Título del Libro

Capítulo 10

Procesos de Decisión de Markov y Aprendizaje por Refuerzo

10.1 Introducción

Un agente, ya sea humano o computacional, se enfreta continuamente al problema de toma de decisiones bajo incertidumbre. En base a información limitada del ambiente, el agente debe tomar la *mejor* decisión de acuerdo a sus objetivos. En muchas ocaciones este proceso se repite en forma secuencial en el tiempo, de manera que en cada instante el agente recibe información y decide que acción tomar, en base a sus objetivos a largo plazo. A esto se le denomina *problemas de decisión secuencial*. Por ejemplo, un agente de bolsa tiene que decidir en cada momento en base a la información con la que cuenta, que acciones vender y comprar, para maximizar su utilidad a largo plazo. Otro ejemplo es el de un robot que debe navegar de un punto a otro, de forma que en cada instante de tiempo debe decidir sus movimientos, en base a la información de sus sensores y conocimiento del ambiente, para llegar en la forma más directa (de menor costo) a la meta.

Esta clase de problemas se pueden modelar, desde una perspectiva de teoría de decisiones, como *Procesos de Decisión de Markov*; donde se especifican los posibles estados en que puede estar el agente, sus posibles acciones, una función de recompensa basada en sus preferencias, y un modelo de su dinámica representado como una función de probabilidad. Establecido el

modelo, se puede encontrar una solución que de la mejor acción para cada estado (política) para maximizar la utilidad del agente a largo plazo. Otra forma de resolver este tipo de problemas es mediante *Aprendizaje por Refuerzo*. En este caso no se tiene un modelo dinámico del agente, por lo que se aprende la política óptima en base a prueba y error explorando el ambiente.

En este capítulo se presenta un panorama general de procesos de decisión de Markov (MDPs) y aprendizaje por refuerzo (RL). En primer lugar se da una definición formal de lo que es un MDP, que se ilustra con un ejemplo. Después se describen las técnicas de solución básicas, ya sea cuando se tiene un modelo (programación dinámica) o cuando no se tiene (métodos Monte Carlo y aprendizaje por refuerzo). En la sección 10.4 se presentan otras técnicas de solución alternativas a los enfoques básicos. En seguida se presentan métodos de solución para problemas más complejos, mediante modelos factorizados y abstractos. Finalmente se ilustran estos métodos en 3 aplicaciones: aprendiendo a volar un avión, controlando una planta eléctrica y coordinando a un robot mensajero.

10.2 Procesos de Decisión de Markov

En esta sección analizaremos el modelo básico de un MDP, así como su extensión a los procesos parcialmente observables o POMDPS.

10.2.1 MDPs

Un MDP modela un problema de decisión sequencial en donde el sistema evoluciona en el tiempo y es controlado por un agente. La dinámica del sistema está determinada por una función de transición de probabilidad que mapea estados y acciones a otros estados.

Formalmente, un MDP es una tupla $M =< S, A, \Phi, R >$ [29]. Los elementos de un MDP son:

- Un conjunto finito de estados $S : \{s_1, \ldots, s_n\}$, donde s_t denota el estado $s \in S$ al tiempo t.
- Un conjunto finito de acciones que pueden depender de cada estado, $A(s)$, donde $a_t \in A(s)$ denota la acción realizada en un estado s en el tiempo t.

- Una función de recompensa ($\mathcal{R}^a_{ss'}$) que regresa un número real indicando lo deseado de estar en un estado $s' \in S$ dado que en el estado $s \in S$ se realizó la acción $a \in A(s)$.

- Una función de transición de estados dada como una distribución de probabilidad ($\mathcal{P}^a_{ss'}$) que denota la probabilidad de llegar al estado $s' \in S$ dado que se tomó la acción $a \in A(s)$ en el estado $s \in S$, que también denotaremos como $\Phi(s, a, s')$.

Dado un estado $s_t \in S$ y una acción $a_t \in A(s_t)$, el agente se mueve a un nuevo estado s_{t+1} y recibe una recompensa r_{t+1}. El mapeo de estados a probabilidades de seleccionar una acción particular definen una *política* (π_t). El problema fundamental en MDPs es encontrar una política óptima; es decir, aquella que maximiza la recompensa que espera recibir el agente a largo plazo.

Las políticas o reglas de decisión pueden ser [29]: (i) deterministas o aleatorias, (ii) estacionarias o no–estacionarias. Una política determinista selecciona simpre la misma acción para cierto estado, mientras que una aleatoria selecciona con cierta probabilidad (de acuerdo a cierta distribución especificada) una acción de un conjunto posible de acciones. Una política estacionaria decide la misma acción para cada estado independientemente del tiempo, esto es, $a_t(s) = a(s)$ para toda $t \in T$; lo cual no se cumple para una no–estacionaria. En este capítulo nos enfocamos a políticas estacionarias y deterministas.

Si las recompensas recibidas después de un tiempo t se denotan como: r_{t+1}, r_{t+2}, r_{t+3}, ..., lo que se quiere es maximizar la recompensa total (R_t), que en el caso más simple es:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T$$

Si se tiene un punto terminal se llaman tareas *episódicas*, si no se tiene se llaman tareas *continuas*. En este último caso, la fórmula de arriba presenta problemas, ya que no podemos hacer el cálculo cuando T no tiene límite. Podemos usar una forma alternativa en donde se van haciendo cada vez más pequeñas las contribuciones de las recompensas más lejanas:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

donde γ se conoce como la *razón de descuento* y está entre: $0 \leq \gamma < 1$. Si $\gamma = 0$ se trata de maximizar la recompensa total tomando en cuenta sólo las recompensas inmediatas.

En general existen los siguientes tipos modelos de acuerdo a lo que se busca optimizar:

1. Horizonte finito: el agente trata de optimizar su recompensa esperada en los siguientes h pasos, sin preocuparse de lo que ocurra después:

$$E(\sum_{t=0}^{h} r_t)$$

   donde r_t significa la recompensa recibida t pasos en el futuro. Este modelo se puede usar de dos formas: (i) *política no estacionaria*: donde en el primer paso se toman los siguientes h pasos, en el siguiente los $h-1$, etc., hasta terminar. El problema principal es que no siempre se conoce cuántos pasos considerar. (ii) *receding-horizon control*: siempre se toman los siguientes h pasos.

2. Horizonte infinito: las recompensas que recibe un agente son reducidas geométricamente de acuerdo a un factor de descuento γ ($0 \leq \gamma < 1$) considerando un número infinito de pasos:

$$E(\sum_{t=0}^{\infty} \gamma^t r_t)$$

3. Recompensa promedio: al agente optimiza a largo plazo la recompensa promedio:

$$lim_{h\to\infty} E(\frac{1}{h}\sum_{t=0}^{h} r_t)$$

   En este caso, un problema es que no hay forma de distinguir políticas que reciban grandes recompensas al principio de las que no.

El modelo más utilizado es el de horizonte infinito.

Las acciones del agente determinan no sólo la recompensa inmediata, sino también en forma probabilística el siguiente estado. Los MDPs, como el nombre lo indica, son procesos markovianos; es decir, que el siguiente estado es independiente de los estados anteriores dado el estado actual:

$$\mathcal{P}^a_{ss'} = P(s_{t+1} = s' \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = P(s_{t+1} = s' \mid s_t = s, a_t = a)$$

La política π es un mapeo de cada estado $s \in S$ y acción $a \in A(s)$ a la probabilidad $\pi(s,a)$ de tomar la acción a estando en estado s. El *valor* de un estado s bajo la política π, denotado como $V^\pi(s)$, es la recompensa total esperada en el estado s y siguiendo la política π. El valor esperado se puede expresar como:

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi\left\{\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

y el valor esperado tomando una acción a en estado s bajo la política π, denotado como $Q^\pi(s,a)$, es:

$$Q^\pi(s,a) = E_\pi\{R_t \mid s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

Las funciones de valor óptimas se definen como:

$$V^*(s) = max_\pi V^\pi(s) \text{ y } Q^*(s,a) = max_\pi Q^\pi(s,a)$$

Las cuales se pueden expresar mediante las ecuaciones de optimalidad de Bellman [3]:

$$V^*(s) = max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$$

y

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$$

o

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \text{max}_{a'} Q^*(s',a')]$$

De esta forma, lo que se busca al resolver un MDP es encontrar la solución a las ecuaciones de Bellman; que equivale a encontrar la política óptima. Existen dos enfoques básicos para resolver MDPs: (i) cuando el modelo es conocido $(\mathcal{P}, \mathcal{R})$, mediante métodos de programación dinámica o programación lineal, y (ii) cuando el modelo es desconocido usando métodos de Monte Carlo o de aprendizaje por refuerzo. También existen métodos híbridos que combinan ambos enfoques. En la sección 10.3 estudiamos los diferentes métodos de solución.

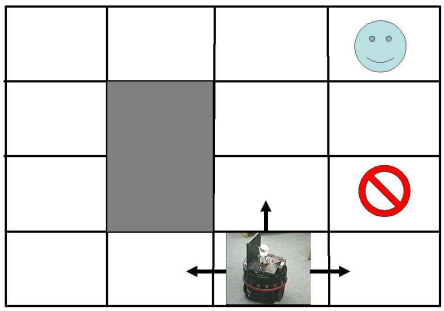

Figura 10.1: Ejemplo del mundo virtual en que el agente debe navegar. La celda de arriba a la derecha (S_4) representa la meta, con una recompensa positiva; mientras que las celdas grises (obstáculos) y la celda con el signo de *prohibido* (peligro), tienen recompensas negativas. El agente (en el dibujo en el estado s_{15}) puede desplazarse a las 4 celdas vecinas, excepto si está en la *orilla* del mundo virtual.

Ejemplo de un MDP

Veremos ahora un ejemplo muy sencillo de como podemos definir un MDP. Supongamos que en un mundo virtual en forma de rejilla (ver figura 10.1) hay un agente que se desplaza de un cuadro a otro, pudiendo moverse a los cuadros continuos: arriba, abajo, derecha o izquierda. El agente desea desplazarse para llegar a cierta meta (recompensa positiva), evitando los obstáculos en el camino (recompensa negativa). Dado que cada movimiento implica cierto gasto de energía para el agente, cada movimiento tiene asociado cierto costo. Como existe incertidumbre en las acciones del agente, asumimos que con un 80% de probabilidad llega a la celda desaeada, y con un 20% de probabilidad cae en otra celda contigua.

En base a la descripción anterior, considerando un mundo de 4×4 celdas, podemos definir un MDP que representa este problema:
Estados:

$$S = \left\{ \begin{array}{cccc} s_1 & s_2 & s_3 & s_4 \\ s_5 & s_6 & s_7 & s_8 \\ s_9 & s_{10} & s_{11} & s_{12} \\ s_{13} & s_{14} & s_{15} & s_{16} \end{array} \right\}$$

Acciones:

$$A = \{izquierda, arriba, derecha, abajo\}$$

Recompensas:

$$R = \left\{ \begin{array}{cccc} -1 & -1 & -1 & 10 \\ -1 & -100 & -1 & -1 \\ -1 & -100 & -1 & -10 \\ -1 & -1 & -1 & -1 \end{array} \right\}$$

La *meta* tiene una recompensa positiva (+10), los obstáculos (-100) y la zona de peligro (-10) negativas. Las demás celdas tiene una pequeña recompensa negativas (-1) que indican el costo de moverse del agente, de forma que lo hacen buscar una trayectoria "corta" a la meta.
Función de transición:

$\Phi(s_1, izquierda, \_) = 0.0$ % no es posible irse a la izquierda
$\ldots$ $\ldots$
$\Phi(s_2, izquierda, s_1) = 0.8$ % llega al estado deseado, s_1
$\Phi(s_2, izquierda, s_5) = 0.1$ % llega al estado s_5
$\Phi(s_2, izquierda, s_2) = 0.1$ % se queda en el mismo estado
$\ldots$ $\ldots$

Una vez definido el modelo, se obtiene la política óptima utilizando los métodos que veremos más adelante.

10.2.2 POMDPs

Un MDP asume que el estado es perfectamente observable; es decir, que el agente conoce con certeza el estado donde se encuentra en cada instante de tiempo. Pero en la práctica, en muchos casos no es así. Por ejemplo, considerando un robot móvil cuyo estado esta dado por su posición y orientación en el plano, (x, y, θ), es común que la posición no se conozca con certeza, por limitaciones de la odometría y los sensores del robot. Entonces el robot tiene cierta *creencia* sobre su posición actual (su estado), que se puede representar como una distribución de probabilidad sobre los posibles estados. Cuando el estado no es conocido con certeza, el problema se transforma en un proceso de decisión de Markov parcialmente observable (POMDP).

