6.5: Tiempo de solución en función del número de estados de SIPS con aceleración de convergencia.

Algoritmo	tiempo de solución (s)	tiempo de solución relativo
iTVI	-	-
VDP	1376.6	24.0
ARVI2	755.1	13.2
ARVI4	486.5	8.5
ARVI5	412.0	7.2
SIPS+ARVI5	193.0	3.4
IPVI	57.4	1.0

Tabla 6.4: Resumen de resultados en términos del tiempo de solución obtenido por los algoritmos probados, para 940896 estados. El tiempo relativo de solución se calculó con respecto a IPVI.

En la Tabla 6.4 se muestran los resultados numéricos correspondientes a la Figura 6.6, para 940896 estados. Ahí se puede observar que IPVI fue 3.4 veces más rápido que SIPS+ARVI5, 7.2 veces más rápido que ARVI5, 8.5 veces más rápido que ARVI4, 13.2 veces más rápido que ARVI2 y 24 veces más rápido que VDP.

Como se puede observar en la citada figura, iTVI agotó la memoria computacional asignada aproximadamente en 400000 estados. Claramente se puede observar que **el algoritmo más rápido resultó ser IPVI**.

Cabe destacar que se decidió detener los experimentos en problemas con aproximadamente 1.6 millones de estados debido a que, aunque el enfoque propuesto todavía tenía la capacidad de resolver problemas, los demás algoritmos ya habían agotado la memoria del ordenador utilizado.

En la Tabla 6.5 se presenta la relación de tiempos de solución en función del número de estados, presentados en experimentos con 1359456 estados. El algoritmo propuesto requirió solamente de 81.5 segundos, mientras que su más cercano competidor, SIPS+ARVI5, requirió de 346.1 segundos. El resto de los algoritmos no logró procesar tal cantidad de estados, pues agotó la memoria asignada, pero la tendencia de sus tiempos de solución con respecto al número de estados presenta un crecimiento bastante notorio con respecto al que presenta IPVI (vea Figura 6.6).

En resumen, IPVI resuelve 4.24 veces más rápido que la combinación que utiliza barrido priorizado, actualización asíncrona y ordenamiento estático de estados priorizados por su máxima recompensa (SIPS+ARVI5), dejando atrás a los otros enfoques compactos del estado del arte aquí probados.

Por otro lado, en la Figura 6.7 se muestra un acercamiento a la figura anterior con el objeto de comparar a iTVI con los demás algoritmos que también aplican ordenamiento de estados. Por ejemplo, para 393216 estados, **la forma**

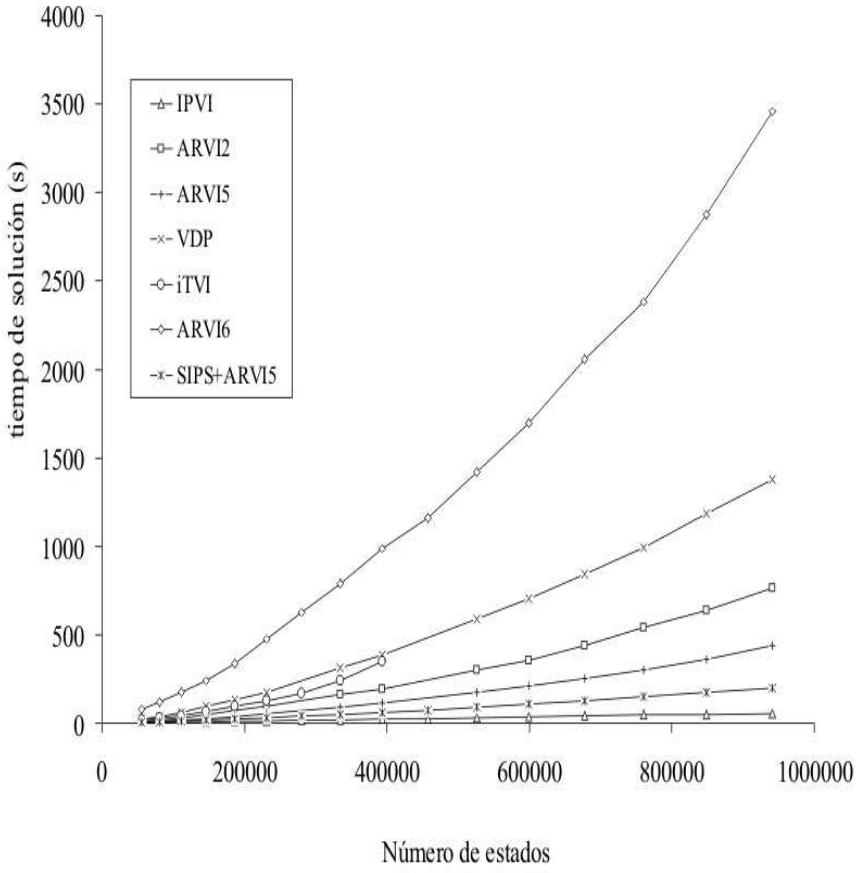

Figura 6.6: Tiempo de solución en función del número de estados de SIPS+ARVI5, ARVI y variantes, VDP, iTVI e IPVI.

Algoritmo	tiempo de solución (s)	tiempo de solución relativo
SIPS+ARVI5	346.1	4.24
IPVI	81.5	1.00

Tabla 6.5: Resumen de resultados en términos de tiempo de solución de SIPS+ARVI5 e IPVI, para 1359456 estados. El tiempo relativo de solución se calculó con respecto a este último algoritmo.

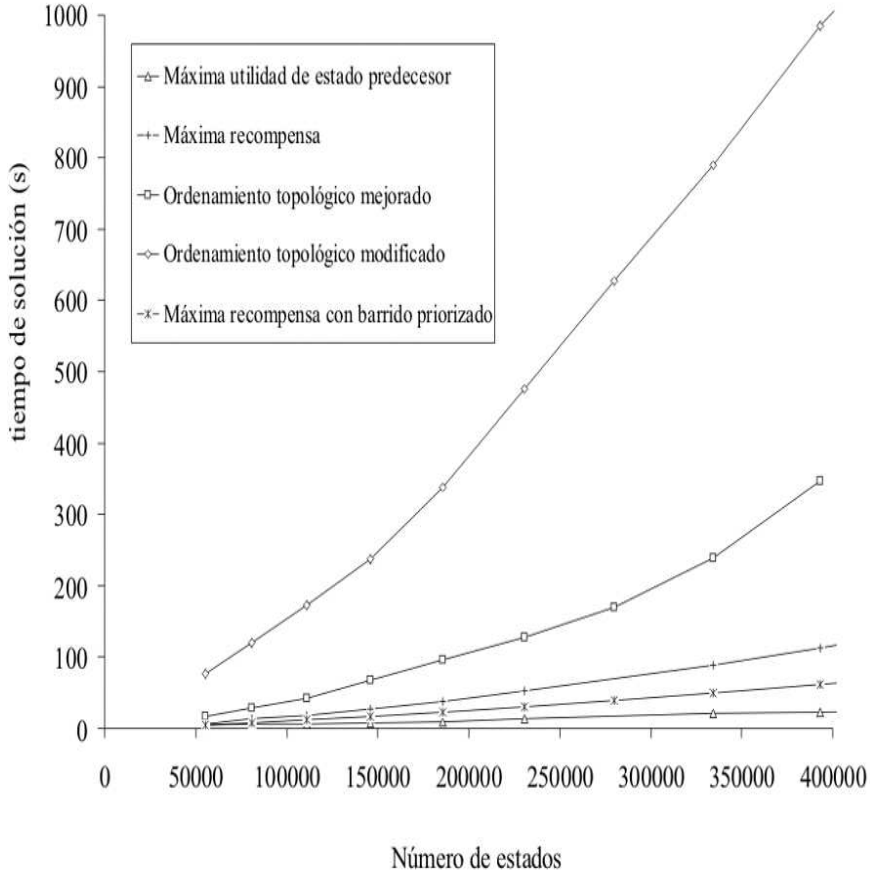

Figura 6.7: Acercamiento de la Figura 6.6, donde se muestra el desempeño de los algoritmos que aplican ordenamiento de estados.

de ordenar a los estados predecesores del estado en evaluación por su máxima utilidad que aplica IPVI fue 2.7 veces más rápido que el ordenar a los estados por su máxima recompensa y con barrido priorizado de McMahan *et al.* [McMahan, 2005a,b] que utiliza SIPS+ARVI5, 4.9 veces más rápido que solamente ordernarlos por su máxima recompensa de Wingate *et al.* [Wingate, 2005] que usa ARVI5, 15.2 veces más rápido que el ordenamiento topológico mejorado de Dibangoye *et al.* [Dibangoye, 2008] que ejecuta iTVI y 43 veces más rápido que el ordenamiento topológico modificado de Wingate *et al.* [Wingate, 2005] que aplica ARVI6.

En la Tabla 6.6 se muestran los resultados numéricos correspondientes a la Figura 6.7.

Algoritmo	tiempo de solución (s)	tiempo de solución relativo
ARVI6	984.4	43.0
iTVI	347.6	15.2
ARVI5	113.3	4.9
SIPS+ARVI5	61.4	2.7
IPVI	22.9	1.0

Tabla 6.6: Resumen de resultados en términos del tiempo de solución obtenido por los algoritmos que usan ordenamiento de estados, para 393216 estados. El tiempo relativo de solución se calculó con respecto a IPVI.

En el **Anexo 6** se muestra una ejecución de los algoritmos que aplican la estrategia de barrido priorizado: SIPS, sus variantes aceleradas y el enfoque propuesto.

Por último, con el objeto de explorar otras estrategias de actualización de predecesores en el enfoque propuesto, se probaron las dos variantes del mismo [Garcia-Hernandez, 2012b] que se preentaron al final del Capítulo 4: **JIPVI** (IPVI con actualización de los predecesores de un estado prometedor mediante el método de Jacobi [Shlakhter, 2005], sin tener en cuenta su orden) y **PPIPVI** (IPVI con priorización de las actualizaciones de los predecesores de un estado prometedor).

En la Figura 6.8 se muestra el tiempo de solución en función del número de estados para IPVI y PPIPVI con las siguientes estrategias de actualización de predecesores: usando la lista completa de transiciones de estado y eliminando repeticiones en la lista de transiciones de estado. En esta figura se puede ver que **no es relevante eliminar estas repeticiones**.

Se puede observar que las mejores estrategias se basaron en IPVI, mientras que las peores estrategias se basaron en PPIPVI. En este caso, es evidente que al priorizar las actualizaciones de los estados predecesores no disminuyó el tiempo de solución debido al coste de inicio que involucra. De hecho, el número de actualizaciones para ambos algoritmos fue casi el mismo.

En la Figura 6.9 se muestra el tiempo de solución en función del número de estados para IPVI y JIPVI. Es notorio que ambos algoritmos lograron un buen tiempo de solución, pero JIPVI fue ligeramente más lento.

En la Figura 6.10 se muestra el número de actualizaciones en función del número de estados de IPVI. Como se puede observar, el número de actualizaciones es casi una función lineal del número de estados. De hecho, el número de actualizaciones realizadas por IPVI fue prácticamente igual al número de transiciones del MDP.

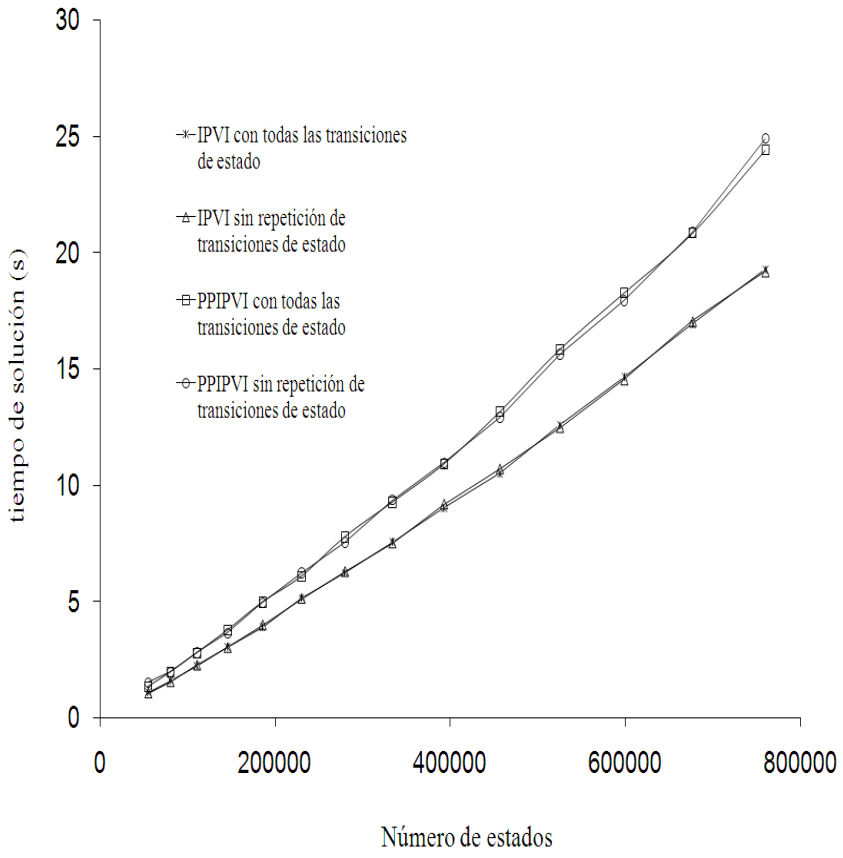

Figura 6.8: Comparación de diferentes estrategias de actualización de predecesores.