En forma similar a un MDP, un POMDP se define formalmente como una tupla $M =< S, A, \Phi, R, O >$. Los elementos S, A, Φ, R se definen de la misma forma que para un MDP. El elemento adicional es la Función de Observación, O, que da la probabilidad de observar o dado que el robot se encuentra en el estado s y ejecuto la acción a, $O(s, a, o)$.

Resolver un POMDP es mucho más complejo que un MDP, ya que al considerar el estado de creencia (la distribución de probabilidad sobre los estados), el espacio de estados de creencia se vuelve infinito, y la solución exacta mucho más compleja. El espacio de estados de creencia (*belief state*) considera la distribución de probabilidad de los estados dadas las observaciones, $Bel(S)$. Cada asignación de probabilidades a los estados originales es un estado de creencia, por lo que el número es infinito.

Se puede resolver un POMDP en forma exacta considerando un horizonte finito y un espacio de estados originales *pequeño*. Para problemas más complejos o de horizonte infinito, se han propuesto diferentes esquemas de solución aproximada [20, 39, 38]:

- Asumir que el agente se encuentra en el estado más probable se transforma en un MDP que se puede resolver por cualquiera de los métodos de solución para MDPs.
- Proyectar el estado de creencia a un espacio de menor dimensión, considerando el estado más probable con cierta medida de incertidumbre.
- Utilizar métodos de simulación Montecarlo, como los filtros de partículas.
- Considerar un número finito de pasos y modelar el problema como una red de decisión dinámica, donde la aproximación depende del número de estados que se "ven" hacia delante o *lookahead* (ver sección 10.5).

En este capítulo nos enfocamos solamente a MDPs, ahora veremos los métodos básicos de solución.

10.3 Métodos de Solución Básicos

Existen tres formas principales de resolver MDPs: (i) usando métodos de programación dinámica, (ii) usando métodos de Monte Carlo, y (iii) usando métodos de diferencias temporales o de aprendizaje por refuerzo.

10.3.1 Programación Dinámica (DP)

Si se conoce el modelo del ambiente; es decir, las funciones de probabilidad de las transiciones ($\mathcal{P}^a_{ss'}$) y los valores esperados de recompensas ($\mathcal{R}^a_{ss'}$), las

ecuaciones de optimalidad de Bellman nos representan un sistema de $|S|$ ecuaciones y $|S|$ incógnitas.

Consideremos primero como calcular la función de valor V^π dada una política arbitraria π.

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t \mid s_t = s\} \\ &= E_\pi\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s_t = s\right\} \\ &= E_\pi\left\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\right\} \\ &= \textstyle\sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')] \end{aligned}$$

donde $\pi(s,a)$ es la probabilidad de tomar la acción a en estado s bajo la política π.

Podemos hacer aproximaciones sucesivas, evaluando $V_{k+1}(s)$ en términos de $V_k(s)$:

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

Podemos entonces definir un algoritmo de evaluación iterativa de políticas como se muestra en el Algoritmo 1, en donde, para una política dada, se evalúan los valores V hasta que no cambian dentro de una cierta tolerancia θ.

Algoritmo 1 Algoritmo iterativo de evaluación de política.

Inicializa $V(s) = 0$ para toda $s \in S$
repeat
  $\Delta \leftarrow 0$
  for all $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$ (número positivo pequeño)
regresa $V \approx V^\pi$

Una de las razones para calcular la función de valor de una política es para tratar de encontrar mejores políticas. Dada una función de valor para una política dada, podemos probar una acción $a \neq \pi(s)$ y ver si su $V(s)$ es mejor o peor que el $V^\pi(s)$.

En lugar de hacer un cambio en un estado y ver el resultado, se pueden considerar cambios en todos los estados considerando todas las acciones de

cada estado, seleccionando aquellas que parezcan mejor de acuerdo a una política *greedy* (seleccionado siempre el mejor). Podemos entonces calcular una nueva política $\pi'(s) = \text{argmax}_a Q^\pi(s,a)$ y continuar hasta que no mejoremos. Esto sugiere, partir de una política (π_0) y calcular la función de valor (V^{π_0}), con la cual encontrar una mejor política (π_1) y así sucesivamente hasta converger a π^* y V^*. A este procedimiento se le llama iteración de políticas y viene descrito en el Algorithmo 2 [19].

Algoritmo 2 Algoritmo de iteración de política.

1. Inicialización:
$V(s) \in \mathcal{R}$ y $\pi(s) \in \mathcal{A}(s)$ arbitrariamente $\forall s \in S$
2. Evaluación de política:
repeat
  $\Delta \leftarrow 0$
  for all $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)}[\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s')]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$ (número positivo pequeño)
3. Mejora de política:
pol-estable $\leftarrow$ true
for all $s \in S$: **do**
  $b \leftarrow \pi(s)$
  $\pi(s) \leftarrow \text{argmax}_a \sum_{s'} \mathcal{P}_{ss'}^{a}[\mathcal{R}_{ss'}^{a} + \gamma V(s')]$
  if $b \neq \pi(s)$ **then**
    pol-estable $\leftarrow$ false
  end if
end for
if *pol-estable* **then**
  stop
else
  go to 2
end if

Uno de los problemas con el algoritmo de iteración de políticas es que cada iteración involucra evaluación de políticas que requiere recorrer todos los estados varias veces. Sin embargo, el paso de evaluación de política lo

podemos truncar de varias formas, sin perder la garantía de convergencia. Una de ellas es pararla después de recorrer una sola vez todos los estados. A esta forma se le llama iteración de valor (*value iteration*). En particular, se puede escribir combinando la mejora en la política y la evaluación de la política truncada como sigue:

$$V_{k+1}(s) = max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

Esto último se puede ver como expresar la ecuación de Bellman en una regla de actualización. Es muy parecido a la regla de evaluación de políticas, solo que se evalúa el máximo sobre todas las acciones (ver Algoritmo 3).

Algoritmo 3 Algoritmo de iteración de valor.

Inicializa $V(s) = 0$ para toda $s \in S$
repeat
  $\Delta \leftarrow 0$
  for all $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$ (número positivo pequeño)
Regresa una política determinísta
Tal que: $\pi(s) = argmax_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$

Para espacios muy grandes, el ver todos los estados puede ser computacionalmente muy caro. Una opción es hacer estas actualizaciones al momento de estar explorando el espacio, y por lo tanto determinando sobre qué estados se hacen las actualizaciones. El hacer estimaciones en base a otras estimaciones se conoce también como *bootstrapping*.

10.3.2 Monte Carlo (MC)

Los métodos de Monte Carlo, solo requieren de experiencia y la actualización se hace por episodio en lugar de por cada paso. La función de valor de un estado es la recompensa esperada que se puede obtener a partir de ese estado. Para estimar V^π y Q^π podemos tomar estadísticas haciendo un promedio de las recompensas obtenidas. El procedimiento para V^π está descrito en el

Algoritmo 4, donde se guardan los promedios de las recompenzas totales obtenidas en cada estado que se visita en la simulación.

Algoritmo 4 Algoritmo de Monte Carlo para estimar V^{π}.

loop
  Genera un episodio usando π
  for all estado s en ese episodio: **do**
    $R \leftarrow$ recompensa después de la primera ocurrencia de s
    Añade R a $recomp(s)$
    $V(s) \leftarrow$ promedio($recomp(s)$)
  end for
end loop

Para estimar pares estado-acción (Q^{π}) corremos el peligro de no ver todos los pares, por lo que se busca mantener la exploración. Lo que normalmente se hace es considerar solo políticas estocásticas; es decir, que tienen una probabilidad diferente de cero de seleccionar todas las acciones.

Con Monte Carlo podemos alternar entre evaluación y mejoras en base a cada episodio. La idea es que después de cada episodio las recompensas observadas se usan para evaluar la política y la política se mejora para todos los estados visitados en el episodio. El algoritmo viene descrito en el Algoritmo 5.

Algoritmo 5 Algoritmo de Monte Carlo para mejorar políticas.

loop
  Genera un episodio usando π con exploración
  for all par s, a en ese episodio: **do**
    $R \leftarrow$ recompensa después de la primera ocurrencia de s, a
    Añade R a $recomp(s, a)$
    $Q(s, a) \leftarrow$ promedio($recomp(s, a)$)
  end for
  for all s en el episodio: **do**
    $\pi(s) \leftarrow \text{argmax}_a Q(s, a)$
  end for
end loop

Para seleccionar una acción durante el proceso de aprendizaje se puede hacer de acuerdo a lo que diga la política o usando un esquema de selección

diferente. Dentro de estos últimos, existen diferentes formas para seleccionar acciones, dos de las más comunes son:

- $\epsilon-$*greedy*: en donde la mayor parte del tiempo se selecciona la acción que da el mayor valor estimado, pero con probabilidad ϵ se selecciona una acción aleatoriamente.

- *softmax*, en donde la probabilidad de selección de cada acción depende de su valor estimado. La más común sigue una distribución de Boltzmann o de Gibbs, y selecciona una acción con la siguiente probabilidad:

$$\frac{e^{Q_t(s,a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(s,b)/\tau}}$$

  donde τ es un parámetro positivo (temperatura) que se disminuye con el tiempo.

Los algoritmos de aprendizaje los podemos dividir en base a la forma que siguen para seleccionar sus acciones durante el proceso de aprendizaje:

- Algoritmos *on-policy*: Estiman el valor de la política mientras la usan para el control. Se trata de mejorar la política que se usa para tomar decisiones.

- Algoritmos *off-policy*: Usan la política y el control en forma separada. La estimación de la política puede ser por ejemplo *greedy* y la política de comportamiento puede ser ϵ-*greedy*. Osea que la política de comportamiento está separada de la política que se quiere mejorar. Esto es lo que hace Q-learning, el cual describiremos en la siguiente sección.

10.3.3 Aprendizaje por Refuerzo (RL)

Los métodos de Aprendizaje por Refuerzo (*Reinforcement Learning*) o diferencias temporales, combinan las ventajas de los dos enfoques anteriores: permiten hacer *bootstrapping* (como DP) y no requiere tener un modelo del ambiente (como MC).

Métodos tipo RL sólo tienen que esperar el siguiente paso para hacer una actualización en la función de valor. Básicamente usan el error o diferencia entre predicciones sucesivas (en lugar del error entre la predicción y la salida

final) para aprender. Su principal ventaja es que son incrementales y por lo tanto fáciles de computar.

El algoritmo de diferencias temporales más simple es TD(0), el cual viene descrito en el Algoritmo 6 y cuya función de actualización es:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

Algoritmo 6 Algoritmo TD(0).

```
Inicializa V(s) arbitrariamente y π a la política a evaluar
for all episodio do
  Inicializa s
  for all paso del episodio (hasta que s sea terminal) do
    a ← acción dada por π para s
    Realiza acción a; observa recompensa, r, y siguiente estado, s'
    V(s) ← V(s) + α [r + γV(s') − V(s)]
    s ← s'
  end for
end for
```

La actualización de valores tomando en cuenta la acción sería:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

y el algoritmo es prácticamente el mismo, solo que se llama SARSA (State - Action - Reward - State' - Action'), y viene descrito en el Algoritmo 7.

Uno de los desarrollos más importantes en aprendizaje por refuerzo fué el desarrollo de un algoritmo "fuera-de-política" (*off-policy*) conocido como Q-learning. La idea principal es realizar la actualización de la siguiente forma (Watkins, 89):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

El algoritmo viene descrito en el Algoritmo 8.

10.4 Técnicas de Soluciones Avanzadas

Se han propuesto diferentes algoritmos de solución que combinan o proponen alguna solución intermedia entre algunas de las técnicas vistas en la

Algoritmo 7 Algoritmo SARSA.

Inicializa $Q(s, a)$ arbitrariamente
for all episodio **do**
  Inicializa s
  Selecciona una a a partir de s usando la política dada por Q (e.g., ϵ–greedy)
  for all paso del episodio (hasta que s sea terminal) **do**
    Realiza acción a, observa r, s'
    Escoge a' de s' usando la política derivada de Q
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  end for
end for

Algoritmo 8 Algoritmo Q-Learning.

Inicializa $Q(s, a)$ arbitrariamente
for all episodio **do**
  Inicializa s
  for all paso del episodio (hasta que s sea terminal) **do**
    Selecciona una a de s usando la política dada por Q (e.g., ϵ–greedy)
    Realiza acción a, observa r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma max'_a Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$;
  end for
end for

sección 10.3. En particular entre Monte Carlo y Aprendizaje por Refuerzo (trazas de elegibilidad) y entre Programación Dinámica y Aprendizaje por Refuerzo (Dyna-Q y *Prioritized Sweeping*). En las siguientes secciones veremos estas técnicas.

10.4.1 Trazas de Elegibilidad (*eligibility traces*)

Esta técnica se puede considerar que está entre los métodos de Monte Carlo y RL de un paso. Los métodos Monte Carlo realizan la actualización considerando la secuencia completa de recompensas observadas, mientras que la actualización de los métodos de RL se hace utilizando únicamente la siguiente recompensa. La idea de las trazas de elegibilidad es considerar las recompensas de n estados posteriores (o afectar a n anteriores).

Si recordamos la recompenza total acumulada es:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-t-1} r_T$$

Lo que se hace en RL es usar:

$$R_t = r_{t+1} + \gamma V_t(s_{t+1})$$

lo cual hace sentido porque $V_t(s_{t+1})$ reemplaza a los términos siguientes $(\gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots)$. Sin embargo, hace igual sentido hacer:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$$

y, en general, para n pasos en el futuro.

En la práctica, más que esperar n pasos para actualizar (*forward view*), se realiza al revés (*backward view*). Se guarda información sobre los estados por los que se pasó y se actualizan hacia atrás las recompensas (descontadas por la distancia). Se puede probar que ambos enfoques son equivalentes.

Para implementar la idea anterior, se asocia a cada estado o par estado-acción una variable extra, representando su traza de elegibilidad (*eligibility trace*) que denotaremos por $e_t(s)$ o $e_t(s, a)$. Este valor va decayendo con la longitud de la traza creada en cada episodio. Para $TD(\lambda)$:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{si } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{si } s = s_t \end{cases}$$

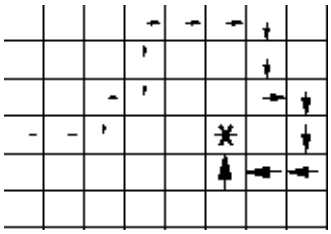

Figura 10.2: Comportamiento de las recompensas dadas a un cierto camino dadas por las trazas de elegibilidad.

Para SARSA se tiene lo siguiente:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{si } s \neq s_t \\ \gamma\lambda e_{t-1}(s,a) + 1 & \text{si } s = s_t \end{cases}$$

$SARSA(\lambda)$ viene descrito en el Algoritmo 9, donde todos los estados visitados reciben parte de las recompensas futuras dependiendo de su distancia, como se ilustra en la figura 10.2.

Algoritmo 9 $SARSA(\lambda)$ con trazas de elegibilidad.

```
Inicializa Q(s,a) arbitrariamente y e(s,a) = 0 ∀s,a
for all episodio (hasta que s sea terminal) do
  Inicializa s,a
  for all paso en el episodeo do
    Toma acción a y observa r,s'
    Selecciona a' de s' usando una política derivada de Q (e.g., ε−greedy)
    δ ← r + γQ(s',a') − Q(s,a)
    e(s,a) ← e(s,a) + 1
    for all (s,a) do
      Q(s,a) ← Q(s,a) + αδe(s,a)
      e(s,a) ← γλe(s,a)
      s ← s'; a ← a'
    end for
  end for
end for
```

Para Q-learning como la selección de acciones se hace, por ejemplo, siguiendo una política $\epsilon-greedy$, se tiene que tener cuidado, ya que a veces los

movimientos, son movimientos exploratorios de los cuales no necesariamente queremos propagar sus recompensas hacia atrás. Aquí se puede mantener historia de la traza solo hasta el primer movimiento exploratorio, ignorar las acciones exploratorias, o hacer un esquema un poco más complicado que considera todas las posibles acciones en cada estado.

10.4.2 Planificación y Aprendizaje

Asumamos que tenemos un modelo del ambiente, esto es, que podemos predecir el siguiente estado y la recompensa dado un estado y una acción. La predicción puede ser un conjunto de posibles estados con su probabilidad asociada o puede ser un estado que es muestreado de acuerdo a la distribución de probabilidad de los estados resultantes.

Dado un modelo, es posible hacer planificación. Lo interesante es que podemos utilizar los estados y acciones utilizados en la planificación también para aprender. De hecho al sistema de aprendizaje no le importa si los pares estado-acción son dados de experiencias reales o simuladas.

Dado un modelo del ambiente, uno podría seleccionar aleatoriamente un par estado-acción, usar el modelo para predecir el siguiente estado, obtener una recompensa y actualizar valores Q. Esto se puede repetir indefinidamente hasta converger a Q^*.

El algoritmo Dyna-Q combina experiencias con planificación para aprender más rápidamente una política óptima. La idea es aprender de experiencia, pero también usar un modelo para simular experiencia adicional y así aprender más rápidamente (ver Algoritmo 10). La experiencia acumulada también se usa para actualizar el modelo.

El algoritmo de Dyna-Q selecciona pares estado-acción aleatoriamente de pares anteriores. Sin embargo, la planificación se puede usar mucho mejor si se enfoca a pares estado-acción específicos. Por ejemplo, enfocarnos en las metas e irnos hacia atrás o más generalmente, irnos hacia atrás de cualquer estado que cambie su valor. Esto es, enfocar la simulación al estado que cambio su valor. Esto nos lleva a todos los estados que llegan a ese estado y que también cambiarían su valor. Esto proceso se puede repetir sucesivamente, sin embargo, algunos estados cambian mucho más que otros. Lo que podemos hacer es ordenarlos y realizar las simulaciones con el modelo solo en los estados que rebacen un cierto umbral. Esto es precisamente lo que hacer el algoritmo de *prioritized sweeping* descrito en el Algoritmo 11.

Algoritmo 10 Algoritmo de Dyna-Q.

Inicializa $Q(s,a)$ y $Modelo(s,a)$ $\forall s \in S, a \in A$
loop
  $s \leftarrow$ estado actual
  $a \leftarrow \epsilon-greedy(s,a)$
  realiza acción a observa s' y r
  $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
  $Modelo(s,a) \leftarrow s', r$
  for N veces **do**
    $s \leftarrow$ estado anterior seleccionado aleatoriamente
    $a \leftarrow$ acción aleatoria tomada en s
    $s', r \leftarrow Modelo(s,a)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
  end for
end loop

10.4.3 Generalización en Aprendizaje por Refuerzo

Hasta ahora hemos asumido que se tiene una representación explícita de las funciones de valor en forma de tabla (i.e., una salida por cada tupla de entradas). Esto funciona para espacios pequeños, pero es impensable para dominios como ajedrez (10^{120} estados) o backgammon (10^{50} estados). Una forma para poder aprender una función de valor en espacios grandes, es hacerlo con una representación implícita, i.e., una función. Por ejemplo en juegos, una función de utilidad estimada de un estado se puede representar como una función lineal pesada sobre un conjunto de atributos (f_i's):

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

En ajedrez se tienen aproximadamente 10 pesos, por lo que es una compresión bastante significativa. La compresión lograda por una representación implícita permite al sistema de aprendizaje, generalizar de estados visitados a estados no visitados. Por otro lado, puede que no exista tal función. Como en todos los sistemas de aprendizaje, existe un balance entre el espacio de hipótesis y el tiempo que se toma aprender una hipótesis aceptable. Muchos sistemas de aprendizaje supervisado tratan de minimizar el error cuadratico medio (MSE) bajo cierta distribución $P(s)$ de las entradas. Supongamos que $\vec{\Theta}_t$ representa el vector de parámetros de la función parametrizada que

Algoritmo 11 Algoritmo de Prioritized sweeping.

Inicializa $Q(s,a)$ y $Modelo(s,a)$ $\forall s \in S, a \in A$ y $ColaP = \emptyset$
loop
  $s \leftarrow$ estado actual
  $a \leftarrow \epsilon{-}greedy(s,a)$
  realiza acción a observa s' y r
  $Modelo(s,a) \leftarrow s', r$
  $p \leftarrow\mid r + \gamma \max_{a'} Q(s',a') - Q(s,a) \mid$
  if $p > \theta$ **then**
    inserta s, a a $ColaP$ con prioridad p
  end if
  for N veces, mientras $ColaP \neq \emptyset$ **do**
    $s, a \leftarrow$ primero$(ColaP)$
    $s', r \leftarrow Modelo(s,a)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    for all $\overline{s}, \overline{a}$ que se predice llegan a s **do**
      $\overline{r} \leftarrow$ recompensa predicha
      $p \leftarrow\mid \overline{r} + \gamma \max_{a} Q(s,a) - Q(\overline{s}, \overline{a}) \mid$
      if $p > \theta$ **then**
        inserta $\overline{s}, \overline{a}$ a $ColaP$ con prioridad p
      end if
    end for
  end for
end loop

queremos aprender, el error cuadrado se calcula como la diferencia entre el valor de la política real y el que nos da nuestra función:

$$MSE(\vec{\Theta}_t) = \sum_{s \in S} P(s)[V^{\pi(s)} - V_t(s)]^2$$

donde $P(s)$ es una distribución pesando los errores de diferentes estados. Cuando los estados son igualmente probables, podemos eliminar este término. Para ajustar los parámetros del vector de la función que queremos optimizar, se obtiene el gradiente y se ajustan los valores de los parámetros en la dirección que produce la máxima reducción en el error, esto es, ajustamos los parámetros de la función restandoles la derivada del error de la función con respecto a estos parámetros:

$$\begin{aligned} \vec{\Theta}_{t+1} &= \vec{\Theta}_t - \tfrac{1}{2}\alpha \nabla_{\vec{\Theta}_t}[V^{\pi(s_t)} - V_t(s_t)]^2 \\ &= \vec{\Theta}_t + \alpha[V^{\pi(s_t)} - V_t(s_t)]\nabla_{\vec{\Theta}_t} V_t(s_t) \end{aligned}$$

donde α es un parámetro positivo $0 \leq \alpha \leq 1$ que determina que tanto cambiamos los parámetros en función del error, y $\nabla_{\vec{\Theta}_t} f(\Theta_t)$ denota un vector de derivadas parciales de la función con respecto a los parámetros. Si representamos la función con n parámetros tendríamos:

$$\nabla_{\vec{\Theta}_t} V_t(s_t) = \left(\frac{\partial f(\vec{\Theta}_t)}{\partial \Theta_t(1)}, \frac{\partial f(\vec{\Theta}_t)}{\partial \Theta_t(2)}, \ldots, \frac{\partial f(\vec{\Theta}_t)}{\partial \Theta_t(n)} \right)^T$$

Como no sabemos $V^{\pi(s_t)}$ la tenemos que aproximar. Una forma de hacerlo es con trazas de elegibilidad y actualizar la función Θ usando información de un función de valor aproximada usando la recompensa obtenida y la función de valor en el siguiente estado, como sigue:

$$\vec{\Theta}_{t+1} = \vec{\Theta}_t + \alpha \delta_t \vec{e}_t$$

donde δ_t es el error:

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

y $\vec{e}_t$ es un vector de trazas de elegibilidad, una por cada componente de $\vec{\Theta}_t$, que se actualiza como:

$$\vec{e}_t = \gamma \lambda \vec{e}_{t-1} + \nabla_{\vec{\Theta}_t} V_t(s_t)$$

con $\vec{e}_0 = 0$. Como estamos usando trazas de elegibilidad, la aproximación de la función de valor se hace usando información de todo el camino recorrido y no sólo de la función de valor inmediata.

10.5 Representaciones Factorizadas y Abstractas

Uno de los principales problemas de los MDPs es que el espacio de estados y acciones puede crecer demasiado (miles o millones de estados, decenas de acciones). Aunque lo métodos de solución, como iteración de valor o política, tienen una complejidad polinomial, problemas tan grandes se vuelven intratables tanto en espacio como en tiempo. Para ello se han propuesto diversos esquemas para reducir o simplificar el modelo, en algunos casos sacrificando optimalidad. Estos métodos los podemos dividir en 3 tipos principales:

1. Factorización. El estado se descompone en un conjunto de variables, reduciendo el espacio para representar las funciones de transición y de recompensa.

2. Abstracción. Se agrupan estados con propiedades similares, o se utilizan representaciones relacionales basadas en lógica de predicados; reduciendo en ambos casos el espacio de estados.

3. Descomposición. El problema se divide en varios sub–problemas, que puedan ser resueltos independientemente, para luego conjuntar las soluciones.

En esta sección veremos las dos primeras estrategias, y en la siguiente hablaremos sobre decomposición. Antes introducimos las redes bayesianas, en las que se basan los MDPs factorizados.

10.5.1 Redes Bayesianas

Las redes bayesianas (RB) modelan un fenómeno mediante un conjunto de variables y las relaciones de dependencia entre ellas. Dado este modelo, se puede hacer inferencia bayesiana; es decir, estimar la probabilidad posterior de las variables no conocidas, en base a las variables conocidas. Estos modelos pueden tener diversas aplicaciones, para clasificación, predicción, diagnóstico, etc. Además, pueden dar información interesante en cuanto a cómo se relacionan las variables del dominio, las cuales pueden ser interpretadas en ocasiones como relaciones de causa–efecto. En particular, nos interesan las RB ya que permiten representar de un forma más compacta las funciones de transición de los MDPs.