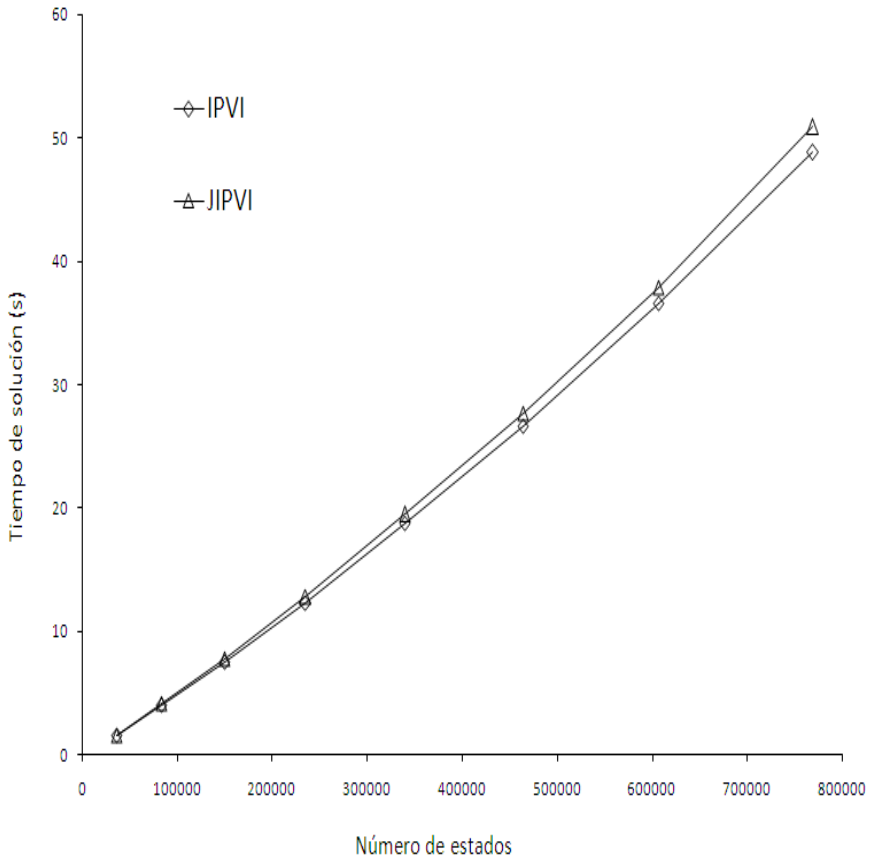

Figura 6.9: Tiempo de solución en función del número de estados para IPVI y JIPVI.

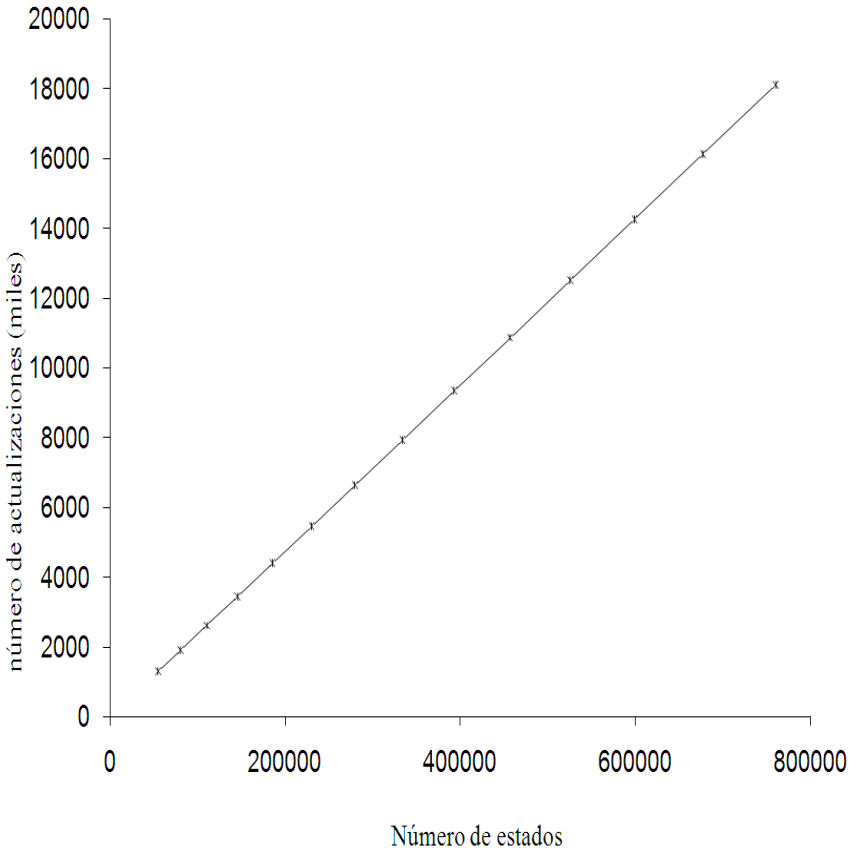

Figura 6.10: Número de actualizaciones de predecesores en función del número de estados para IPVI.

Cabe señalar que tanto IPVI como sus variantes (JIPVI y PPIPVI) entregaron el mismo número de actualizaciones para cada experimento.

Se concluye que el enfoque propuesto en esta tesis resultó ser el más rápido de todos los algoritmos aquí probados en resolver MDP planos (no factorizados), totalmente observables y de considerables dimensiones, incluyendo a sus propias variantes: con actualización de los predecesores de un estado prometedor por el método de Jacobi, sin tener en cuenta su orden; y con priorización de esas actualizaciones.

Experimentalmente se obtuvo que el enfoque propuesto fue el algoritmo más rápido de todos los aquí probados. Al menos en los experimentos realizados, la priorización de las actualizaciones de los predecesores no dió lugar a un menor tiempo de solución debido al coste de inicio que involucra dicho ordenamiento. Por último, el número de actualizaciones realizadas por el enfoque propuesto y sus dos variantes fue cercano a la función lineal del número de transiciones del proceso de decisión de Markov.

6.3.1. Reducción del Tiempo de Solución

De acuerdo con la literatura, la complejidad temporal (o tiempo de ejecución) de los MDP está en función de n_s que es el número de estados, de n_{it} que es el número de iteraciones, de n_a que es el número de acciones y se calcula mediante la ecuación (2.5).

Sin embargo, para problemas complejos ($n_s \to \infty$), como el número de acciones es un valor relativamente pequeño, así como el número de iteraciones, entonces la complejidad temporal queda solamente en función de n_s, resultando cuadrática para este formalismo según la citada ecuación.

Cabe señalar que en esta tesis se calculó la complejidad temporal a partir del tiempo de solución requerido experimentalmente por cada algoritmo implementado, por lo que a las curvas de la Figura 6.6 se les agregó la línea de tendencia y su ecuación correspondiente, resultando de ahí la Figura 6.11.

En la figura anterior se puede observar que la mayoría de las curvas ajustadas presentó un crecimiento potencial del tiempo de solución con respecto al número de estados. De esta manera se obtuvo la potencia a la cual el número de estados (n_s) queda elevado para el cálculo de la complejidad temporal (T) de cada algoritmo y a continuación se presenta:

Para iTVI (VI con ordenamiento topológico mejorado de Dibangoye *et al.* [Dibangoye, 2008]),

$$T\left(n_s\right) = n_s^2 \tag{6.1}$$

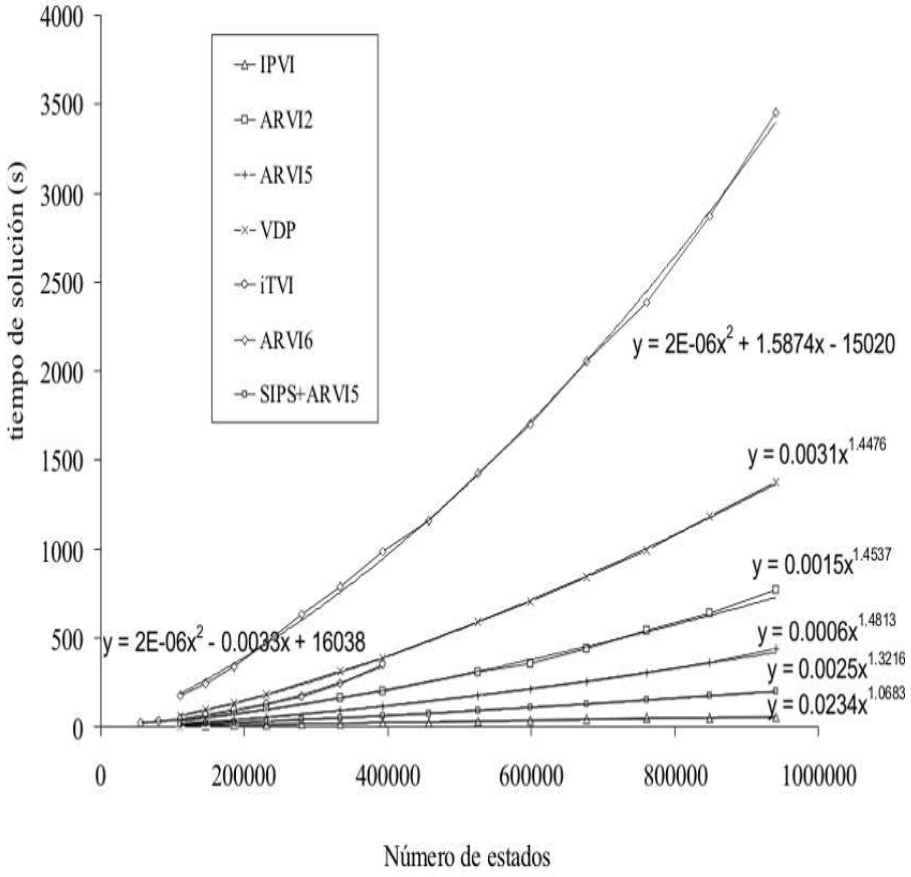

Figura 6.11: Línea de tendencia y ecuación ajustada a la curva del tiempo de solución en función del número de estados de cada algoritmo probado, medida a partir de 110976 estados.

Para ARVI6 (ARVI [García-Hernández, 2009] con ordenamiento topológico modificado de Wingate *et al.* [Wingate, 2006]),

$$T(n_s) = n_s^2 \tag{6.2}$$

Para VDP (programación dinámica usada por el dominio *Sailing* [Vanderbei, 2008]),

$$T(n_s) = n_s^{1,448} \tag{6.3}$$

Para ARVI2 (ARVI con actualización asíncrona de Gauss-Seidel [Puterman, 2005]),

$$T(n_s) = n_s^{1,454} \tag{6.4}$$

Para ARVI5 (ARVI con actualización asíncrona de estados priorizados y reordenamiento estático de Wingate *et al.* [Wingate, 2005]),

$$T(n_s) = n_s^{1,481} \tag{6.5}$$

Para SIPS+ARVI5 (ARVI5 con barrido priorizado mejorado de McMahan *et al.* [McMahan, 2005a,b]),

$$T(n_s) = n_s^{1,322} \tag{6.6}$$

Para IPVI (ARVI con priorización de estados, que es el enfoque propuesto [Garcia-Hernandez, 2012a]),

$$T(n_s) = n_s^{1,068} \tag{6.7}$$

Cabe destacar que la disminución de la complejidad temporal del algoritmo propuesto en esta tesis es considerable, en comparación con la presentada por los demás algoritmos probados.

6.3.2. Reducción del Número de Actualizaciones

Como se señaló en el Capítulo 2, es bien conocido que la complejidad computacional de iteración de valor (y variantes aceleradas) puede ser calculada como la complejidad de la actualización de un estado multiplicada por el número de actualizaciones requeridas para obtener la convergencia a la política óptima.

Por otro lado, como la complejidad de una actualización de estado fue la misma para todos los algoritmos probados (debido a que es igual al número de acciones), es posible hacer una comparación usando solamente el número de actualizaciones ejecutadas por cada algoritmo cuando alcanza la convergencia al plan óptimo.