Representación

Las redes bayesianas [28] son una representación gráfica de dependencias para razonamiento probabilístico, en la cual los nodos representan variables aleatorias y los arcos representan relaciones de dependencia directa entre las variables. La Figura 10.3 muestra un ejemplo hipotético de una red bayesiana (RB) que representa cierto conocimiento sobre medicina. En este caso, los nodos representan enfermedades, síntomas y factores que causan algunas enfermedades. La variable a la que apunta un arco es dependiente de la que está en el origen de éste, por ejemplo *Fiebre* depende de *Tifoidea* y *Gripe* en la red de la Figura 10.3. La topología o estructura de la red nos da información sobre las dependencias probabilísticas entre las variables. La red también representa las independencias condicionales de una variable (o conjunto de variables) dada(s) otra(s) variable(s). Por ejemplo, en la red de la Figura 10.3, *Reacciones* (R) es condicionalmente independiente de *Comida* (C) dado *Tifoidea* (T):

$$P(R|C,T) = P(R|T) \tag{10.1}$$

Esto se representa gráficamente por el nodo T *separando* al nodo C de R.

Complementa la definición de una red bayesiana las probabilidades condicionales de cada variable dados sus padres:

- Nodos raíz: vector de probabilidades marginales.
- Otros nodos: matriz de probabilidades condicionales dados sus padres.

Aplicando la regla de la cadena y las independencias condicionales, se puede verificar que con dichas probabilidades se puede calcular la probabilidad conjunta de todas las variables; como el producto de las probabilidades de cada variable (X_i) dados sus padres ($Pa(X_i)$):

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i|Pa(X_i)) \tag{10.2}$$

La Figura 10.3 ilustra algunas de las matrices de probabilidad asociadas al ejemplo de red bayesiana.

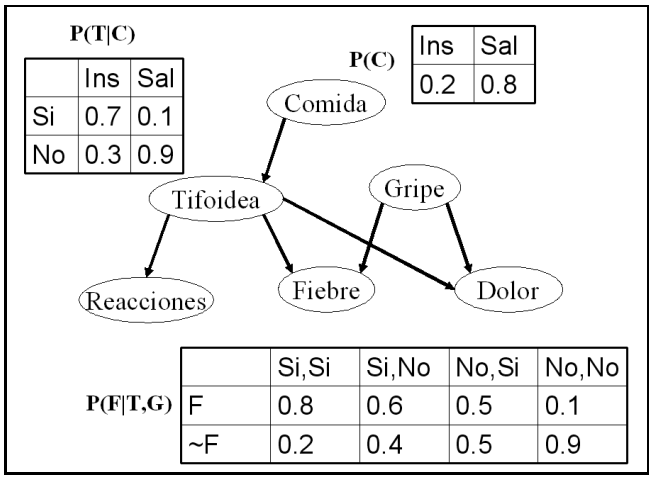

Figura 10.3: Ejemplo de una red bayesiana. Los nodos representan variables aleatorias y los arcos relaciones de dependencia. Se muestran las tablas de probabilidad condicional de algunas de las variables de la red bayesiana: probabilidad *a priori* de Comida, $P(C)$; probabilidad de Tifoidea dada Comida, $P(T \mid C)$; y probabilidad de Fiebre dada Tifoidea y Gripe, $P(F \mid T, G)$. En este ejemplo se asume que todas las variables son binarias.

Redes bayesianas dinámicas

Las redes bayesianas, en principio, representan el estado de las variables en un cierto momento en el tiempo. Para representar procesos dinámicos existe una extensión a estos modelos conocida como *red bayesiana dinámica* (RBD) [26]. Una RBD consiste en una representación de los estados del proceso en cierto tiempo (red estática) y las relaciones temporales entre dichos procesos (red de transición). Para las RBDs generalmente se hacen las siguientes suposiciones:

- Proceso markoviano. El estado actual sólo depende del estado anterior (sólo hay arcos entre tiempos consecutivos).
- Proceso estacionario en el tiempo. Las probabilidades condicionales en el modelo no cambian con el tiempo.

Lo anterior implica que podemos definir una red bayesiana dinámica en base a dos componentes: (i) una red base estática que se repite en cada periodo, de acuerdo a cierto intervalo de tiempo predefinido; y (ii) una red de transición entre etapas consecutivas (dada la propiedad markoviana). Un ejemplo de una RBD se muestra en la Figura 10.4.

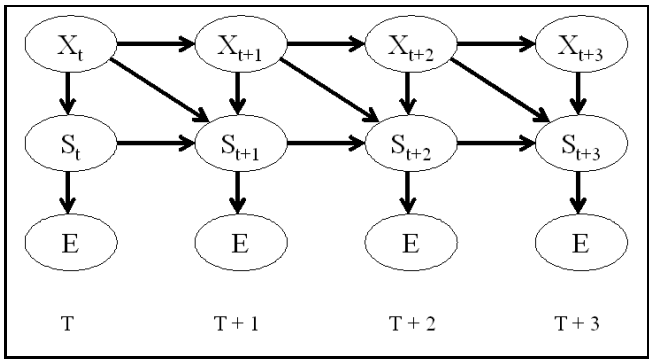

Figura 10.4: Ejemplo de una red bayesiana dinámica. Se muestra la estructura base que se repite en cuatro etapas temporales, así como las relaciones de dependencia entre etapas.

Estos modelos, en particular las redes bayesianas dinámicas, permiten representar en una forma mucho más compacta las funciones de transición en MDPs, como veremos en la siguiente sección.

10.5.2 MDPs Factorizados

Al aumentar el número de estados en un MDP, crecen las tablas para representar las funciones de transición, las cuales se pueden volver inmanejables. Por ejemplo, si tenemos un MDP con 1000 estados (algo común en muchas aplicaciones), la tabla de transición, por acción, tendría un millón (1000×1000) de parámetros (probabilidades). Una forma de reducir esta complejidad es mediante el uso de redes bayesianas dinámicas, en lo que se conoce como *MDPs factorizados* [4].

En los MDPs factorizados, el estado de descompone en un conjunto de variables o factores. Por ejemplo, en vez de representar el estado de un robot *mensajero* mediante una sóla variable, S, con un gran número de valores; podemos descomponerla en n variables, $x_1, x_2, \ldots, x_n$, donde cada una representa un aspecto del estado (posición del robot, si tiene un mensaje que entregar, posición de la meta, etc.). Entonces el modelo de transición se representa usando RBDs. Se tiene una RBD con dos etapas temporales por acción, que especifican las variables en el tiempo t y en el tiempo $t + 1$. La estructura de la RBD establece las dependencias entre las variables. Se tiene asociada una tabla de probabilidad por variable en $t + 1$, que establece la

probabilidad condicional de la variable $x_i(t+1)$ dados sus padres, $x_{1..k}(t)$, asumiendo que tiene k padres. Un ejemplo de un modelo factorizado se muestra en la figura 10.5. La representación factorizada de las funciones de transición implica, generalmente, un importante ahorro de espacio, ya que en la práctica, tanto las funciones de transición como las de recompensa tienden a depender de pocas variables, resultando en modelos *sencillos*.

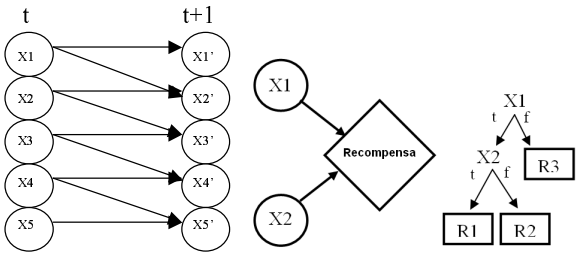

Figura 10.5: MDP factorizado. Izquierda: función de transición representada mediante una RBD, para un MDP con 5 variables de estado. Centro: la función de recompensa sólo depende de dos de las variables de estado. Derecha: función de recompensa como un árbol de decisión binario.

Al descomponer el estado en un conjunto de variables, también es posible representar la función de recompensa en forma factorizada. La función de recompensa se representa usando árboles de decisión, donde cada nodo en el árbol corresponde a una de las variables de estado, y las hojas a los diferentes valores de recompensa. La figura 10.5 ilustra un ejemplo de una función de recompensa factorizada.

En base a estas representaciones factorizadas se han desarrollado extensiones de los algoritmos básicos de solución, que son mucho más eficientes al operar directamente sobre las respresntaciones estructuradas. Dos ejemplo son el método de *iteración de política estructurado* [5] y *SPUDD* [18]. SPUDD además de usar una representación factorizada, simplifica aún más el modelo al representar las tablas de probabilidad condicional mediante grafos llamados *ADDs*.

10.5.3 Agregación de estados

Otra alternativa para reducir la complejidad es reducir el número de estados, agrupando estados con propiedades similares (por ejemplo, que tengan la misma función de valor o política). Si dos o más estados tienen la misma

recompensa y probabilidades de transición a otros estados, se dice que son *estados equivalentes*, y se pueden fusionar en un sólo estado sin pérdida de información. Al agregar estados de esta forma, se mantiene una solución óptima del MDP. Sin embargo, en la práctica, generalmente la reducción de estados es relativamente pequeña, por lo que se han planetado otras alternativas. Éstas agrupan estados aunque no sean equivalentes, pero si *parecidos*, con lo que se sacrifica optimalidad pero se gana en eficiencia, con una solución aproximada.

MDPs cualitativos

Se han propuesto varios esquemas de agregación de estados [4], veremos uno de ellos, basado en *estados cualitativos* (Q) [32]. Además de permitir reducir el número de estados en MDPs discretos, este método permite resolver MDPs continuos. El método se basa en una representación cualitativa de los estados, donde un estado cualitativo, q_i, es un grupo de estados (o una partición del espacio en el caso de MDPs continuos) que tienen la misma recompensa inmediata.

La solución de un MDP cualitativo se realiza en dos fases [32]:

Abstracción: El espacio de estados se divide en un conjunto de estados cualitativos, $q_1, \ldots, q_n$, donde cada uno tiene la misma recompensa. En el caso de un MDP discreto simplemente se agrupan todos los estados vecinos que tengan la misma recompensa. En el caso de un MDP continuo, esta partición se puede aprender mediante una exploración del ambiente, particionando el espacio de estados en secciones (hiper–rectángulos) de igual recompensa. El espacio de estados cualitativos se representa mediante un árbol de decisión, denominado *q-tree*. La figura 10.6 muestra un ejemplo sencillo en dos dimensiones de un espacio de estados cualitativos y el *q-tree* correspondiente. Este MDP cualitativo se resuelve usando métodos clásicos como iteración de valor.

Refinamiento: La agrupación incial puede ser demasiado *abstracta*, obteniendo una solución sub–óptima. Para ello se plantea una segunda fase de refinamiento, en la cual algunos estados se dividen, basado en la varianza en utilidad respecto a los estados vecinos. Para ello se selecciona el estado cualitativo con mayor varianza en valor (diferencia del valor con los estados vecinos), y se parte en dos, resolviendo el nuevo

MDP. Si la política resultante difiere de la anterior, se mantiene la partición, sino se descarta dicha partición y se marca ese estado. Este procedimiento se repite recursivamente hasta que ya no haya estado sin marcar o se llegue a un tamaño mínimo de estado (de acuerdo a la aplicación). La figura 10.7 ilustra el proceso de refinamiento para el ejemplo anterior.

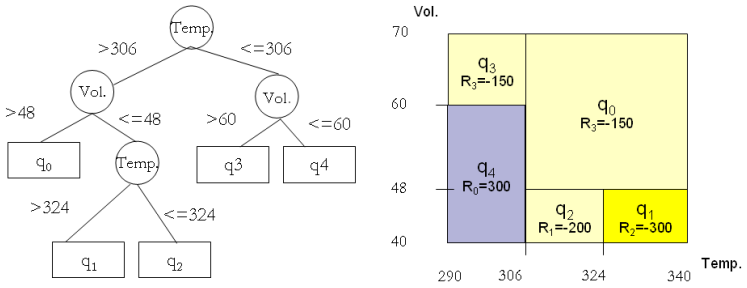

Figura 10.6: Ejemplo de una partición cualitativa en dos dimensiones, donde las variables de estado son Volumen y Temperatura. Izquierda: $q - tree$, las ramas son restricciones y las hojas los estados cualitativos. Derecha: partición del espacio bidimensional en 5 estados.

Aunque este esquema no garantiza una solución óptima, se ha encontrado experimentalmente que se obtienen soluciones aproximadamente óptimas con ahorros significativos en espacio y tiempo. En la sección 10.7 se ilustra este método en una aplicación para el control de plantas eléctricas.

10.5.4 Aprendizaje por Refuerzo Relacional

A pesar del trabajo que se ha hecho en aproximaciones de funciones (e.g., [7]), abstracciones jerárquicas (e.g., [11]) y temporales (e.g., [36]) y representaciones factorizadas(e.g., [21]), se sigue investigando en cómo tratar de forma efectiva espacios grandes de estados, cómo incorporar conocimiento del dominio y cómo transferir políticas aprendidas a nuevos problemas similares.

La mayoría de los trabajos utilizan una representación proposicional que sigue sin escalar adecuadamente en dominios que pueden ser definidos de manera más natural en términos de objetos y relaciones. Para ilustrar esto, supongamos que queremos aprender una política para jugar un final de juego

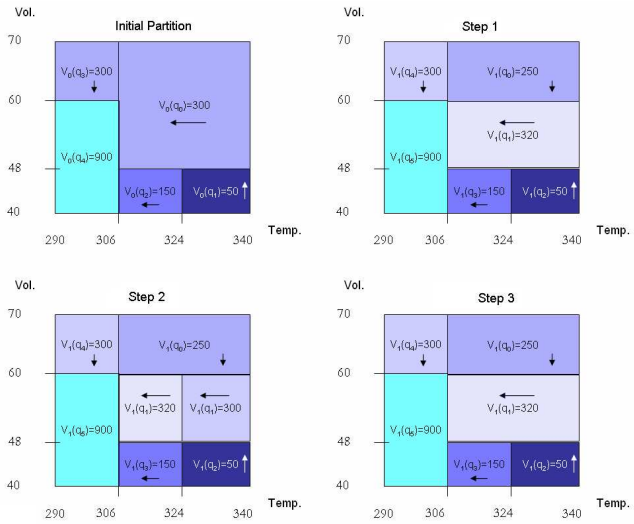

Figura 10.7: Varias etapas en el proceso de refinamiento para el ejemplo del MDP cualitativo en dos dimensiones. A partir de la partición inicial, se divide un estado cualitativo en dos (Step 1). Después se divide uno de los nuevos estados (Step 2). Al resolver el nuevo MDP se encuentra que la política no varía, por lo que se vuelven a unir en un estado (Step 3).

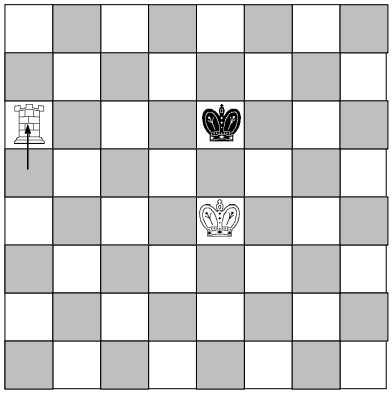

Figura 10.8: Obligando al rey contrario a moverse hacia una orilla.

sencillo en ajedrez. Inclusive para un final sencillo como Rey-Torre vs. Rey (RTR), existen más de 175,000 posiciones legales donde no existe un jaque. El número de posibles acciones por estado en general es de 22 (8 para el rey y 14 para la torre), lo cual nos da cerca de 4 millones de posibles pares estado-acción. Aprender directamente en esta representación esta fuera del alcance de computadoras estándar, sin embargo, existen muchos estados que representan escencialmente lo mismo, en el sentido que comparten las mismas relaciones. Para un jugador de ajedrez la posición exacta de las piezas no es tan importante como las relaciones existentes entre las piezas para decidir que mover (ver [8, 10]). Por ejemplo, en RTR cuando los dos reyes estan en oposición (misma fila o renglón con un cuadro entre ellos), la torre divide a los reyes y la torre está alejada al menos dos cuadros del rey contrario (i.e., puede hacer jaque en forma segura), una buena jugada es hacer jaque y forzar al rey a moverse hacia una orilla (ver Figura 10.8). Esta acción es aplicable a todas las posiciones del juego que cumplen con estas relaciones, lo cual captura más de 1,000 posiciones para una torre blanca. De hecho se puede aplicar para tableros de diferente tamaño.

La idea de usar una representación relacional es usar un conjunto reducido de estados abstractos que representan un conjunto de relaciones (e.g., en_oposición, torre_divide, etc.), un conjunto de acciones en términos de esas relaciones (e.g., If en_oposición And torre_divide Then jaque) y aprender qué acción usar para cada estado abstracto. La ventaja de una representación relacional es que puede contener variables y por lo tanto representar a muchas instancias particulares, lo cual reduce considerablemente el conjunto de estados y acciones, simplificando el proceso de aprendizaje. Por otro lado, es posible transferir las políticas aprendidas en esta representación abstracta a otras instancias del problema general.

En este capítulo seguimos la caracterización originalmente propuesta en [24]. Un MDP relacional, está definido de la misma forma por una tupla $M =< S_R, A_R, \Phi_R, R_R >$. En este caso, S_R es un conjunto de estados, cada uno definido por una conjunción de predicados en lógica de primer orden. $A_R(S_R)$ es un conjunto de acciones cuya representación es parecida a operadores tipo STRIPS. Φ_R es la función de transición que opera sobre estados, acciones y estados resultantes, todos ellos relacionales. Si existen varios posibles estados relaciones por cada acción se tienen que definir transiciones parecidas a operadores STRIPS probabilísticos. R_R es una función de recompensa que se define como en un MDP tradicional, dando un número real a cada estado $R_R : S \rightarrow \mathcal{R}$, solo que ahora se le da la misma recompensa

a todas las instancias del estado s_R. A diferencia de un MDP tradicional, el espacio de estados esta implícitamente definido.

Una vez definido un MPD relacional podemos definir también algoritmos para aprender una política óptima. En este capítulo solo describimos cómo aplicar Q-learning en esta representación, sin embargo, lo mismo puede aplicarse a otros algoritmos. El Algoritmo 12 contiene el pseudo–código para el algoritmo rQ-learning. El algoritmo es muy parecido a Q-learning, con la diferencia que los estados (S) y las acciones (A) están representados en forma relacional. El algoritmo sigue tomando acciones primitivas (a) y moviendose en estados primitivos (s), pero el aprendizaje se realiza sobre los pares estado - acción relacionales. En este esquema particular, de un estado primitivo se obtienen primero las relaciones que definen el estado relacional. En ajedrez, de la posición de las piezas del tablero se obtendrían relaciones entre las piezas para definir el estado relacional.

Algoritmo 12 El algoritmo rQ-learning.

Initialize $Q(S, A)$ arbitrarily
(where S is an *r-state* and A is an *r-action*)
for all episodio **do**
  Inicializa s
  $S \leftarrow rels(s)$ {conjunto de relaciones en estado s}
  for all paso del episodio (hasta que s sea terminal) **do**
    Selecciona A de S usando una política no determinística (e.g., ϵ-greedy)
    Selecciona aleatoriamente una acción a aplicable en A
    Toma acción a, observa r, s'
    $S' \leftarrow rels(s')$
    $Q(S, A) \leftarrow Q(S, A) + \alpha(r + \gamma max_{A'} Q(S', A') - Q(S, A))$
    $S \leftarrow S'$
  end for
end for

Esta representación relacional se ha probado en laberintos, el problema del Taxi, en el mundo de los bloques, en ajedrez para aprender a jugar un final de juego sencillo y para aprender a volar un avión (como se verá en la sección 10.7.1).

Lo bueno de aprender una política relacional es que se puede usar en diferentes instancias del problema o para problemas parecidos. La política

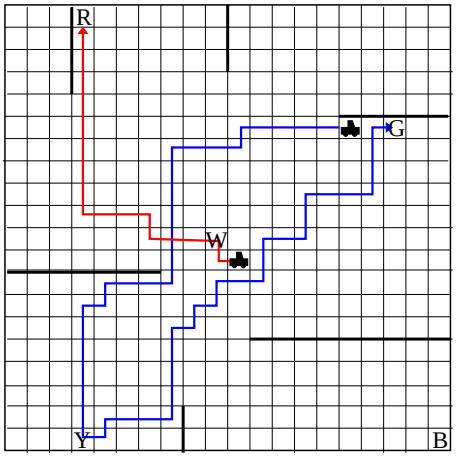

Figura 10.9: Dos caminos seguidos en dos instancias de un problema de Taxi aumentado.

aprendida en el dominio del Taxi (ver figura 10.13) usando una representación relacional, se uso como política en un espacio más grande, con nuevos destinos, cuyas posiciones son diferentes, e inclusive con paredes horizontales. En este caso, la misma política se puede aplicar y dar resultados óptimos. La figura 10.9 ilustra dos caminos tomados por el taxi para ir a un punto a recoger a un pasajero y llevarlo a otro punto seleccionado aleatoriamente.

10.6 Técnicas de Descomposición

Otra clase de técnicas de para reducir la complejidad de MDPs se base en el principio de "divide y vencerás". La idea es tratar de dividir el problema en sub–problemas que puedan ser resueltos en forma independiente, y después de alguna manera combinar las soluciones para obtener una solución global aproximada. Existen básicamente dos enfoques para descomponer un MDP:

1. Jerárquico. Se descompone el MDP en una serie de subprocesos, submetas o subrutinas, de forma que para obtener la solución global, se tienen que resolver los subprocesos, típicamente en forma secuencial. Por ejemplo, para que un robot vaya de un cuarto a otro en un edificio, el problema se puede separar en 3 sub–MDPs: salir del cuarto, ir por el pasillo al otro cuarto, y entrar al cuarto.

2. Concurrente. Se divide el MDP en un conjunto de subprocesos donde cada uno contempla un aspecto del problema, de forma que en conjunto lo resuelven en forma concurrente. Por ejemplo, en navegación robótica se podría tener un MDP que guía al robot hacia la meta y otro que evade obstáculos; al conjuntar las soluciones el robot va a la meta evitando obstáculos.

Estas técnicas sacrifican, generalmente, optimalidad para poder resolver problemas mucho más complejos. A continuación analizaremos ambos enfoques.

10.6.1 Decomposición jerárquica

Debido a que el espacio de solución al resolver un MDP tradicional crece de manera exponencial con el número de variables, se han propuesto esquemas que descomponen el problema original en sub-problemas. Las principales ventajas de esta descomposición son:

- Poder escalar los MDP a problemas más complejos.
- El poder compartir o re-utilizar soluciones de subtareas, en la solución del mismo problema o inclusive de problemas similares, transfiriendo las políticas aprendidas en las subtareas.

Por ejemplo, supongamos que se tiene una rejilla de 10 por 10, como se ilustra en la figura 10.10. En este problema existen 10,000 combinaciones de posiciones del agente y la meta (mostrados con el hexágono y cuadrado obscuros). Este implica que para resolver este problema para cualquier estado inicial y final se requieren 10,000 valores V o aproximadamente 40,000 valores Q, asumiendo que se tienen cuatro movimientos posibles (norte, sur, este y oeste). Una opción es partir en espacio en regiones, seleccionando un conjunto de puntos característicos (los círculos mostrados en la figura) y asignándoles las celdas más cercanas siguiendo un criterio de Voronoi. Con esto se puede imponer una restricción en la política siguiendo el siguiente esquema:

1. Ir del punto de inicio a la posición del punto característico de su región,
2. Ir de un punto característico de una región al punto característico de la región que tiene la meta, siguiendo los puntos característicos de las regiones intermedias,

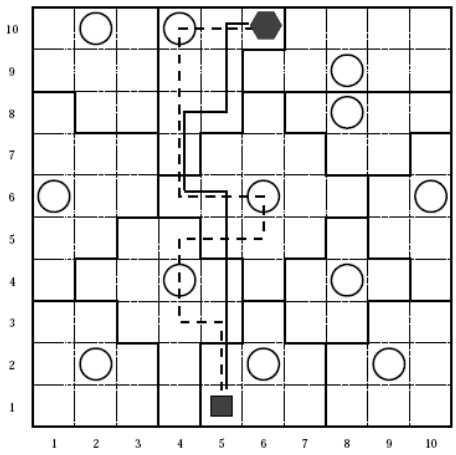

Figura 10.10: La figura muestra una descomposición de un espacio siguiendo un criterio de Voronoi, puntos característicos de cada región (círculos) y la trayectoria entre dos puntos pasando por puntos característicos.

3. Ir del punto característico de la región con el estado meta a la meta.

Lo que se tiene que aprender es como ir de cada celda a su punto característico, como ir entre puntos caracteríticos, y como ir de un punto característico a cualquier celda de su región. Para el ejemplo anterior, esto requiere 6,070 valores Q (comparados con los 40,000 originales). Esta reducción se debe a que muchas de las subtareas son compartidas por muchas combinaciones de estados inicial y final. Sin embargo, estan restricciones pueden perder la optimalidad.