Algoritmo	número de actualizaciones	número relativo de actualizaciones
iTVI	-	-
VDP	1487405248	66.4
ARVI6	397999008	17.8
ARVI4	397999008	17.8
ARVI2	397999008	17.8
ARVI5	318022848	14.2
SIPS+ARVI5	53034256	2.4
IPVI	22410696	1.0

Tabla 6.7: Resumen de resultados en términos del número de actualizaciones requeridas por los algoritmos probados, para 940896 estados.

Es importante destacar que de los mismos experimentos realizados se extrajo el número de actualizaciones requeridas para alcanzar la convergencia por cada algoritmo implementado. Cabe señalar que IPVI hace **una sola pasada a cada estado relevante** obteniendo la política óptima, correspondiente a una acción contenida en una regla de asociación (vea Capítulo 4). Por lo que **el número de actualizaciones que ejecuta es idéntico al número de reglas de asociación** del problema.

En la Figura 6.12 se muestra el número de actualizaciones ejecutadas en función del número de estados por los algoritmos probados. Ahí se puede observar que IPVI, además de ser el más rápido (en la anterior subsección), también se distingue por ejecutar la menor cantidad de actualizaciones en la obtención de la política óptima, mientras que VDP ejecuta el mayor número de actualizaciones en el mismo fin. Cabe recordar que iTVI agotó el recurso computacional aproximadamente en 400 000 estados.

En la misma figura se puede observar que ARVI2, ARVI4 y ARVI6 ejecutan el mismo número de actualizaciones. La única diferencia que existe entre estos algoritmos es su coste de inicio. También es notorio que las curvas de iTVI y de ARVI5 se empalman. Sin embargo, iTVI demanda una mayor cantidad de memoria debido a que tiene más alto coste de inicio que ARVI5.

Por ejemplo, en la Tabla 6.7 se muestra una comparación en términos del número de actualizaciones de los algoritmos probados cuando el número de estados fue de 940896.

En la misma tabla se puede observar que IPVI requirió 2.4 veces menos actualizaciones que SIPS+ARVI5, 14.2 veces menos actualizaciones que ARVI5, 17.8 veces menos actualizaciones que ARVI2, ARVI4 y que ARVI6, y 66.4 veces menos actualizaciones que VDP.

Por otro lado, a las curvas de la Figura 6.12 se les agregó la línea de tendencia

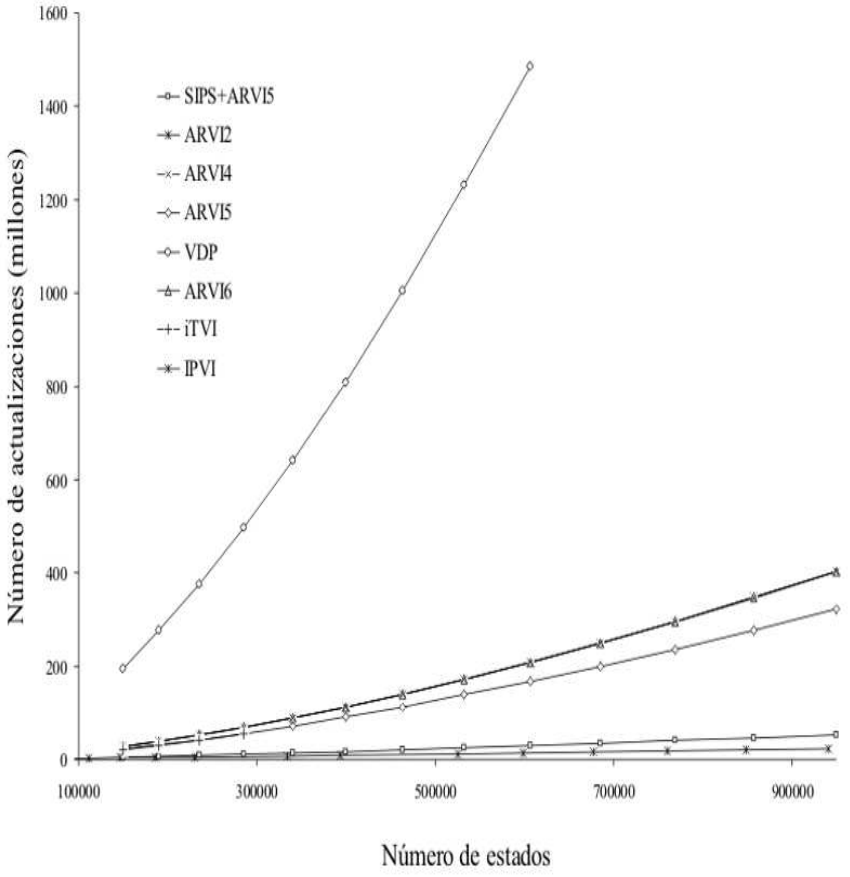

Figura 6.12: Número de actualizaciones en función del número de estados, requerido por cada algoritmo probado.

y ecuación correspondiente, resultando la Figura 6.13.

De la figura anterior se extrajo la ecuación ajustada del número de actualizaciones requeridas (n_{ac}) en función del número de estados (n_s), resultando entonces:

Para VDP (programación dinámica usada por el dominio *Sailing* [Vanderbei, 2008])

$$n_{ac} = 6{,}17 n_s^{1{,}449} \tag{6.8}$$

Para ARVI2 (ARVI [García-Hernández, 2009] con actualización asíncrona de Gauss-Seidel [Puterman, 2005])

$$n_{ac} = 0{,}702 n_s^{1{,}465} \tag{6.9}$$

Para ARVI4 (ARVI con actualización asíncrona de estados priorizados de Wingate *et al.* [Wingate, 2005])

$$n_{ac} = 0{,}702 n_s^{1{,}465} \tag{6.10}$$

Para ARVI6 (ARVI con actualización asíncrona de estados priorizados y ordenamiento topológico de Wingate *et al.* [Wingate, 2005])

$$n_{ac} = 0{,}702 n_s^{1{,}465} \tag{6.11}$$

Para iTVI (de Dibangoye *et al.* [Dibangoye, 2008], sin considerar coste de inicio),

$$n_{ac} = 0{,}646 n_s^{1{,}455} \tag{6.12}$$

Para ARVI5 (ARVI con reordenamiento estático y actualización asíncrona de estados priorizados de Wingate *et al.* [Wingate, 2005]),

$$n_{ac} = 0{,}646 n_s^{1{,}455} \tag{6.13}$$

Para SIPS+ARVI5 (ARVI5 con barrido priorizado mejorado de McMahan *et al.* [McMahan, 2005a,b]),

$$n_{ac} = 0{,}464 n_s^{1{,}348} \tag{6.14}$$

Para IPVI (ARVI con priorización de estados, enfoque propuesto [Garcia-Hernandez, 2012a]),

$$n_{ac} = 21{,}4 n_s^{1{,}008} \tag{6.15}$$

Por último, es importante destacar la notoria disminución del número de actualizaciones necesitadas por el algoritmo propuesto en esta tesis, comparada con la de los demás algoritmos probados.

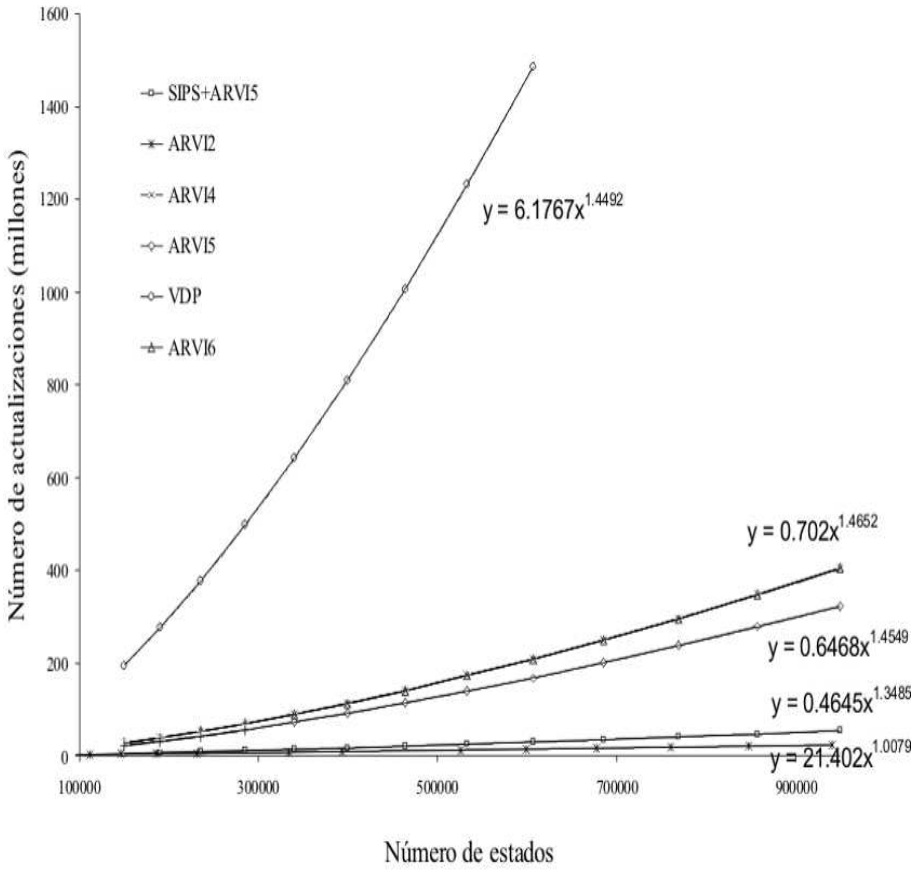

Figura 6.13: Línea de tendencia y ecuación ajustada a la curva del número de actualizaciones en función del número de estados de cada algoritmo probado, medida a partir de 110976 estados.

6.4. Conclusiones del Capítulo

En este capítulo se presentaron los resultados experimentales del desempeño tenido por el enfoque propuesto y por los demás algoritmos implementados, en una tarea compleja de ruta estocástica más corta de navegación marítima.

Es importante señalar que experimentalmente se encontró que el tiempo requerido para el cálculo de las reglas de asociación no resultó significativo dentro del tiempo de solución de los MDP. Lo anterior es debido a que cada regla se calcula una sola vez, mientras que iteración de valor tiene que ejecutar una considerable cantidad de iteraciones para cada estado.

Por otro lado, al aplicar ajuste de curvas en las gráficas obtenidas, en todos los experimentos se demostró que la complejidad temporal del enfoque propuesto disminuyó desde un orden cuadrático hasta uno cercano a la linealidad, en función del número de estados. La misma disminución fue obtenida en la complejidad computacional, medida por el número de actualizaciones requeridas en función del número de estados. Por lo que el enfoque propuesto resolvió más rápidamente que los demás enfoques del estado del arte aquí probados.

En resumen, al calcular solamente sobre un subconjunto de acciones del dominio (mediante reglamentación de acciones) y al ordenar dinámicamente a los estados vecinos más prometedores aplicando el criterio de prioridad de la máxima función de valor, el enfoque propuesto logra reducir el espacio de búsqueda del algoritmo de iteración de valor. De esta manera cumple con el objetivo de esta tesis, que es la simplificación de este formalismo.

Por lo que se concluye que, al menos en los experimentos aquí presentados, la aplicación del nuevo algoritmo de iteración de valor priorizado, basado en el clásico algoritmo de Dijkstra, resulta especialmente atractivo frente a problemas complejos de ruta estocástica más corta. A continuación se presentan las conclusiones finales de la tesis.

Capítulo 7

Conclusiones

En esta tesis se propuso y experimentó un nuevo algoritmo de iteración de valor priorizado con acciones reglamentadas, basado en el clásico algoritmo de Dijkstra para resolver a los procesos de decisión de Markov (MDP) de ruta estocástica más corta.

Cabe destacar que los MDP estudiados en esta tesis son planos (no factorizados) y totalmaente observables. También que los MDP han tenido considerable éxito en la solución de problemas de investigación de operaciones, teoría de control, economía e inteligencia artificial.

A diferencia de otros enfoques priorizados tales como el barrido priorizado mejorado, el enfoque aquí propuesto es capaz de tratar con múltiples estados meta y de inicio y, puesto que sucesivamente se actualiza cada estado utilizando la ecuación de Bellman, este enfoque garantiza la convergencia a la solución óptima. Además, este algoritmo utiliza la función de valor actual como métrica de prioridad, puesto que el algoritmo de Dijkstra sugiere que un orden de actualización más adecuado está dado por el valor de la programación dinámica funcional.

La forma de trabajar del algoritmo propuesto se puede resumir de la siguiente manera: inicia introduciendo en cola de prioridad a los estados meta, luego va extrayendo cada estado habido en cola y actualizando la máxima función de valor de sus estados predecesores, usando operaciones de expansión. Cuando el error de Bellman de un estado predecesor es mayor que el umbral de paro, entonces devuelve a dicho estado a cola de prioridad. En caso contrario, entrega la política óptima de dicho estado. Entonces repite lo anterior sucesivamente, hasta alcanzar al estado de inicio seleccionado.