En general, en las estrategias de descomposición se siguen dos enfoques:

- Una descomposición del espacio de estados y recompensas, pero no de las acciones.

- Descomposición de las acciones, en donde cada agente es responsable de parte de las acciones.

En este capítulo solo vamos a revisar algunos de los trabajos más representativos de MPDs jerárquicos, como *Options*, HAM, y MAX-Q, que caen en el primer enfoque, y descomposición concurrente, que puede incluir el segundo enfoque.

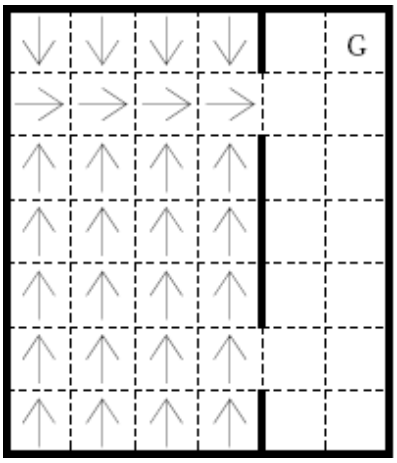

Figura 10.11: Se muestran dos regiones conectadas. Si el agente se encuentra en cualquier estado de la izquieda, se inicia la macro-acción siguiendo la política ilustrada en la figura y que tiene como objetivo salir de la región.

10.6.2 Options o Macro-Actions

Sutton, Precup, y Singh en [37] extienden el modelo clásico de MDPs e incluyen "options" o macro acciones que son políticas que se siguen durante cierto tiempo. Una macro acción tiene asociada una política, un criterio de terminación y una región en el espacio de estados (la macro-acción puede empezar su ejecución en cualquier estado de la región). Al introducir macro-acciones el modelo se convierte en un proceso de decisión semi-markoviano o SMDP, por sus siglas en inglés.

La figura 10.11 muestra un ejemplo. La región de iniciación de la macro-acción es la parte izquierda de la figura, la política es la mostrada en la figura y la condición de terminación es terminar con probabilidadad de 1 en cualquier estado fuera del cuarto y 0 dentro del cuarto.

Una forma de aprender una política sobre las macro acciones es como sigue:

- En s selecciona macro-acción a y siguela
- Observa el estado resultante s', la recompensa r y el número de pasos N.

- Realiza la siguiente actualización:

$$Q(a,s) := (1-\alpha)Q(s,a) + \alpha[r + \gamma^N max_{a'} Q(s',a')]$$

  donde N es el número de pasos. Si $\gamma = 1$ se convierte en Q-learning tradicional.

La política óptima puede no estar representada por una combinación de macro acciones, por ejemplo, la macro acción *sal-por-la-puerta-más-cercana* es subóptima para los estados que están una posición arriba de la salida inferior. Esto es, si el estado meta está más cercano a la puerta inferior, el agente recorrería estados de más.

10.6.3 HAMs

Parr y Russell proponen lo que llaman máquinas jerárquicas abstractas o HAMs por sus siglas en inglés. Una HAM se compone de un conjunto de máquinas de estados, una función de transición y una función de inicio que determina el estado inicial de la máquina [27]. Los estados en las máquinas son de cuatro tipos: (i) *action*, en el cual se ejecuta una acción, (ii) *call*, llama a otra máquina, (iii) *choice* selecciona en forma no determinista el siguiente estado, y (iv) *stop* regresa el control al estado *call* previo. Con esta técnica se resuelve el problema mostrado en la figura 10.12, en donde se aprenden políticas parciales para: *ir por pasillo* quien llama a *rebotar en pared* y *retroceder*. *Rebotar en pared* llama a *hacia pared* y *seguir pared*, etc.

Una HAM se puede ver como un programa y de hecho Andre y Russell propusieron un lenguaje de programación para HAMs (PHAM) que incorpora al lenguaje LISP [1].

10.6.4 MAXQ

Dietterich [12] propone un esquema jerárquico llamado MAXQ bajo la suposición que para un programador es relativamente fácil identificar submetas y subtareas para lograr esas submetas. MAXQ descompone el MDP original en una jerarquía de MDPs y la función de valor del MDP original se descompone en una combinación aditiva de las funciones de valor de los MDPs más pequeños. Cada subtarea tiene asociado un conjunto de estados, un predicado de terminación y una función de recompensa. La descomposición

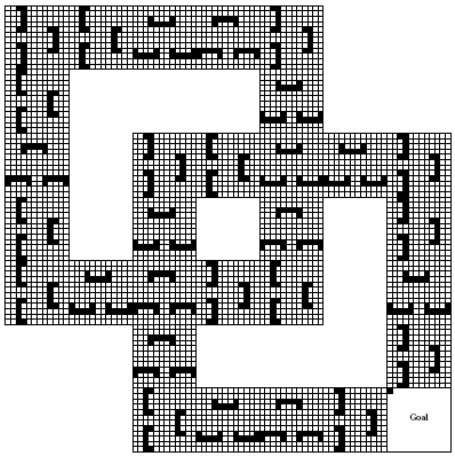

Figura 10.12: Se muestra un dominio que tiene cierta regularidad que es aprovechada por una HAM para solucionarlo eficientemente.

jerárquica define un grafo dirigido y al resolver el nodo raíz se resuelve el problema completo. Se obtiene una política para cada subtarea, la cual consiste en acciones primitivas o en llamar a una subtarea. Esto permite reutilizar las políticas de subtareas en otros problemas. Se puede probar que MAXQ obtiene la política óptima consistente con la descomposición.

MAXQ también introduce una abstración de estados, en donde para cada subtarea se tiene un subconjunto de variables reduciendo el espacio significativamente. Por ejemplo en el problema del Taxi mostrado en la figura 10.13 el objetivo es recoger a un pasajero en uno de los estados marcados con una letra y dejarlo en algún destino (de nuevo marcado con una letra). La parte derecha de la figura muestra un grafo de dependencias en donde la navegación depende sólo de la posición del taxi y el destino y no si se va a recoger o dejar a un pasajero. Por otro lado el completar la tarea depende del origen y destino del pasajero más que en la localización del taxi. Estas abstracciones simplifican la solución de las subtareas.

Existen diferentes formas de como dividir un problema en sub-problemas, resolver cada sub-problema y combinar sus soluciones para resolver el problema original. Algunos otros enfoques propuestos para MDPs jerárquicos están descritos en [4, 2].

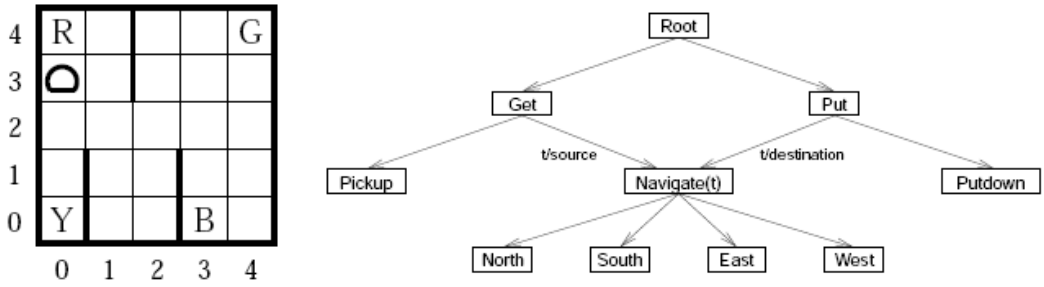

Figura 10.13: La figura de la izquierda muestra el dominio del Taxi donde las letras indican posiciones para recoger o dejar un pasajero. El grafo de la derecha representa las dependencias entre diferentes posibles sub-tareas a realizar.

10.6.5 Decomposición Concurrente

La descomposición concurrente divide el problema entre varios agentes, cada uno ve un aspecto del problema, y entre todos resuelven el problema global operando en forma concurrente. Analizaremos aquí dos enfoques: (i) MDPs paralelos (PMDPs), (ii) MDPs multi–seccionados (MS-MDPs).

MDPs paralelos

Los MDPs paralelos [35] consisten de un conjunto de MDPs, donde cada uno ve un aspecto del mismo problema, obteniendo la política óptima desde cierto punto de vista. Cada MDP tiene, en principio, el mismo espacio de estados y acciones, pero diferente función de recompensa. Esto asume que la parte del problema que ve cada uno es *relativamente* independiente de los otros. Se tiene un árbitro que coordina a los diferentes MDPs, recibiendo sus posibles acciones con su valor Q, y seleccionando aquella acción que de el mayor valor combinado (asumiendo una combinación aditiva), que es la que se ejecuta.

Formalmente un PMDP es un conjunto de K procesos, $P_1, P_2, ..., P_k$, donde cada P_i es un MDP. Todos los MDPs tienen el mismo conjunto de estados, acciones y funciones de transición: $S_1 = S_2 = ... = S_k$; $A_1 = A_2 = ... = A_k$; $\Phi_1(a, s, s') = \Phi_2(a, s, s') = ... = \Phi_k(a, s, s')$. Cada MDP tiene una función diferente de recompensa, $R_1, R_2, ..., R_k$. La recompensa total, RT, es la suma de las recompensas individuales: $RT = R_1 + R_2 + ... + R_k$.

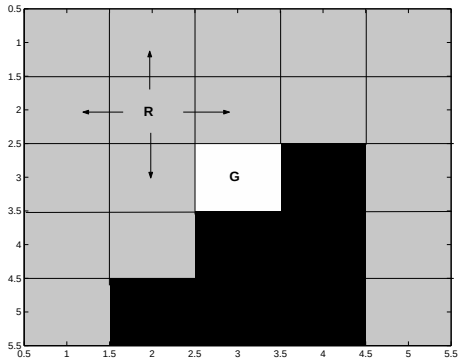

Figura 10.14: Ejemplo de un robot simulado en una rejilla. El ambiente consiste de celdas que pueden ser libres (gris), obstáculos (negro) o la meta (blanco). El robot (R) puede moverse a cada una de las 4 celdas vecinas como se muestra en la figura.

Por ejemplo, consideremos un robot simulado en una rejilla (ver figura 10.14). El robot tienen que ir de su posición actual a la meta, y al mismo tiempo evadir obstáculos. Definimos dos MDPs: (i) a *Navegador*, para ir a la meta, y (ii) *Evasor*, para evadir los obstáculos. Ambos tienen el mismo espacio estados (cada celda en la rejilla) y acciones (arriba, abajo, izquierda o derecha); pero una recompensa diferente. El *Navegador* obtiene una recompensa positiva cuando llega a la meta, mientras que el *Evasor* tiene una recompensa negativa cuando choca con un obstáculo.

Si asumimos que las funciones de recompensa y valor son aditivas, entonces la pólitica óptima es aquella que maximize la suma de los valores Q; esto es: $\pi_I^*(s) = argmax_a \sum_{i \in K} Q_i^*(s,a)$, donde $Q_i^*(s,a)$ es el Q óptimo que se obtiene al resolver cada MDP individualmente. En base a esto podemos definir el algoritmo 13 para resolver un PMDP.

Algoritmo 13 Solución de un MDP paralelo.

Resolver cada MDP mediante cualquier método de solución básico.
Obtener el valor Q_i óptimo por estado–acción:

$$Q_i^*(s,a) = \{r_i(s,a) + \gamma \Sigma_{s' \in S} \Phi(a,s,s') V_i^*(s')\}$$

Calcular el valor óptimo global de Q: $Q^*(s,a) = \Sigma_{i=1}^K Q_i^*(s,a)$
Obtener la política óptima: $\pi^*(s) = argmax_a \{\Sigma_{i=1}^K Q_i^*(s,a)\}$

La figura 10.15 muestra otro ejemplo de un PMDP en el mundo de rejillas,

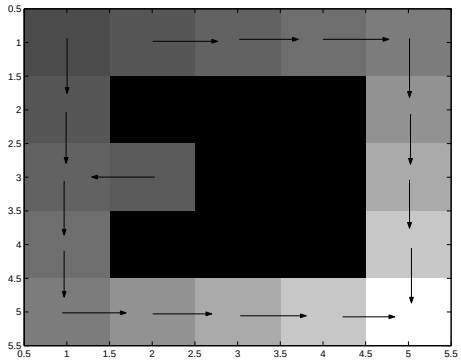

Figura 10.15: Función de valor y política para una rejilla con la meta en $5, 5$. Para cada celda libre se muestra el valor (como un tono de gris, más claro indica mayor valor) y la acción óptima (flecha) obtenidas al resolver el PMDP.

con dos agentes, un *Navegador* y un *Evasor*. En este caso la meta está en la celda $5, 5$ (abajo a la izquierda), y observamos que el PMDP obtiene la solución correcta, indicada por la flecha en cada celda.