Es importante destacar que el enfoque propuesto solamente calcula sobre los estados predecesores de cada estado prometedor, reduciendo sustancialmente el espacio de búsqueda. Por otro lado, cuando este algoritmo actualiza la máxima

función de valor de un estado predecesor, solamente calcula sobre las acciones ejecutables en dicho estado usando reglas de asociación, reduciendo también de esta manera el espacio de búsqueda en buena medida.

También cabe señalar que al experimentar con el algoritmo de minería de datos seleccionado para el cálculo de las reglas de asociación, se encontró que el tiempo requerido en dicho cálculo no resultó significativo dentro del tiempo de solución de los MDP. Lo anterior es debido a que cada regla se calcula una sola vez, mientras que el algoritmo de iteración de valor ejecuta una considerable cantidad de iteraciones para cada estado.

El excelente comportamiento del enfoque propuesto es debido a que presenta las siguientes ventajas sobre las otras variantes del algoritmo de iteración de valor aquí probadas:

1. cuando explora un estado prometedor, inmediatamente calcula **la máxima utilidad de sus predecesores**; en cambio otros enfoques del estado del arte calculan **la utilidad del mismo estado prometedor, con el criterio de cota superior que no ha resultado óptimo**.
2. es capaz de ejecutar varias actualizaciones de cada predecesor de un estado prometedor y de utilizar los cálculos obtenidos en el último estado visitado; en cambio otros enfoques de barrido priorizado no actualizan inmediatamente la función de valor.
3. es capaz de calcular solamente sobre las acciones ejecutables en el estado en evaluación, contenidas en reglas de asociación que él mismo calcula. Esto significa que no debe calcular sobre todo el dominio. Además, el tiempo de cálculo de estas reglas no resulta significativo dentro del tiempo de solución de los MDP.
4. es capaz de resolver **problemas no deterministas con múltiples estados meta y de inicio**, mientras que otros enfoques de barrido priorizado del estado del arte solamente pueden tratar **problemas deterministas con un estado meta y uno de inicio**.

Para validar el excelente comportamiento del algoritmo aquí propuesto se comparó con otros algoritmos del estado del arte. Al menos en el problema de estrategias de navegación marítima, el método propuesto fue el más rápido en entregar la política óptima y esto lo logró al ser reducido significativamente su espacio de búsqueda. De esta manera cumplió con el objetivo de esta tesis: simplificar a los MDP no deterministas de considerables dimensiones.

Por otro lado, mediante la línea de tendencia de la gráfica del tiempo de solución en función del número de estados de cada algoritmo implementado, se comprobó que el algoritmo propuesto redujo considerablemente el tiempo de solución requerido por el algoritmo de iteración de valor, desde un crecimiento de

orden cuadrático hasta uno de orden cercano a la linealidad. Lo mismo sucede con el número de actualizaciones requeridas por dicho algoritmo.

Como trabajos relacionados se encontraron en la literatura dos métodos que abordan a los procesos estocásticos como variables de estado continuas y ellos son: discretización de MDP basados en rejilla y aproximaciones paramétricas. Desafortunadamente los algoritmos de rejilla clásicos crecen exponencialmente con el número de variables de estado [Brafman, 1997] [Bonet, 2002]. Por otro lado, el uso de aproximaciones paramétricas con base en técnicas de agrupamiento jerárquico no ha sido exitoso [Pineau, 2003].

7.1. Trabajo futuro

Del trabajo desarrollado en esta tesis se vislumbran tres vertientes:

1. Generalización del enfoque aquí propuesto, de modo que sea capaz de aplicarse en cualquier dominio, no solamente en el dominio de rutas de navegación.

2. Aplicación en cómputo en tiempo real, especialmente en los campos de investigación de operaciones y de teoría de control. Este cómputo tiene inherente una considerable complejidad, pero con la reducción lograda en este aspecto, se espera que el enfoque aquí propuesto tenga un notable desempeño.

3. Como el enfoque aquí propuesto solamente funciona para MDP planos, es decir no factorizados, y completamente observables, entonces otra vertiente sería trabajar con MDP parcialmente observables que incluyeran, además, factorización.

7.2. Publicaciones derivadas de la tesis

A continuación se enlistan en orden cronológico descendente las publicaciones más relevantes (cuatro de ellas pertenecientes al *Journal Citation Report*) derivadas de la presente tesis:

1. Garcia-Hernandez M. G., Ruiz-Pinales J., Onaindia E., Reyes-Ballesteros A., Solving the Sailing Problem with a new Prioritized Value Iteration, Applied Artificial Intelligence, DOI: 10.1080/08839514.2012.687662, ISSN 0883-9514, Taylor & Francis, Vol. 26, Issue 6, 571-587, 2012.

2. Garcia-Hernandez M. G., Ruiz-Pinales J., Onaindia E., Reyes-Ballesteros A., Aviña Cervantes J. G., Ledesma-Orozco S., Alvarado Mendez E., New Prioritized Value Iteration for Markov Decision Processes, Artificial Intelligence Review, An International Science and Engineering Journal, DOI: 10.1007/s10462-011-9224-z, Springer Editors, ISSN 0269-2821, Vol. 37, No.2, 157-167, 2012.

3. García-Hernández M. G., Ruiz-Pinales J., Onaindía E., Reyes-Ballesteros A., Ledesma S., Aviña J. G., Alvarado E., Mixed Acceleration Techniques for solving quickly Stochastic Shortest-Path Markov Decisión Processes, Journal of Applied Research and Technology, ISSN 1665-6423, Vol. 9 No. 2, 129-144, 2011.

4. Garcia-Hernandez M. G., Ruiz-Pinales J., Reyes-Ballesteros A., Onaindía E., Ledesma S., Aviña J. G., Combination of acceleration procedures for solving stochastic shortest-path Markov decision processes, 2010 IEEE International Conference on Intelligent Systems and Knowledge Engineering (ISKE), ISBN 978-1-4244-6790-7, 89-94, Hangzhou, China, November 15, 2010.

5. García-Hernández M. G., Ruiz-Pinales J., Reyes A., Onaindía E., Aviña G., Ledesma S., Acceleration of Association Rule-Based Markov Decision Processes, Journal of Applied Research and Technology, ISSN 1665-6423, Vol. 7, No. 3, 354-375, 2009.

Referencias

Agrawal, 1993 Agrawal R., Imielinski T. & Swami A., Mining Association Rules between Sets of Items in Large Databases, Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 207-216, Washington DC, USA, May 1993.

Agrawal, 1994 Agrawal R. & Srikant R., Fast Algorithms for Mining Association Rules, Proceedings of the 20^{th} Very Large Data Bases (VLDB) Conference, IBM Almaden Research Center, 1215:487-499, Santiago, Chile, 1994.

Agrawal, 2002 Agrawal S.& Roth D., Learning a Sparse Representation for Object Detection, Proceedings of the 7^{th} European Conference on Computer Vision - Part I, Springer-Verlag London, LNCS 2353:1-15, Copenhagen, Denmark, 2002.

Aho, 1988 Aho A. V., Hopcroft J. E. & Ullman J. D., Data Structures and Algorithms, Addison-Wesley Iberoamericana, Wilmington, Delaware, USA, 1988.

Amstron, 1965 Astrom, K. J., Optimal control of Markov decision processes with incomplete state estimation, Journal of Math. Anal. Appl., 10, 1965.

Barto, 1995 Barto, A. G., Reinforcement learning and dynamic programming, Proceedings of the IFAC, Conference on Man-Machine Systems, Cambridge, MA, USA, June 1995.

Barto, 2003 Barto A. & Mahadevan S., Recent advances in hierarchical reinforcement learning, Discrete Event Systems, 13(4):341-379, 2003.

Bellman, 1954 Bellman R. E., The theory of dynamic programming, Bulletin of the American Mathematical Society, 60:503-516, 1954.

Bellman, 1957 Bellman R. E., Dynamic Programming, Princeton United Press, Princeton, NJ, USA, 1957.

Bertsekas, 1987 Bertsekas D. P., Dynamic Programming: Deterministic and Stochastic Models, Prentice Hall, Eaglewood Cliffs, NJ, USA, 1987.

Bertsekas, 1995 Bertsekas D. P., Dynamic Programming and Optimal Control (Chapter 2), Athena Scientific, Belmont, MA, USA, 1995.

Bhuma, 2003 Bhuma K. & Goldsmith J., Bidirectional LAO* Algorithm, Proceedings of Indian International Conferences on Artificial Intelligence, IICAI Pub., pp. 980-992, Hyderabad, India, 2003.

Blackwell, 1965 Blackwell D., Discounted dynamic programming, Annals of Mathematical Statistics, 36:226-235, 1965.

Bonet, 2002 Bonet B. & Pearl J., Qualitative MDP and POMDP: An order-of-magnitude approach, Proceedings of the 18^{th} Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, pp. 61-68, Edmonton, Canada, 2002.

Bonet, 2003a Bonet B. & Geffner H., Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming, Proceedings of the International Conference on Automated Planning and Scheduling, AAI Press, pp. 12-21, Trento, Italy, 2003.

Bonet, 2003b Bonet B. & Geffner H., Faster Heuristic Search Algorithms for Planning with uncertainty and full feedback, Proceedings of the 18^{th} International Joint Conference on Artificial Intelligence, Morgan Kaufmann, pp. 1233-1238, Acapulco, Mexico, 2003.

Bonet, 2006 Bonet B. & Geffner H., Learning depth-first search: A unified approach to heuristic search in deterministic and non-deterministic settings and its application to MDP, Proceedings of the 16^{th} International Conference on Automated Planning and Scheduling, AAAI Press, pp. 142-151, The English Lake District, Cumbria, UK, 2006.

Boutilier, 1999 Boutilier C., Dean T. & Hanks S., Decision-Theoretic Planning: Structural Assumptions and Computational Leverage, Journal of Artificial Intelligence Research, 11:1-94, 1999.

Boutilier, 2000 Boutilier C., Dearden R. & Goldszmidt M., Stochastic Dynamic Programming with factored representations, Artificial Intelligence, 121(1-2):49-107, 2000.

Brafman, 1997 Brafman R., A heuristic variable grid solution method of POMDP, Proceedings of the 14^{th} National Conference on Artificial Intelligence, AAAI Press, pp. 727-733, Menlo Park, CA, USA, 1997.

Brijs, 2004 Brijs T., Swinnen G., Van Hoof K. & Wets G., Building an association rules framework to improve product assortment decisions, Data Mining and Knowledge Discovery, 8(1):7-23, 2004.

Ceglar, 2006 Ceglar A. & Roddick J. F., Association Mining, ACM Computing Surveys, 38(2):Article 5, July 2006.

Cervellera, 2007 Cervellera C., Wen A. & Chen V. C. P., Neural network and regression spline value function approximations for stochastic dynamic programming, Computers & OR, 34(1):70-90, 2007.

Chang, 2005 Chang H. S., Fu M., Hu J. & Marcus S. I., An Adaptive sampling algorithm for solving MDP, Operations Research, 53(1):126-139, 2005.

Chang, 2007 Chang H. S., Fu M., Hu J. & Marcus S. I., Simulation-based algorithms for Markov decision processes (Chapter 4), Communications and Control Engineering, Springer Verlag London Limited, London, UK, 2007.

Chenggang, 2007 Chenggang W., Saket J. & Khardon R., First-Order Decision Diagrams for Relational MDP, Proceedings of International Joint Conference on Artificial Intelligence, Morgan Kaufmann, pp. 1095-1100, Hyderabad, India, 2007

Dai, 2007a Dai P. & Goldsmith J., Faster Dynamic Programming for Markov Decision Processes, *Technical Report*, Doctoral Consortium, Department of Computer Science and Engineering, University of Washington, Washington DC, USA, 2007.

Dai, 2007b Dai P. & Goldsmith J., Topological Value Iteration Algorithm for Markov Decision Processes, Proceedings of the 20^{th} International Joint Conference on Artificial Intelligence, Morgan Kaufmann, pp. 1860-1865, Hyderabad, India, 2007.

Dai, 2007c Dai P. & Hansen E. A., Prioritizing Bellman Backups Without a Priority Queue, Proceedings of the 17^{th} International Conference on Automated Planning and Scheduling, AAAI Press, pp. 113-119, Providence, Rhode Island, USA, September 2007.

De Farías, 2003 De Farias D. P. & Van Roy B., The Linear Programming approach to approximate Dynamic Programming, Operations Research, 51(6):850-856, 2003.

Dean, 1993 Dean T., Kaelbling L. P., Kirman J. & Nicholson A., Planning with deadlines in stochastic domains, Proceedings of the 11^{th} National Conference on Artificial Intelligence, AAAI Press, pp. 574-579, Menlo Park, CA, USA, 1993.

Degris, 2006 Degris T., Sigaud O. & Wuillemin P. H., Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems, Proceedings of the 23^{th} International Conference on Machine Learning, ACM Pub., Vol. 148, pp. 257-264, Pittsburgh, PA, USA, 2006.