Los PMDPs no reducen directamente el espacio de estados y acciones, pero facilitan la solución de problemas complejos, al poder separar diferentes aspectos del mismo; ya sea si se construye el modelo por un experto y luego se resuleve, o mediante aprendizaje por refuerzo. También ayudan a la abstracción de estados, lo que pudiera llevar a una reducción en el tamaño del problema.

MDPs multi-seccionados

Otro enfoque relacionado, llamado MDPs *multi-seccionados* [14], si considera una reducción en el número de estados y acciones. Se asume que se tiene una tarea que se puede dividir en varias sub–tareas, de forma que cada una implica diferentes acciones, y, en general, no hay conflicto entre ellas. Por ejemplo, en un robot de servicio que interactua con personas, se podría dividir el problema en: (i) navegador, permite al robot desplazarse y localizarse en el ambiente, (ii) interacción por voz, realiza la interacción mediante voz con las personas, y (iii) interacción por ademanes, permite la interacción gestual entre el robot y los usuarios. Cada uno de estos aspectos considera una parte del espacio de estados, y acciones diferentes; aunque todos se requiren para

realizar la tarea, operan en cierta forma *independiente*.

Un MS-MDP es un conjunto de N MDPs, que comparten el mismo espacio de estados y recompensas (objetivo), pero tienen diferentes conjuntos de acciones. Se asume que las acciones de cada MDP no tienen *conflictos* con los otros procesos, de forma que cada uno de los MDPs puede ejecutar sus acciones (basadas en la política óptima) en forma concurrente con los demás. Considerando una representación factorizada del espacio de estados, cada MDP sólo necesita incluir las variables de estado que son relevantes para sus acciones y recompensa, lo que puede resultar en una reducción considerable del espacio de estados–acciones. Esto implica que cada MDP, P_i, sólo tendrá un subconjunto de las variables de estado, que conforman su espacio de estados local, S_i. No se consideran explícitamente los efectos de acciones combinadas.

La solución de un MS-MDP incluye dos fases. En la fase de diseño, fuera de línea, cada MDP se define y se resuelve independientemente, obteniendo la política óptima para cada sub–tarea. En la fase de ejecución, en línea, las políticas de todos los MDPs se ejecutan concurrentemente, seleccionando la acción de acuerdo al estado actual y a la política de cada uno. No hay una coordinación explícta entre los MDPs, ésta se realiza implícitamente mediante el vector de estados común (puede haber algunas variables de estado que estén en el espacio de dos o más de los MDPs).

Como mencionamos anteriormente, se asume que no hay conflictos entre acciones; donde un conflicto es alguna restricción que no permite la ejecución de dos acciones en forma simultánea. Por ejemplo, para el caso del robot de servicio, si el robot tiene una sola cámara que utiliza para navegar y reconocer personas, podría haber conflicto si al mismo tiempo se quieren hacer las dos cosas. Actualmente el MS-MDP da prioridades a las sub–tareas, de forma que en caso de conflicto se ejecutá solamente la acción del MDP de mayor prioridad. En general, el problema de que hacer en caso de conflictos sigue abierto, si se considera encontrar la solución óptima desde un punto de vista global. En la sección 10.7 se ilustra la aplicación de MS-MDPs para coordinar las acciones de un robot mensajero.

10.7 Aplicaciones

Para ilustrar la aplicación práctica de MDPs y aprendizaje por refuerzo, en esta sección describimos tres aplicaciones, que ejemplifican diferentes técnicas

que hemos analizado en este capítulo:

- Aprendiendo a volar. Mediante una representación relacional y aprendizaje por refuerzo se logra controlar un avión.
- Control de una planta eléctrica. Se utiliza una representación cualitativa de un MDP para controlar un proceso complejo dentro de una planta eléctrica.
- Homer: el robot mensajero. A través de MDPs multi-seccionados se coordina un robot de servicio para que lleve mensajes entre personas.

10.7.1 Aprendiendo a Volar

Supongamos que queremos aprender a controlar un avión de un simulador de alta fidelidad. Esta tarea la podemos formular como un problema de aprendizaje por refuerzo, donde queremos aprender cuál es la mejor acción a realizar en cada estado para lograr la máxima recompensa acumulada reflejando un vuelo exitoso. Sin embargo, este dominio tiene típicamente entre 20 y 30 variables, la mayoría continuas que describen el movimiento del avión en un espacio tridimensional potencialmente "infinito".

Para esta aplicación se uso un modelo de alta fidelidad de un avión acrobático Pilatus PC-9. El PC-9 es un avión extremadamente rápido y maniobrable que se usa para el entrenamiento de pilotos. El modelo fué proporcionado por la DSTO (Australian Defense Science and Technology Organization) y construído usando datos de un tunel de viendo y del desempeño del avión en vuelo.

En el avión se pueden controlar los alerones, elevadores, impulso, los *flaps* y el tren de aterrizaje. En este trabajo solo se considera como controlar los alerones y elevadores con lo cual se puede volar el avión en vuelo y que son las dos partes más difíciles de aprender. Se asume en los experimentos que se tiene un impulso constante, los *flaps* planos, el tren de aterrizaje retraído y que el avión está en vuelo. Se añade turbulencia durante el proceso de aprendizaje como un cambio aleatorio en los componentes de velocidad del avión con un desplazamiento máximo de 10 pies/seg. en la posición vertical y 5 pies/seg. en la dirección horizontal.

La salida del simulador incluye información de la posición, velocidad, orientación, los cambios en orientación e inclinación vertical y horizontal, y la posición de objetos, tales como edificios, que aparecen en el campo visual.

Una misión de vuelo se especifica como una secuencia de puntos por lo que el avión debe pasar con una tolerancia de ±50 pies, osea que el avión debe de pasar por una ventana de 100 x 100 pies^2 centrada alrededor del punto.

Este problema es difícil para aprendizaje por refuerzo, inclusive con una discretización burda de los parámetros[1]. Además, si se logra aprender una política, ésta solo sirve para cumplir una misión particular y se tendría que aprender una nueva política por cada misión nueva. Lo que nos gustaría es aprender una sola política que se pueda usar para cualquier misión bajo condiciones de turbulencia. Para esto, usamos una representación relacional para representar la posición relativa del avión con otros objetos y con la meta.

Decidimos dividir el problema en dos: (i) movimientos para controlar la inclinación vertical o elevación y (ii) movimientos para controlar la inclinación horizontal y dirección del avión. Para el control de la elevación, los estados se caracterizaron con predicados que determinan la distancia y elevación a la meta. Para el control de los alerones, además de la distancia a la meta se usaron predicados para determinar la orientación a la meta, la inclinación del avión y la tendencia seguida por el avión en su inclinación. Las acciones se pueden aprender directamente de trazas de pilotos usando lo que se conoce como *behavioural cloning* o clonación.

La clonación aprende reglas de control a partir de trazas de expertos humanos, e.g., [33, 22, 6]. Se sigue un esquema relativamente simple: De la información de una traza de vuelo, por cada estado se evaluan los predicados que aplican y se obtiene la acción realizada. Si la acción de control es una instancia de una acción anterior, no se hace nada, de otro forma se crea una nueva acción con una conjunción de los predicados que se cumplieron en el estado. Este esquema también puede usarse en forma incremental, incorporando nuevas acciones en cualquier momento (ver también [23] para una propuesta similar en ajedrez).

En total se aprendieron 407 acciones, 359 para alerón (de un total de 1,125 posibles) y 48 para elevación (de un total de 75 posibles) después de cinco trazas de vuelo realizadas por un usuario y de 20 trazas seguidas en un esquema de exploración para visitar situaciones no vistas en las trazas originales. Con esto solo se aprenden un subconjunto de las acciones potenciales

[1]Por ejemplo, considerando un espacio bastante pequeño de 10 km^2 con 250m. de altura. Una discretización burda de $500 \times 500 \times 50$ m. con 5 posibles valores para la orientación e inclinaciones verticales y horizontales, y 5 posibles acciones por estado, nos da 1,250,000 pares estado-acción.

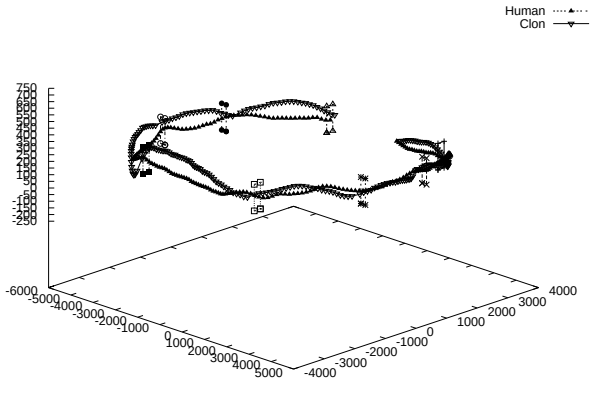

Figura 10.16: Traza de un humano y seguida por la política aprendida con aprendizaje por refuerzo y clonación en la misión de ocho metas y con el máximo nivel de turbulencia.

(una tercera parte) lo cual simplifica el proceso de aprendizaje por refuerzo con un promedio de 1.6 acciones por cada estado de alerón y 3.2 por estado de elevación.

Una vez que se inducen las acciones se usa rQ-learning para aprender una política que permita volar el avión. En todos los experimentos los valores Q se inicializaron a -1, $\epsilon = 0.1, \gamma = 0.9, \alpha = 0.1$, y $\lambda = 0.9$ (dado que usamos trazas de elegibilidad). Los experimentos se hicieron con ocho metas en un recorrido como se muestra en la figura 10.16. Si la distancia del avión se incrementa con respecto a la meta actual durante 20 pasos de tiempo, se asume que se paso de la meta y se le asigna la siguiente meta.

La política aprendida después de 1,500 vuelos es capaz de volar la misión completa en forma exitosa. Se probó su robustez bajo diferentes condiciones de turbulencia. La figura 10.16 muestra una traza de un humano en esta misión y la traza seguida por la política aprendida. La política aprendida se probó en otra misión de cuatro metas. La intención era probar maniobras no vistas antes. La nueva misión incluyó: una vuelta a la derecha[2], una vuelta cerrada a la izquierda subiendo no vista antes, otra vuelta rápida a la derecha y una vuelta cerrada de bajada a la derecha.

La figura 10.17 muestra una traza de un humano y de la política aprendida, en una nueva misión con un valor máximo de turbulencia. Claramente, la política aprendida puede volar en forma razonable una misión completamente nueva.

[2]La misión de entrenamiento tiene solo vueltas a la izquierda.

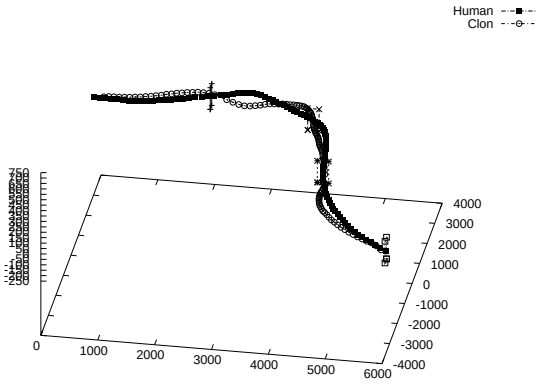

Figura 10.17: Traza de vuelo de un humano y de la política aprendida de la primera misión en una nueva misión de cuatro metas.

10.7.2 Control de una Planta Eléctrica

Una planta de generación de energía eléctrica de ciclo combinado calienta vapor de agua que alimenta una turbina de gas y luego recupera parte de esa energía para alimentar a una turbina de vapor y generar más electricidad. Un generador de vapor, usando un sistema de recuperación de calor, es una proceso capaz de recuperar la energía sobrante de los gases de una turbina de gas, y usarlos para generar vapor. Sus componentes principales son: el domo, la pared de agua, la bomba de recirculación y los quemadores. Por otro lado, los elementos de control asociados a la operación son: la válvula de agua de alimentación, la válvula de combustible de los quemadores, la válvula de vapor principal, la válvula de paso y la válvula de gas. Un diagrama simplificado de una planta eléctrica se muestra en la figura 10.18.

Durante la operación normal de la planta, el sistema de control regula el nivel de vapor del domo. Sin embargo, cuando existe un rechazo de carga parcial o total, los sistemas de control tradicionales no son capaces de estabilizar el nivel del domo. En este caso, se crea un incremento en el nivel de agua debido al fuerte desplazamiento de agua en el domo. El sistema de control reacciona cerrando la válvula de alimentación de agua. Algo parecido sucede cuando existe una alta demanda repentina de vapor. Bajo estas circustancias, la intervención de un operador humano es necesaria para ayudar al sistema de control a tomar las acciones adecuadas para salir del transitorio. Una solución práctica es usar recomendaciones de un asistente de operador inteligente que le diga las mejores acciones a tomar para corregir el problema. Para esto se modelo el problema como un MDP [31].