Dibangoye, 2008 Dibangoye J. S., Chaib-draa B. & Mouaddib A., A Novel Prioritization Technique for Solving Markov Decision Processes, Proceedings of the 21^{st} International FLAIRS (The Florida Artificial Intelligence Research Society) Conference, Association for the Advancement of Artificial Intelligence, IEEE Computer Society, pp. 537-542, Boca Raton, FL, USA, 2008.

Dietterich, 1997 Dietterich T. G. & Flann N.S., Explanation-based learning and reinforcement learning: A unified view, Machine Learning, 28(503):169-210, 1997.

Dietterich, 2000 Dietterich T. G., Hierarchical Reinforcement Learning with the MAXQ value function decomposition, Journal of Artificial Intelligence Research, 13:227-303, 2000.

Driessens, 2001 Driessens K. & Blockeel H., Learning digger using hierarchical reinforcement learning for concurrent goals, Proceedings of the European Workshop on Reinforcement Learning, pp. 11-12, Utrecht, The Netherlands, October 5-6, 2001.

Driessens, 2003 Driessens K. & Ramon J., Relational instance based regression for relational reinforcement learning, Proceedings of the 20^{th} International Conference on Machine Learning, AAAI Press, pp. 123-130, Washington DC, USA, 2003.

Driessens, 2005 Driessens K., Relational Reinforcement Learning, Artificial Intelligence Communications, 18:71-73, 2005.

Dzeroski, 2001 Dzeroski L., De Raedt L. & Driessens K., Relational Reinforcement Learning, Machine Learning, 43:7-52, 2001.

Epshteyn, 2006 Epshteyn A. & De Jong G., Qualitative Reinforcement Learning, Proceedings of the 23^{th} International Conference on Machine Learning, ACM Pub., Vol. 148, pp. 305-312, Pittsburgh, PA, USA, 2006.

Feldman, 1977 Feldman J. A. & Sproull R. F., Decision Theory and Artificial Intelligence II: The hungry monkey, Cognitive Science, 1:158-192, USA, 1977.

Feng, 2004 Feng Z., Dearden R. & Meuleau N., Washington, R., Dynamic programming for structured continuous Markov decision problems, Proceedings of the 20^{th} Conference on Uncertainty in Artificial Intelligence, AUAI Press, pp. 154-161, Banff, Canada, 2004.

Ferguson, 2004 Ferguson D. & Stentz A., Focused Propagation of MDP for Path Planning, Proceedings of the 16^{th} IEEE International Conference on Tools with Artificial Intelligence, IEEE Computer Society, pp. 310-317, 2004.

Fern, 2003 Fern A., Yoon S. & Givan R., Approximate policy iteration with a policy language bias, Proceedings of the Neural Information Processing Conference, 25:75-118, 2003.

Fernandez, 2008 Fernandez F. & Borrajo D., Two steps reinforcement learning, International Journal of Intelligence Systems, Vol. 23, No. 2, pp. 213-245, 2008.

Ghallab, 2004 Ghallab M., Nau D. & Traverso P., Automated Planning Theory and Practice, Elsevier, Morgan Kaufmann, USA, 2004.

García-Hernández, 2009 García-Hernández M. G., Ruiz-Pinales J., Reyes A., Onaindía E., Aviña G. & Ledesma S., Acceleration of Association Rule-Based Markov Decision Processes, Journal of Applied Research and Technology, 7(3):354-375, 2009.

García-Hernández, 2011 García-Hernández M. G., Ruiz-Pinales J., Onaindía E., Reyes-Ballesteros A., Ledesma S., Aviña J. G. & Alvarado E., Mixed Acceleration Techniques for solving quickly Stochastic Shortest-Path Markov Decisión Processes, Journal of Applied Research and Technology, 9(2):129-144, 2011.

Garcia-Hernandez, 2012a Garcia-Hernandez M. G., Ruiz-Pinales J., Onaindia E., Reyes-Ballesteros A., Avina Cervantes J. G., Ledesma-Orozco S. & Alvarado Mendez E., New Prioritized Value Iteration for Markov Decision Processes, Artificial Intelligence Review, An International Science and Engineering Journal, Springer Ed., 37(2):157-167, 2012.

Garcia-Hernandez, 2012b Garcia-Hernandez M. G., Ruiz-Pinales J., Onaindia E. & Reyes-Ballesteros A., Solving the Sailing Problem with a new Prioritized Value Iteration, Applied Artificial Intelligence, Taylor & Francis, in press, 2012.

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

Getoor, 2001 Getoor L., Friedman N., Koller D. & Taskar B., Learning probabilistic models of relational structure, Proceedings of the 18^{th} International Conference on Machine Learning, Morgan Kaufmann, pp. 170-177, Williamstown, MA, USA, 2001.

Givan, 2003 Givan R., Dean T. & Greig M., Equivalence notions and model minimization in Markov Decision Processes, Artificial Intelligence, 147:163-223, 2003.

Gretton, 2004 Gretton C. & Thiebaux S., Exploiting first-order regression in inductive policy selection, Proceedings of the 20^{th} Conference on Uncertainty in Artificial Intelligence, pp. 217-225, AUAI Press, Banff, Canada, 2004.

Guestrin, 2001 Guestrin C., Koller D. & Parr R., Max-norm projections for factored MDP, Proceedings of the 17^{th} International Joint Conference on Artificial Intelligence, Morgan Kaufman, pp. 673-682, 2001.

Guestrin, 2002 Guestrin C. & Gordon G. J., Distributed Planning in Hierarchical Factored MDP, Proceedings of the 18^{th} Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, pp. 197-206, 2002.

Guestrin, 2003 Guestrin C., Koller D. & Parr R., Efficient solution algorithms for factored MDP, Journal of Artificial Intelligence Research, 19:399-468, 2003.

Guestrin, 2004 Guestrin C., Hauskrecht M. & Kveton B., Solving Factored MDP with Continuous and Discrete Variables, Proceedings of the 20^{th} Conference on Uncertainty in Artificial Intelligence, AUAI Press, pp. 235-242, Banff, Canada, July 2004.

Gupta, 2006 Gupta G. K., Introduction to Data Mining with Case Studies, pp. 76-82, Prentice-Hall of India, Pvt. Ltd, 2006.

Hansen, 2001 Hansen E. A. & Zilberstein S., LAO*: A Heuristic Search Algorithm that finds solutions with Loops, Artificial Intelligence, 129:35-62, 2001.

Hashler, 2006 Hashler M., Hornik K. & Reutterer T., Implications of Probabilistic Data Modeling for Mining Association Rules, Studies in Classification, Data Analysis and Knowledge Organization, pp. 598-605, Springer Verlag, 2006.

Hauskrecht, 2004 Hauskrecht M. & Kveton B., Linear program approximations for factored continuous-state Markov Decision Processes, Advances in Neural Information Processing Systems, MIT Press, 16:895-902, 2004.

Hinderer, 2003 Hinderer K. & Waldmann K. H., The critical discount factor for Finite Markovian Decision Processes with an absorbing set, Mathematical Methods of Operations Research, Physica Verlag, 57:1-19, Heidelberg, Germany, 2003.

Hoey, 1999 Hoey J., St. Aubin R., Hu A. & Boutilier C., SPUDD: Stochastic planning using decision diagrams, Proceedings of the 15^{th} Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, pp. 279-288, 1999.

Hoey, 2005 Hoey J. & Poupart P., Solving POMDP with Continuous or Large Discrete Observation Spaces, Proceedings of the International Joint Conference on Artificial Intelligence, Professional Book Center, pp. 1332-1338, Edinburgh, Scotland, July 2005.

Howard, 1960 Howard R. A., Dynamic Programming and Markov Processes, Technology Press-Wiley, MIT Press, Cambridge, MA, USA, 1960.

Jamali, 2006 Jamali M., Learning to Solve Stochastic Shortest Path Problems, Technical Report No. 2006-02, Web Intelligence Web Laboratory, Sharif University of Technology, pp. 1-9, Tehran, Iran, July 2006.

Jong, 2006 Jong N. K. & Stone P., Kernel-based Models for Reinforcement Learning (ICML), Proceedings of the Workshop on Kernel Methods and Reinforcement Learning at International Conference on Machine Learning, Pittsburgh, PA, USA, June 2006.

Jonsson, 2006 Jonsson A. & Barto A., Causal Graph based Decomposition of Factored MDP, Journal of Machine Learning Research, 7:2259-2302, 2006.

Kaelbling, 1998 Kaelbling L., Littman M. L. & Cassandra A. R., Planning and Acting in Partially Observable Stochastic Domains, Artificial Intelligence, 101(1-2):99-134, 1998.

Kalyanakrishnan, 2007 Kalyanakrishnan S. & Stone P., Batch Reinforcement Learning in a Complex Domain, Proceedings of the Autonomous Agents and Multi Agent Systems Conference, IFAAMAS Pub., pp. 650-657, Honolulu, Hawai, May 2007.

Kersting, 2003 Kersting K. & De Raedt L., Logical Markov Decision Programs, Proceedings of the International Joint Conference on Artificial Intelligence, Workshop on Learning Statistical Models of Relational Data, Morgan Kaufmann, pp. 63-70, 2003.

Kersting, 2006 Kersting K., An Inductive Logic Programming Approach to Statistical Relational Learning, Frontiers in Artificial Intelligence and its Applications Series (Dissertations), IOS Press, 148:389-390, Amsterdam, The Netherlands, 2006.

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

Koller, 2000 Koller D. & Parr R., Policy iteration for factored MDP, Proceedings of the 16^{th} Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, pp. 326-334, Stanford, CA, USA, 2000.

Kochenderfer, 2005 Kochenderfer M. & Hayes G., Adaptive Partitioning of State Spaces using decision graphs for Real-Time Modeling and Planning, Proceedings of the Workshop on Planning and Learning in a Priori Unknown or Dynamic Domains at International Joint Conference on Artificial Intelligence, Professional Book Center, pp. 9-14, 2005.

Kveton, 2006 Kveton B., Hauskrecht M. & Guestrin C., Solving Factored MDP with Hybrid State and Action Variables, Journal of Artificial Intelligence Research, 27:153-201, 2006.

Lawrence, 2001 Lawrence R. D., Almasi G. S., Kotlyar V., Viveros M. S. & Duri S., Personalization of supermarket product recommendations, Data Mining and Knowledge Discovery, 5(1-2):11-31, 2001.

Lecoeuche, 2001 Lecoeuche R., Learning optimal dialogue management rules by using reinforcement learning and inductive logic programming, North American Chapter of the Association for Computational Linguistics (NAACL), N01-1028, Pittsburgh, USA, 2001.

Li, 2009 Li L., A Unifying Framework for Computational Reinforcement Learning Theory, *PhD thesis dissertation*, Department of Computer Science, The State University of New Jersey, Rutgers University, New Brunswick, NJ, USA, October, 2009.

Littman, 1995 Littman M. L. & Dean T. L., Kaebling L., On the Complexity for Solving MDP, Proceedings of the 11^{th} International Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, pp. 394-402, Montreal, Quebec, Canada, 1995.

Littman, 2002 Littman M., Sutton R. & Singh S., Predictive representations of State, Advances in Neural Information Processing Systems, 14:1555-1561, MIT Press, 2002.

Luce, 1957 Luce R. D. & Raiffa H., Games and Decisions: Introduction and Critical Survey, Wiley Ed., USA, 1957.

Mannor, 2004 Mannor S. & Menache I., Dynamic Abstraction in Reinforcement Learning via Clustering, Proceedings of the 21^{th} International Conference on Machine Learning, ACM Pub., Vol. 69, Banff, Canada, 2004.

Mausam, 2003 Mausam & Weld D. S., Solving relational MDP with first-order Machine Learning, Proceedings of the Workshop on Planning under Uncertainty and Incomplete Information at 13^{th} International Conference on Automated Planning and Scheduling, AAAI Press, pp. 1-8, Trento, Italy, June 2003.

McGovern, 1997 McGovern A., Sutton R. & Fagg A., Roles of macroactions in Accelerating Reinforcement Learning, Grace Hopper Celebration of Women in Computing, pp. 13-18, 1997.

McMahan, 2005a McMahan H. B. & Gordon G., Fast Exact Planning in Markov Decision Processes, Proceedings of the 15^{th} International Conference on Automated Planning and Scheduling, AAAI Press, pp. 151-160, Monterey, CA, USA, 2005.

McMahan, 2005b McMahan H. B. & Gordon G., Generalizing Dijkstra's Algorithm and Gaussian Elimination for Solving MDP, Technical Report, Carnegie Mellon University, Pittsburgh, PA, USA, May 2005b.

Moore, 1993 Moore A., Atkeson C., Prioritized Sweeping: Reinforcement Learning with less data and less real time, Machine Learning Journal, 13:103-130, 1993.

Morales, 2003 Morales E., Scaling up reinforcement learning with a Relational Representation, Workshop on Adaptability in Multi Agent Systems (AORC),Commonwealth Scientific and Industrial Research Organisation, Sydney, Australia, 2003.

Morrison, 1999 Morrison J. R. & Kumar P. R., New linear program performance bounds for queueing networks, Journal Optimization Theory Application, 100(3):575-597, 1999.

Munos, 2002 Munos R. & Moore A., Variable resolution discretization in optimal control. Machine Learning Journal, 49:291-323, 2002.

Paschalidis, 2000 Paschalidis I. C. & Tsitsiklis J. N., Congestion-dependent pricing of network services, IEEE/ACM Trans. Networking, 8(2):171-184, 2000.

Pasula, 2004 Pasula H. M, Zettlemoyer L. S. & Kaelbling L. P., Learning probabilistic planning rules, Proceedings of the 14^{th} International Conference on Automated Planning and Scheduling, AAAI Press, pp. 73-82, 2004.

Pasula, 2005 Pasula H. M., Zettlemoyer L. S. & Kaelbling L. P., Learning planning rules in noisy stochastic worlds, AAAI Press, pp. 911-918, 2005.

Pineau, 2003 Pineau J., Gordon G. & Thrun S., Policy-contingent abstraction for robust control, Proceedings of the 19^{th} Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, pp. 477-484, 2003.

Puterman, 1994 Puterman M. L., Markov Decision Processes - Discrete Stochastic Dynamic Programming, John Wiley & Sons, New York, USA, 1994.

Puterman, 2005 Puterman M. L., Markov Decision Processes, Wiley Interscience, John Wiley & Sons, New York, USA, 2005.

Quinlan, 1993 Quinlan J. R., C4.5: Programs for machine learning, Morgan Kaufmann, San Francisco, CA, USA, 1993.

Rafols, 2005 Rafols E., Ring M., Sutton R. & Tanner B., Using Predictive Representations to Improve Generalization in Reinforcement Learning, Proceedings of the 9^{th} International Joint Conference on Artificial Intelligence, Professional Book Center, pp. 835-840, 2005.

Reyes, 2006a Reyes A., Planificación con Incertidumbre usando Procesos de Decisión de Markov Cualitativos, *Tesis Doctoral*, Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM), México, Noviembre 2006.

Reyes, 2006b Reyes A., Ibarguengoytia P., Sucar L. E. & Morales E., Abstraction and Refinement for Solving Continuous Markov Decision Processes, Proceedings of the 3^{rd} European Workshop on Probabilistic Graphical Models (PGM), pp. 263-270, Czech Republic, 2006.

Roncagliolo, 2004 Roncagliolo S. & Tadepadelli P., Function approximation in hierarchical relational reinforcement learning, Proceedings of the Workshop on Relational Reinforcement Learning at International Conference on Machine Learning, ACM Pub., 2004.

Russell, 2004 Russell S. & Norvig P., Artificial Intelligence: A Modern Approach. Making Complex Decisions (Chapter 17), 2^{nd} edition, Pearson-Prentice Hall, Berkeley, CA, USA, 2004.

Ryan, 2004 Ryan M. R. K., Hierarchical Reinforcement Learning: A hybrid approach, *PhD Thesis*, University of NSW, School of Computer Science and Engineering, Sydney, Australia, 2004.

Safaei, 2007 Safaei J. & Ghassem-Sani G., Incremental Learning of Planning Operators in Stochastic Domains, Proceedings of the 33^{rd} International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), Springer Verlag Pub., LNCS 4362:644-655, 2007.

Sallans, 2002 Sallans B., Reinforcement learning for factored Markov Decision Processes, *PhD Thesis*, Department of Computer Science, University of Toronto, Toronto, Canada, 2002.

Sanner, 2005 Sanner S., Simultaneous learning of structure and value in relational reinforcement learning, Proceedings of the Workshop on Rich Representations for Reinforcement Learning at the International Conference on Machine Learning, Bonn, Germany, 2005.

Schuurmans, 2001 Schuurmans D. & Patrascu R., Direct value approximations for factored MDP, Advances in Neural Information Processing Systems, MIT Press, pp. 1579-1586, 2001.

Shani, 2008 Shani G., Brafman R. & Shimony S., Prioritizing point based POMDP solvers, IEEE Transactions on Systems, Man. and Cybernetics, 38(6):1592-1605, 2008.

Shlakhter, 2005 Shlakhter O., Lee Ch., Khmelev D. & Jaber N., Acceleration Operators in the Value Iteration Algorithms for Markov Decision Processes, *Technical Report*, Department of Mathematics, University of Toronto, Toronto, Ontario, Canada, M5S 3G8, June 19, 2005.

Sniedovich, 2006 Sniedovich M., Dijkstra's algorithm revisited: the dynamic programming connection, Control and Cybernetics, 35(3):599-620, 2006.

Sniedovich, 2010 Sniedovich M., Dynamic Programming: Foundations and Principles, 2^{nd} edition, Pure and Applied Mathematics Series, Taylor & Francis, Melbourne, Australia, 2010.

Spaan, 2004 Spaan M., Porta J. & Vlassis N., Value Iteration for continuous-state POMDP, *Technical Report IAS-UVA-04-04*, Informatics Institute, University of Amsterdam, The Netherlands, 2004.

Sutton, 1998 Sutton R. S. & Barto A. G., Reinforcement Learning: An introduction (Chapter 9), MIT Press, Cambridge, MA, USA, 1998.

Sutton, 1999 Sutton R. S., Precup D. & Singh S., Between MDP and semi-MDP: A framework for temporal abstraction in reinforcement learning, Artificial Intelligence, 112(1-2):181-211, 1999.

Tijms, 2003 Tijms H. C., A First Course in Stochastic Models, Wiley Ed., Discrete-Time Markov Decision Processes (Chapter 6), John Wiley & Sons, Ltd, Chichester, UK, 2003.

Trigos, 2005 Trigos F. & Frausto J., Experimental Evaluation of the theta-heuristic for starting the cosine simplex method, Proceedings of the International Conference on Computational Science and its Applications, Springer, Singapore, May 2005.

Vanderbei, 1996 Vanderbei R. J., Optimal Sailing Strategies, Statistics and Operations Research Program, University of Princeton, Princeton, NJ, USA, 1996 (http://www.princeton.edu/ rvdb/).

Vanderbei, 2008 Vanderbei R. J., Linear Programming: Foundations and Extensions (Chapter 15), Springer Verlag, 3^{th} edition, Princeton, NJ, USA, 2008.

Van den Poel, 2004 Van den Poel D., Schamphelaere J. & Wets G., Direct and indirect effects of retail promotions on sales and profits in the do-it-yourself market, Expert Systems with Applications, 27(1):53-62, 2004.

Van Otterlo, 2004 Van Otterlo M., Kersting K. & De Raedt L., Bellman goes Relational, Proceedings of the 21^{st} International Conference on Machine Learning, ACM Pub., Vol. 69, Banff, Canada, 2004.

Van Otterlo, 2005 Van Otterlo M., A survey of Reinforcement Learning in Relational Domains, TR-CTIT-05-31, CTIT Technical Report Series, ISBN-1381-3625, 2005.

Wang, 2006 Wang T., Poupart P., Bowling M. & Schuurmans D., Compact, Convex Upper Bound Iteration for Approximate POMDP Planning, AAAI Press, pp. 1245-1251, 2006.

Wierstra, 2004 Wierstra D. & Wiering M., Utile distinction Hidden Markov Models, Proceedings of the International Conference on Machine Learning, ACM Pub., Vol.69, Banff, Canada, 2004.

Wingate, 2005 Wingate D. & Seppi K. D., Prioritization Methods for Accelerating MDP Solvers, Journal of Machine Learning Research, 6:851-881, 2005.

Witten, 2005 Witten I. H., Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations (Chapter 6), 2^{nd} Edition, Morgan Kaufmann, USA, 2005.

Yoon, 2005 Yoon S. W., Fern A. & Givan R., Learning measures of progress for planning domains, Proceedings of the 16^{th} International Conference on Automated Planning and Scheduling, AAAI Press, pp. 1217-1222, 2005.

Zhang, 1996 Zhang, N. L. & Poole, D., Exploiting causal independence in Bayesian network inference, Journal of Artificial Intelligence Research, 5:301-328, 1996.

Zhang, 2006 Zhang W. & Zhang N., Restricted Value Iteration: Theory and Algorithms, Journal of Artificial Intelligence Research, 23:123-165, 2006.

Anexo 1

Ejemplo de reglamentación de acciones devuelta por el algoritmo Apriori.

Escenario: movimiento de un robot en un espacio bidimensional con 400 estados, 5 acciones y 12 recompensas.

ALGORITMO ARVI

NOTA: Se presenta impresa sólo una parte de las reglas.

```
s222,s202 => a1 Prob =
3.9588499999999875E-6
s222,s223 => a2 Prob =
0.7054012817806001
s222,s221 => a3 Prob =
0.7072623860905641
s222,s222 => a4 Prob =
0.9995200575999972
s223,s243 => a0 Prob =
6.299216300366285E-6
s223,s203 => a1 Prob =
3.958849999999985E-6
s223,s224 => a2 Prob =
0.6684361932441699
s223,s222 => a3 Prob =
0.6632477948955956
s223,s223 => a4 Prob =
0.9995200575999972
s224,s244 => a0 Prob =
6.299216300366285E-6
s224,s204 => a1 Prob =
3.958849999999985E-6
s224,s225 => a2 Prob =
9.999999999999973E-11
s224,s223 => a3 Prob =
0.7810595078867363
s224,s224 => a4 Prob =
0.9995200575999971
s225,s245 => a0 Prob =
2.0342482300884914E-6
s225,s205 => a1 Prob =
7.781655429864217E-6
s225,s226 => a2 Prob =
0.8025769506769214
s225,s224 => a3 Prob =
9.999999999999978E-11
s225,s225 => a4 Prob =
0.9995200575999978
s226,s246 => a0 Prob =
2.0342482300884914E-6
s226,s206 => a1 Prob =
7.781655429864215E-6
s226,s227 => a2 Prob =
0.8288317612683 4
s226,s225 => a3 Prob =
0.8296858275999983
s226,s226 => a4 Prob =
0.999520057599998
s227,s247 => a0 Prob =
2.0342482300884914E-6
s227,s207 => a1 Prob =
7.781655429864219E-6
s227,s228 => a2 Prob =
0.6988697733627104
s227,s226 => a3 Prob =
0.6975898786638284
s227,s227 => a4 Prob =
0.9995200575999978
s228,s248 => a0 Prob =
2.0342482300884914E-6
s228,s208 => a1 Prob =
7.781655429864219E-6
s228,s229 => a2 Prob =
0.8199128927685376
s228,s227 => a3 Prob =
0.8407088224717931
s228,s228 => a4 Prob =
0.999520057599998
s229,s249 => a0 Prob =
2.0342482300884914E-6
s229,s209 => a1 Prob =
7.781655429864226E-6
s229,s230 => a2 Prob =
0.6691804250895373
s229,s228 => a3 Prob =
0.6413239656530959
s229,s229 => a4 Prob =
0.9995200575999978
s230,s250 => a0 Prob =
7.963451769911489E-6
s230,s210 => a1 Prob =
7.781655429864215E-6
s230,s231 => a2 Prob =
0.7454687564135576
s230,s229 => a3 Prob =
0.7829482788315255
s230,s230 => a4 Prob =
0.9995200575999977
s231,s251 => a0 Prob =
7.963451769911489E-6
s231,s211 => a1 Prob =
7.781655429864217E-6
s231,s232 => a2 Prob =
0.753067449124662
s231,s230 => a3 Prob =
0.7132272978702674
s231,s231 => a4 Prob =
0.9995200575999977
s232,s252 => a0 Prob =
7.963451769911487E-6
s232,s212 => a1 Prob =
7.781655429864219E-6
s232,s233 => a2 Prob =
0.7203598454137636
s232,s231 => a3 Prob =
0.7085756605113909
s232,s232 => a4 Prob =
0.9995200575999978
s233,s253 => a0 Prob =
7.963451769911489E-6
s233,s213 => a1 Prob =
7.781655429864219E-6
s233,s234 => a2 Prob =
0.8428199295607824
s233,s232 => a3 Prob =
0.8473006466476172
s233,s233 => a4 Prob =
0.9995200575999977
s234,s254 => a0 Prob =
7.96345176991149E-6
s234,s214 => a1 Prob =
7.781655429864226E-6
s234,s235 => a2 Prob =
0.7834807032756725
s234,s233 => a3 Prob =
0.7744738564288272
s234,s234 => a4 Prob =
0.9995200575999977
s235,s255 => a0 Prob =
7.963451769911489E-6
s235,s215 => a1 Prob =
7.781655429864219E-6
s235,s236 => a2 Prob =
0.6803939351630236
s235,s234 => a3 Prob =
0.6985124974795166
s235,s235 => a4 Prob =
0.9995200575999977
s236,s256 => a0 Prob =
7.963451769911489E-6
```

- NOTA: SE CORTÓ LA IMPRESIÓN DE REGLAS POR CUESTIÓN DE ESPACIO -

```
s603,s603 => a4 Prob =
0.9995200576000005
s604,s604 => a0 Prob =
0.9995200576000004
s604,s584 => a1 Prob =
7.374265282392031E-6
s604,s605 => a2 Prob =
0.66918042508954
s604,s603 => a3 Prob =
0.6413239656530976
s604,s604 => a4 Prob =
0.9995200576000004
s605,s605 => a0 Prob =
0.9995200576000003
s605,s585 => a1 Prob =
7.37426528239203E-6
s605,s606 => a2 Prob =
0.7454687564135595
s605,s604 => a3 Prob =
0.7829482788315274
s605,s605 => a4 Prob =
0.9995200576000004
s606,s606 => a0 Prob =
0.9995200576000004
s606,s586 => a1 Prob =
7.37426528239203E-6
s606,s607 => a2 Prob =
0.753067449124664
s606,s605 => a3 Prob =
0.7132272978702705
s606,s606 => a4 Prob =
0.9995200576000004
s607,s607 => a0 Prob =
0.9995200576000003
s607,s587 => a1 Prob =
7.37426528239203E-6
s607,s608 => a2 Prob =
0.7203598454137655
s607,s606 => a3 Prob =
0.7085756605113926
s607,s607 => a4 Prob =
0.9995200576000003
s608,s608 => a0 Prob =
0.9995200576000004
s608,s588 => a1 Prob =
```

```
7.3742652823920 31E-6
s608,s609 => a2 Prob =
0.8428199295607846
s608,s607 => a3 Prob =
0.8473006466476193
s608,s608 => a4 Prob =
0.9995200576000004
s609,s609 => a0 Prob =
0.9995200576000003
s609,s589 => a1 Prob =
7.3742652823920 31E-6
s609,s610 => a2 Prob =
0.7834807032756759
s609,s608 => a3 Prob =
0.7744738564288292
s609,s609 => a4 Prob =
0.9995200576000003
s610,s610 => a0 Prob =
0.9995200576000002
s610,s590 => a1 Prob =
7.3742652823920 31E-6
s610,s611 => a2 Prob =
0.6803939351630254
s610,s609 => a3 Prob =
0.6985124974795183
s610,s610 => a4 Prob =
0.9995200576000003
s611,s611 => a0 Prob =
0.9995200576000003
s611,s591 => a1 Prob =
7.3742652823920 33E-6
s611,s612 => a2 Prob =
0.7115785869220342
s611,s610 => a3 Prob =
0.7010154172103897
s611,s611 => a4 Prob =
0.9995200576000003
s612,s612 => a0 Prob =
0.9995200576000003
s612,s592 => a1 Prob =
7.3742652823920 33E-6
s612,s613 => a2 Prob =
0.7865814210426232
s612,s611 => a3 Prob =
0.8112844478898552
s612,s612 => a4 Prob =
0.9995200576000003
s613,s613 => a0 Prob =
0.9995200576000003
s613,s593 => a1 Prob =
7.3742652823920 32E-6
s613,s614 => a2 Prob =
0.7406163003272728
s613,s612 => a3 Prob =
0.7125852256198022
s613,s613 => a4 Prob =
0.9995200576000003
s614,s614 => a0 Prob =
0.9995200576000002
s614,s594 => a1 Prob =
7.3742652823920 33E-6
s614,s615 => a2 Prob =
0.6839313433894738
s614,s613 => a3 Prob =
0.724155605377778
s614,s614 => a4 Prob =
0.9995200576000002
s615,s615 => a0 Prob =
0.9995200576000002
s615,s595 => a1 Prob =
7.3742652823920 31E-6
s615,s616 => a2 Prob =
0.7851216267902835
s615,s614 => a3 Prob =
0.7293017622938778
s615,s615 => a4 Prob =
0.9995200576000001
s616,s616 => a0 Prob =
0.9995200576000002
s616,s596 => a1 Prob =
7.3742652823920 32E-6
s616,s617 => a2 Prob =
0.8086796343415732
s616,s615 => a3 Prob =
0.7788145055116467
s616,s616 => a4 Prob =
0.9995200576000002
s617,s617 => a0 Prob =
0.9995200576000002
s617,s597 => a1 Prob =
7.3742652823920 34E-6
s617,s618 => a2 Prob =
0.6629203053777779
s617,s616 => a3 Prob =
0.6964919953419355
s617,s617 => a4 Prob =
0.9995200576000002
s618,s618 => a0 Prob =
0.9995200576000002
s618,s598 => a1 Prob =
7.3742652823920 32E-6
s618,s619 => a2 Prob =
0.7703767569233084
s618,s617 => a3 Prob =
0.7698614792129034
s618,s618 => a4 Prob =
0.9995200576000002
s619,s619 => a0 Prob =
0.9995200576000001
s619,s599 => a1 Prob =
7.3742652823920 34E-6
s619,s619 => a2 Prob =
0.2534473464405797
s619,s618 => a3 Prob =
0.7215427740788732
s619,s619 => a4 Prob =
0.9995200576000001

numero de iteraciones: 485
converge en 266 milisegundos
La ruta optima de accion es:

4 2 2 2 2 0 0 0 0 0 0 0 2 0 0 0

0 0 3 0 0 2 0 0 0 2 0 0 0 0 3 0
2 0 0 0 0 3 0 0 2 2 0 0 2 0 2 0
0 0 2 0 0 0 0 0 0 3 3 3 0 2 2 0
0 2 2 0 0 0 0 2 2 0 0 0 0 0 3 0
2 2 2 2 2 0 0 2 0 0 2 0 0 0 0 0
3 3 0 3 2 2 2 2 2 0 0 2 2 0 0 2
2 0 0 0 3 0 3 0 0 2 2 0 0 2 2 2
2 0 2 2 2 2 0 0 0 3 0 0 2 2 0 2
2 2 2 2 0 0 0 2 2 2 0 3 3 3 3 3
2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 3
3 3 3 3 2 2 2 2 2 1 2 2 2 2 2 2
2 2 1 3 3 3 3 3 2 0 0 0 0 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 4
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
2 1 1 1 1 3 3 3 3 3 0 0 0 3 3 0
3 0 3 3 0 2 2 0 0 3 3 0 3 3 3 3
3 0 3 0 0 0 3 3 2 2 2 4 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 2 2 1 1
1 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3
0 2 2 0 0 1 3 3 3 1 3 3 1 3 3 3
3 3 3 3 2 2 0 2 2 0 3 3 0 3 0 0
0 3 3 3 3 3 3 3 2 2 2 2 2 4 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2
2 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3
2 2 2 2 2 2 2 2 2 2 0 2 2 0 0 3
3 0 3 3 2 2 2 2 2 2 2 2 2 2 2 2
2 2 4 3 3 3 3 3 3 2 2 2 2 2 1 2 2
1 1 2 2 1 1 1 3 3 1 3 3 1 2 2 1
1 1 1 1 2 2 1 2 1 1 1 1 3 3 3 1
2 2 2 2 1 1 2 1 1 1 1 1 1 1 1 3
3 3 1 3 2 2 1 1 1 1 2 1 2 2 1 1
2 1 1 3 1 1 3 1 0 2 2 0 0 2 2 2
2 0 0 0 2 2 0 0 0 3 0 0 2 2 0 2
2 0 2 2 0 0 2 2 2 2 0 0 0 3 3 3
0 2 2 2 2 2 2 0 2 2 2 2 2 0 0 0
0 0 3 3 2 2 2 2 2 2 2 2 2 2 2 2
2 2 4 3 3 3 3 3 3 2 2 2 2 2 2 2
2 2 1 2 2 1 1 3 1 1 3 3 4 4 4 4
4
```

Anexo 2

Procedimiento para ejecutar el simulador de planificación de movimientos robóticos (SPRM) [Reyes, 2006a].

- Se abre la carpeta "sistema" → ProyectoJBuilder → ProyectoJBuilder.java
- Se abren las clases MarkovDecisionProcess.java, RelationalActions.java y FMDP.java
- En FMDP.java se crea un objeto del tipo MarkovDecisionProcess (sin acciones reglamentadas). En otra ejecución se creará un objeto del tipo RelationalActions (con acciones reglamentadas).
- En el menú se pincha "Run Project", que es la ejecución del proyectoJBuilder.
- Aparece la interfaz "Sistema de Planificación con Incertidumbre" en el que se selecciona "Simulation" 'Robotics
- .ªutomáticamente se abre otra interfaz:" Ambiente de Navegación"
- Para cargar un ejemplo, en "Ambiente de Navegación" se selecciona de su menú "Ambiente"→ recuperar → russell → escoge problema → discreto → archivo "ambiente".
- Se presiona "cargar", en el apartado "Planificador SPI", con partición "discreta".
- Inmediatamente, en el espacio de estados (interfaz "Ambiente de Navegación"), aparecen los estados con recompensa (azules) y con penalización (amarillos), además se pueden ver diferentes tonalidades de ellos, lo cual significa que son de diferente valor (multivaluados), correspondiendo a mayor valor el de tono mas fuerte y a menor valor el de tono mas débil.
- En la misma interfaz se selecciona el número de muestras y el tiempo (milisegundos, ms), paso lineal del robot, coordenadas de inicio de exploración. En el resto de los parámetros se pueden usar los predeterminados.

****

- En el caso en que se desea explorar y generar los archivos de atributos y ejemplos, se debe seleccionar el archivo donde se desean ponerlos (resultan 26). Para esto se abre la carpeta "ProyectoJBuilder" → ejemplos → russell → el problema seleccionado → discreto → se genera "nueva carpeta" → todos los archivos que estén fuera de esta "nueva carpeta" se seleccionan y se pegan dentro de dicha carpeta. De esta manera quedó libre de archivos la carpeta previa "discreto".
- Después se establece el numero de muestras, la velocidad de exploración (time step, ms), paso lineal del robot, paso angular del mismo, coordenadas del punto de inicio del robot. Se pueden usar los valores predeterminados.
- Se presiona "Explorar" en la interfaz "Ambiente de Navegación". Con esto conocerá el modelo de transición y su vector de recompensas.
- Se puede observar cómo el robot (punto rojo) se va moviendo a lo largo y a lo ancho del espacio de estados. Esta operación puede tardar muchos minutos.
- Una vez terminada la exploración se puede verificar que se hayan generado archivos en la carpeta que seleccionamos para ello.

****

- En el menú "Planificador" de la interfaz "Ambiente de Navegación" se selecciona "compilar". Con esto aparece una tercera interfaz gráfica "Compilador de FMDP".
- En el apartado "Aprendizaje" (a la derecha) se capturan los atributos y el ejemplo, para cada caso abre la interfaz correspondiente: archivo de atributos, archivo de atributos discretos, archivo de Ejemplos.
- Se carga "Atributos" → "russell" → "discreto" → atributos.txt
- Se carga "Atributos Discretos" ""russell" → "discreto" → atributosDisc.txt
- Se carga "Ejemplo" → "russell" → "discreto" → Ejemplos.dat
- Si se desea, se pueden modificar algunos campos.
- Se selecciona "fmdp" parte superior del compilador.
- Se verifica que en "Planificador SPI" este seleccionado partición "discreta".
- Se presiona "compilar". En este momento esta ejecutando el método "ValueIteration" de la clase MarkovDecisionProcess para cualquiera de las dos pruebas mencionadas.
- Devuelve el plan óptimo, número de iteraciones necesarias y tiempo de ejecución. Se puede seleccionar y copiar este resultado en un archivo de texto, para su futura consulta.

- Se puede volver a presionar "compilar" para una segunda prueba del mismo tipo.
- En la interfaz del IIE se selecciona el menú "Planificador" → "ver" → "política" o "recompensa" o "función de valor" o "modelo de transición". Sobre el espacio de estados se verán las diferentes opciones seleccionadas.

Anexo 3

Ejemplos de ejecución de algoritmos síncronos: ARVI y ARVI3.

Escenario: dominio *Sailing* con 940896 estados y 24 acciones probables.

```
ARVI:
M = 200
N = 200
Punto A = (1, 1)
Punto B = (198, 198)
Direccion inicial viento = NE
Direccion inicial velero = N
Direccion viento con respecto a velero = 1
Ns = 940896
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Planificador = Rules MDP
numero de iteraciones: 624
numero de operaciones: 1099252608
converge en 1108219 milisegundos
numero de iteraciones: 624
numero de operaciones: 1099252608
converge en 1068078 milisegundos
numero de iteraciones: 624
numero de operaciones: 1099252608
converge en 1065312 milisegundos
Tiempo esperado para alcanzar
punto B = 913.3088744869368

ARVI3:
M = 200
N = 200
Punto A = (1, 1)
Punto B = (198, 198)
Dirección inicial viento = NE
Dirección inicial velero = N
Dirección viento con respecto a velero = 1
Ns = 940896
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Planificador = Rules MDP
Numero de reglas = 22410504


Value iteration con Priorizacion
----+------+--------+---------
 n    iter    nop     tiempo
----+------+--------+---------
0       623 377589706   777735
1       623 377589706   765375
2       623 377589706   762484
3       623 377589706   774266
4       623 377589706   842015
5       623 377589706   763094
6       623 377589706   779797
7       623 377589706   763844
8       623 377589706   795922
9       623 377589706   798313
----+------+--------+---------
Tiempo de convergencia promedio = 782284.5 ms
Variacion = 23426.752055075318 ms
heapSize = 1144303616 heapMaxSize = 1598226432
heapFreeSize = 464685016

Tiempo esperado para alcanzar
punto B = 913.3088694289472
```

NOTA: Ambos algoritmos obtuvieron el **plan óptimo**, lo cual se puede verificar observando el tiempo esperado para alcanzar el punto B, que ambos reportan.

Anexo 4

Ejemplos de ejecución de algoritmos asíncronos: ARVI2, ARVI4, ARVI5 y VDP.

Escenario: dominio *Sailing* con 940896 estados y 24 acciones probables.

```
ARVI2

M = 200
N = 200
Punto A = (1, 1)
Punto B = (198, 198)
Direccion inicial viento = NE
Direccion inicial velero = N
Direccion viento con respecto a velero = 1
Ns = 940896
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Planificador = Rules MDP
numero de iteraciones: 423
numero de operaciones: 889708600
converge en 755094 milisegundos
... 0 755094.0
1 744781.0
2 764671.0
3 780047.0
4 783188.0
5 764968.0
6 769375.0
7 768516.0
8 766297.0
9 764312.0
Tiempo de convergencia promedio = 781124.9 ms

Tiempo esperado para alcanzar
punto B = 913.4430869319652

ARVI4:

M = 200
N = 200
Punto A = (1, 1)
Punto B = (198, 198)
Dirección inicial viento = NE
Dirección inicial velero = N
Dirección viento con respecto a velero = 1
Ns = 940896
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Planificador = Rules MDP
Numero de reglas = 22410504
Value iteration con Priorizacion

----+------+--------+---------\\
 n    iter    nop     tiempo\\
----+------+--------+---------\\

0       423 213458928   486547\\
1       423 213458928   473375\\
2       423 213458928   517860\\
3       423 213458928   468797\\
4       423 213458928   563828\\
5       423 213458928   475797\\
6       423 213458928   495625\\
7       423 213458928   472797\\
8       423 213458928   527719\\
9       423 213458928   509375\\
----+------+--------+---------\\
Tiempo de convergencia promedio = 499172.0 ms
Variacion = 29070.261464252537 ms
heapSize = 1146011648 heapMaxSize = 1598226432
heapFreeSize = 470592688

Tiempo esperado para alcanzar
punto B = 913.4430835163721

ARVI5:

M = 200
N = 200
Punto A = (1, 1)
Punto B = (198, 198)
Direccion inicial viento = NE
Direccion inicial velero = N
Direccion viento con respecto a velero = 1
Ns = 940896
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Planificador = Rules MDP
Numero de reglas = 22410504
Value iteration con Priorizacion
Reordenamiento estatico de estados
```

```
----+------+--------+---------\\
 n    iter    nop     tiempo\\
----+------+--------+---------\\
0       338 183567087   412000\\
1       338 183567087   423812\\
2       338 183567087   406766\\
3       338 183567087   489453\\
4       338 183567087   411000\\
5       338 183567087   425968\\
6       338 183567087   524750\\
7       338 183567087   414000\\
8       338 183567087   438765\\
9       338 183567087   416515\\
----+------+--------+---------\\

Tiempo de convergencia promedio = 436302.9 ms
Variacion = 37294.763437913825 ms
heapSize = 1205796864 heapMaxSize = 1598226432
heapFreeSize = 127646208
```

```
Tiempo esperado para alcanzar
punto B = 913.4430847142711

VDP:


0 373 1377110\\
1 373 1376062\\
2 373 1376219\\
3 373 1375906\\
4 373 1376188\\
5 373 1376578\\
6 373 1376516\\
7 373 1376890\\
8 373 1377344\\
9 373 1376906\\
Tiempo promedio de convergencia = 1376571.9 ms
Variación = 455.74125344090874ms
expected best time = 913.4430870163233
```

NOTA: Los cuatro algoritmos obtuvieron el plan óptimo, lo cual se puede verificar observando el tiempo esperado para alcanzar el punto B, que todos reportan.

Anexo 5

Ejemplos de ejecución de algoritmos con ordenamiento de estados sin barrido priorizado: ARVI5 y ARVI6.

Se probaron ARVI5 y ARVI6 hasta con 848256 estados.

```
ARVI5:

M = 190
N = 190
Punto A = (1, 1)
Punto B = (188, 188)
Dirección inicial viento = NE
Dirección inicial velero = N
Dirección viento con respecto a velero = 1
Ns = 848256
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Planificador = Rules MDP
Numero de reglas = 20195784
Value iteration con Priorizacion
Reordenamiento estatico de estados

----+------+--------+---------\\
 n    iter    nop     tiempo\\
----+------+--------+---------\\

0       322 157604659   346875\\
1       322 157604659   348829\\
2       322 157604659   353203\\
3       322 157604659   347953\\
4       322 157604659   400594\\
5       322 157604659   355828\\
6       322 157604659   358953\\
7       322 157604659   352422\\
8       322 157604659   404671\\
9       322 157604659   356125\\
----+------+--------+---------\\
Tiempo de convergencia promedio = 362545.3 ms
Variacion = 20390.96697584513 ms
heapSize = 1065549824 heapMaxSize = 1598226432
heapFreeSize = 275638824

Tiempo esperado para alcanzar punto B = 868.0319628971403
```

```
ARVI6:

M = 190
N = 190
Punto A = (1, 1)
Punto B = (188, 188)
Dirección inicial viento = NE
Dirección inicial velero = N
Dirección viento con respecto a velero = 1
Ns = 848256
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Planificador = Rules MDP
Numero de reglas = 20195784
Value iteration con Priorizacion
Reordenamiento estatico de estados

----+------+--------+---------\\
 n    iter    nop     tiempo\\
----+------+--------+---------\\
0       308 185829516  2635297\\
1       308 185829516  2634359\\
2       308 185829516  2629469\\
3       308 185829516  2637718\\
4       308 185829516  2676594\\
5       308 185829516  2638000\\
6       308 185829516  2651578\\
7       308 185829516  2652218\\
8       308 185829516  2695609\\
9       308 185829516  2654953\\
----+------+--------+---------\\
Tiempo de convergencia promedio = 2650579.5 ms
Variacion = 19982.977522140955 ms
heapSize = 1065549824 heapMaxSize = 1598226432
heapFreeSize = 321201040

Tiempo esperado para alcanzar punto B = 868.0319640200812
```

Anexo 6

Ejemplos de ejecución de algoritmos con barrido priorizado, incluyendo a IPVI

Se muestra la ejecución de problemas con 1359456 estados, aunque IPVI siguió ejecutando para mayor cantidad de estados en el ordenador descrito en el Capítulo 4, en el dominio *Sailing* [Vanderbei, 1996].

```
SIPS
(solo ejecutó hasta 393,216 estados)

M = 130
N = 130
Punto A = (1, 1)
Punto B = (128, 128)
Dirección inicial viento = NE
Dirección inicial velero = N
Dirección viento con respecto a velero = 1
Ns = 393216
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Ncorridas = 1
Planificador = Rules MDP
Numero de reglas = 9326856

Sparse Improved Prioritized Sweeping
----+------+--------+---------
n iter nop tiempo
----+------+--------+---------
0 1 0 61235
----+------+--------+---------
Tiempo de convergencia promedio = 61235.0 ms
Variacion = 0.0 ms
heapSize = 832438272 heapMaxSize = 832438272
heapFreeSize = 383945512
Tiempo esperado para alcanzar
punto B = -609.5542115351805 (NOTA: es plan
subóptimo)

SIPS+PI

M = 240
N = 240
Punto A = (1, 1)
Punto B = (238, 238)
Dirección inicial viento = NE
Dirección inicial velero = N
Dirección viento con respecto a velero = 1
Ns = 1359456
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Ncorridas = 1
Planificador = Rules MDP
Numero de reglas = 32421576

Sparse Improved Prioritized Sweeping + Policy
Iteration
----+------+--------+---------
n iter nop tiempo
----+------+--------+---------
0 7 0 1580594
----+------+--------+---------
Tiempo de convergencia promedio = 1580594.0 ms
Variacion = 0.0 ms
heapSize = 1598226432 heapMaxSize = 1598226432
heapFreeSize = 47578376
Tiempo esperado para alcanzar
punto B = -1095.0180633346226

SIPS+MPI

M = 240
N = 240
Punto A = (1, 1)
Punto B = (238, 238)
Dirección inicial viento = NE
Dirección inicial velero = N
Dirección viento con respecto a velero = 1
Ns = 1359456
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Ncorridas = 1
Planificador = Rules MDP
Numero de reglas = 32421576

Sparse Improved Prioritized Sweeping + Modified
Policy Iteration
----+------+--------+---------
n iter nop tiempo
```

```
----+------+--------+---------
0 232 0 763672
----+------+--------+---------
Tiempo de convergencia promedio = 763672.0 ms
Variacion = 0.0 ms
heapSize = 1598226432 heapMaxSize = 1598226432
heapFreeSize = 68723344
Tiempo esperado para alcanzar
punto B = -1095.018063798485
```

SIPS+ARVI2

```
M = 240
N = 240
Punto A = (1, 1)
Punto B = (238, 238)
Dirección inicial viento = NE
Dirección inicial velero = N
Dirección viento con respecto a velero = 1
Ns = 1359456
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Ncorridas = 10
Planificador = Rules MDP
Numero de reglas = 32421576

Sparse Improved Prioritized Sweeping + Asynchronous
Sparse Value Iteration
----+------+--------+---------
n iter nop tiempo
----+------+--------+---------
0 418 0 482578
----+------+--------+---------
Tiempo de convergencia promedio =482578.0 ms
Variacion = 0.0 ms
heapSize = 1598226432 heapMaxSize = 1598226432
heapFreeSize = 68723344
Tiempo esperado para alcanzar
punto B = -1095.018063798485
```

SIPS+ARVI4

```
M = 240
N = 240
Punto A = (1, 1)
Punto B = (238, 238)
Dirección inicial viento = NE
Dirección inicial velero = N
Dirección viento con respecto a velero = 1
Ns = 1359456
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Ncorridas = 1
Planificador = Rules MDP
Numero de reglas = 32421576

Sparse Improved Prioritized Sweeping + Accelerated
Asynchronous Sparse Value Iteration
----+------+--------+---------
n iter nop tiempo
----+------+--------+---------
0 413 0 458390
----+------+--------+---------
Tiempo de convergencia promedio = 458390.0 ms
Variacion = 0.0 ms
heapSize = 1598226432 heapMaxSize = 1598226432
heapFreeSize = 43176160
Tiempo esperado para alcanzar
punto B = -1095.0598485570426
```

SIPS+ARVI5

```
M = 240
N = 240
Punto A = (1, 1)
Punto B = (238, 238)
Dirección inicial viento = NE
Dirección inicial velero = N
Dirección viento con respecto a velero = 1
Ns = 1359456
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Ncorridas = 1
Planificador = Rules MDP
Numero de reglas = 32421576

Sparse Improved Prioritized Sweeping + Accelerated
Sparse Value Iteration with Static Reordering
----+------+--------+---------
n iter nop tiempo
----+------+--------+---------
0 294 0 346125
----+------+--------+---------
Tiempo de convergencia promedio = 346125.0 ms
Variacion = 0.0 ms
heapSize = 1550385152 heapMaxSize = 1550385152
heapFreeSize = 11993088
Tiempo esperado para alcanzar
punto B = -1095.0180646020997
```

IPVI

```
M = 240
N = 240
Punto A = (1, 1)
Punto B = (238, 238)
Dirección inicial viento = NE
Dirección inicial velero = N
Dirección viento con respecto a velero = 1
Ns = 1359456
Factor desc = 1.0 Nit = 1000 max error = 1.0E-7
Ncorridas = 1
Planificador = Rules MDP
Numero de reglas = 32421576

Improved Prioritized Value Iteration
----+------+--------+---------
n iter nop tiempo
----+------+--------+---------
0 1 32424432 81562
----+------+--------+---------
Tiempo de convergencia promedio = 81562.0 ms
Variacion = 0.0 ms
heapSize = 1502003200 heapMaxSize = 1550385152
heapFreeSize = 149761640
Tiempo esperado para alcanzar
punto B = -1095.018169312484
```