El conjunto de estados en el MDP se obtiene directamente de las variables

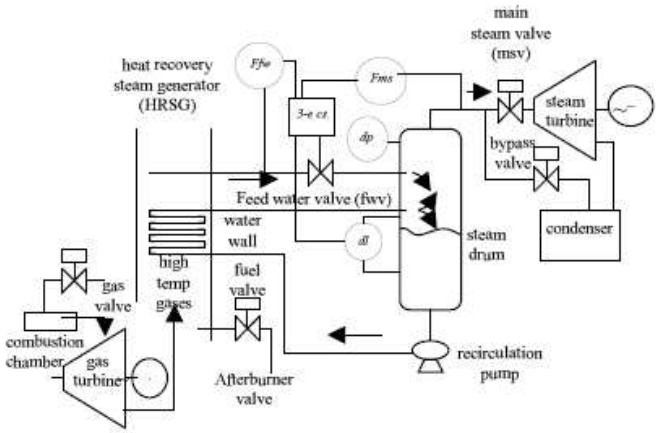

Figura 10.18: Diagrama de una planta de energía eléctrica de ciclo combinado.

de operación del generador de vapor: el flujo de agua de alimentación (Ffw), el flujo principal de vapor (Fms), la presión del domo (Pd) y la generación de potencia (g), como las variables a controlar y un disturbio (d) como evento exógeno. Inicialmente se considera el "rechazo de carga" como el disturbio. Los valores de estas variables representan el espacio de los posibles estados del sistema.

Para la operación óptima de la planta, existen ciertas relaciones entre las variables que se deben de cumplir, especificadas por la curva de operación recomendada. Por ejemplo, la curva deseada para la presión del domo y el flujo principal de vapor se muestra en la figura 10.19.

La función de recompensa para el MDP se basa en la curva de operación óptima. Los estados en la curva reciben refuerzo positivo y el resto de los estados un refuerzo negativo. El objetivo entonces del MDP es encontrar la política óptima para llevar a la planta a un estado dentro de la curva de operación deseada.

El conjunto de acciones se basa en abrir y cerrar las válvulas de agua de alimentación (fwv) y de vapor principal (msv). Se asume que estas válvulas regulan el flujo de agua de alimentación y el flujo principal de vapor.

Para resolver este problema se utilizó la representación cualitativa de MDPs [32] descrita en la sección 10.5.3. Para esto se aprendió un MDP cualitativo a partir de datos generados en el simulador. Primeramente, se generaron datos simulados con información de las variables de estado y re-

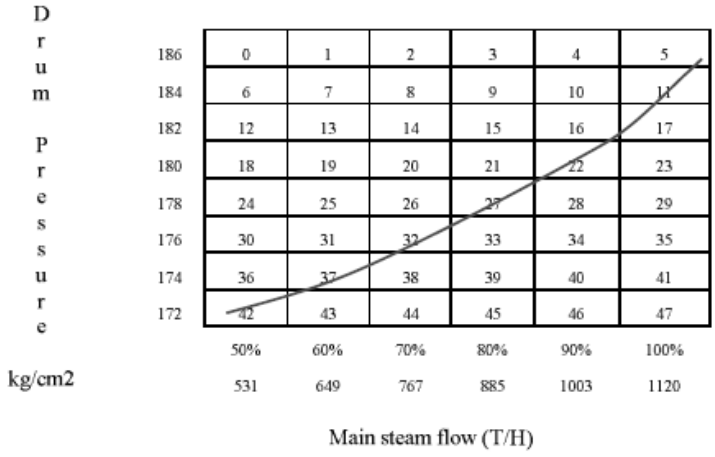

Figura 10.19: Curva de operación deseada para una planta de ciclo combinado. La curva establece las relaciones *óptimas* de operación entre la presión del domo y el flujo de vapor.

compensas de acuerdo a las curvas de operación óptima de la planta. A partir de estos datos se aprendió un árbol de decisión para predecir el valor de recompensa en términos de las variables de estado usando la implementación de C4.5 [30] de Weka [40]. El árbol de decisión se transformó en árbol cualitativo en donde cada hoja del árbol es asociada a un valor cualitativo. Cada rama del árbol representa un estado cualitativo. Los datos iniciales se expresan en términos de los rangos encontrados en el árbol para estas variables, y el resto de las variables que no están incluidas en el árbol se discretizan. Los datos se ordenan para representar transiciones de valores entre un tiempo t y un siguiente tiempo $t+1$. A partir de estos datos secuenciales se aprende una red bayesiana dinámica de dos etapas por cada acción del problema, en esta caso abrir y cerrar válvulas, usando el algoritmo de aprendizaje bayesiano K2 [9] implentado en Elvira [17].

La figura 10.20 muestra la red bayesiana dinámica que representa el modelo de transición para las acciones de abrir y cerrar la válvula de agua de alimentación. Las líneas sólidas representan relaciones entre nodos relevantes y las líneas punteadas representan aquellos nodos que no tienen efecto durante la misma acción.

Al transformar el espacio continuo inicial a una representación de alto nivel, el problema puede ser resuelto usando técnicas estandar de solución

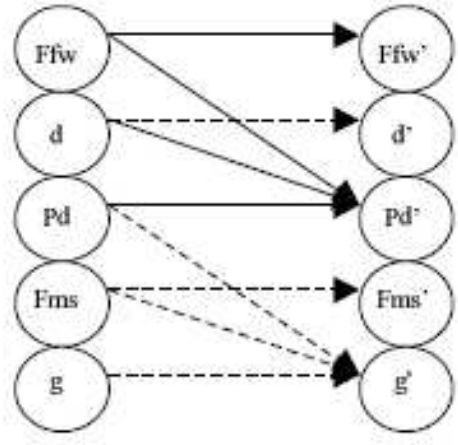

Figura 10.20: Red bayesiana dinámica que representa la función de transición para las acciones abrir o cerrar la válvula de de agua de alimentación.

de MDPs, tales como iteración de valor, para obtener la política óptima. La figura 10.21 muestra una partición de estados automática y la política encontrada para una versión simplificada del problema con 5 acciones y 5 variables. Esta política fué validada por un experto humano, encontrandose satisfactoria. Se están realizando pruebas con un simulador para validarla experimentalmente.

10.7.3 Homer: El robot mensajero

Homer [15], ver figura 10.22, es un robot móvil basado en la plataforma *Real World Interface B-14* que tiene un sólo sensor: una cámara estéreo (Point Grey Research Bumblebee^{TM}).

El objetivo de Homer es llevar mensajes entre personas en un ambiente de interiores, por ejemplo entre oficinas en una empresa. Para ello cuenta con un mapa previo del ambiente, mediante el cual puede localizarse y planear su ruta; además de que sabe en que lugar se encuentra, normalmente, cada persona. Tiene la capacidad de interactuar con las personas mediante síntesis y reconocimiento de voz, interpretando comandos sencillos y guardando los mensajes que se desean enviar. Cuenta con una *cara* simulada, que le permite expresar diferentes "emociones" (alegría, tristeza, enojo y neutro), ver figura 10.22(b), para comunicarse con las personas.

El software de Homer está basado en una arquitectura de capas mediante

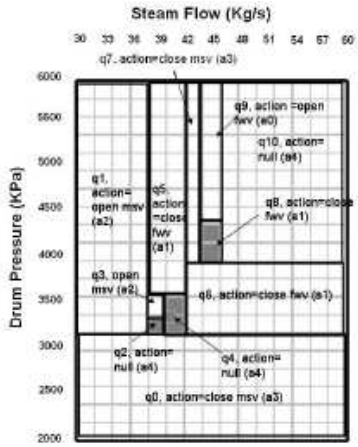

Figura 10.21: Partición cualitativa del espacio de estados, proyectándolo a las dos variables que tienen que ver con la recompensa. Para cada estado cualitativo, q_1 a q_10 se indica la acción óptima obtenida al resolver el MDP cualitativo.

(a) (b)

Figura 10.22: (a) Homer, el robot mensajero interactuando con una persona. (b) Un acercamiento de la *cabeza* de Homer.

comportamientos, de forma que cada una de las capacidades de Homer está contenida en un módulo. Los módulos o comportamientos de Homer incluyen: navegación, [25], localización, [25, 34], detección de caras, [16], generación de expresiones faciales, planeación, [13], reconocimiento de voz, [16], detección de voz, y monitor de recursos. Mediante la combinación de estos comportamientos Homer realiza la tarea de entrega de mensajes.

Para coordinar los módulos se plantea el uso de un MDP, teniendo como objetivo el recibir y entregar mensajes, de forma que Homer reciba recompensas positivas. De esta forma al resolver el MDP, se obtienen las acciones que debe realizar Homer en cada momento (en función del estado) para llegar al objetivo. Pero al considerar todos los posibles estados y acciones en este problema, el espacio crece a aproximadamente $10,000,000$ estados–acciones, por lo que se vuelve imposible resolverlo como un sólo MDP. Para esto, se utiliza el enfoque de MDPs multi–seccionados [14], dividiendo la tarea en un conjunto de sub–tareas que puedan ser resueltas en forma independiente, y ejecutadas concurrentemente. Utilizando MS-MDPs, el MDP más complejo tiene $30,000$ estados–acciones.

La tarea de entrega de mensajes se puede dividir en 3 sub–tareas, cada una controlada por un MDP. El **N**avegador controla la navegación y localización del robot. El manejador de **D**iálogos controla la interacción con las personas mediante voz. El **G**enerador de gestos controla las expresiones realizadas mediante la cara animada para la interacción con las personas. Cada MDP considera sólo las variables relevantes para la sub–tarea dentro del espacio de estados de Homer, y controla varios de los módulos mediante sus acciones. El estado completo consiste de 13 variables que se muestran en la tabla 10.1. Para cada variable se indica cual(es) de los MDPs la consideran. En la tabla 10.2 se listan las accions de cada MDP, y los módulos que las implementan.

Los 3 MDPs se resolvieron utilizando el sistema SPUDD (ver sección 10.5), generandose la política óptima para cada uno. Para coordinar la tarea se ejecutan concurrentemente las 3 políticas mediante un manejador, que simplemente lee las variables de estado, y selecciona la acción óptima para cada sub–tarea de acuerdo a la política. Durante la ejecución de las acciones se evitan conflictos potenciales, simplemente dando prioridad al navegador.

Homer fue evaluado experimentalmente realizando tareas de recepción y entrega de mensajes entre varias personas en el Laboratorio de Inteligencia Computacional en la Universidad de la Columbia Británica en Canadá [14]. Homer logró realizar exitosamente la tarea en varias ocaciones. La tabla 10.3

Variable	MDP
Tiene mensaje	N,D,G
Nombre receptor	N,D,G
Nombre del que envía	N,D,G
En la meta	N,D,G
Tiene meta	N
Meta inalcanzable	N
Receptor inalcanzable	N
Batería baja	N
Localización incierta	N
Voz escuchada (speech)	D,G
Persona cerca	D,G
Llamaron a Homer	D,G
Yes/No	D,G

Tabla 10.1: Variables de estado para Homer. Para cada una se indican los MDPs que la incluye.

MDP	acciones	módulos
Navegador	explorar	navegación
	navegar	navegación
	localizarse	localización
	obten nueva meta	location generator
	va a casa	navegación
	espera	navegación
Diálogo	pregunta	síntesis de voz
	confirma	síntesis de voz
	entrega mensaje	síntesis de voz
Gestos	neutral	generación de gestos
	feliz	generación de gestos
	triste	generación de gestos
	enojado	generación de gestos

Tabla 10.2: Los MDPs que coordinan a Homer, y sus acciones. Para cada acción se indica el módulo que la implementa.

ilustra varias etapas durante la ejecución de la tarea de recepción y entrega de un mensaje.

Tabla 10.3: Ejemplo de una corrida para entregar un mensaje. Para cada estapa se muestra: (i) una imagen representativa, (ii) variables significativas, (iii) acciones que toma cada MDP (**N**avegador, **D**iálogo, **G**estos), y una breve descripción.

imagen	variables cambiadas	acciones	descripción
	Persona cerca = true Voz escuchada = true	**N**: - **D**: pregunta **G**: feliz	Se acerca persona. HOMER inicia conversación preguntandole su nombre y sonriendo.
	tiene_mensaje = true Tiene meta = true Voz escuchada = false Nombre envía = 4 Nombre Receptor = 1	**N**: navega **D**: - **G**: neutral	HOMER recive mensaje y destinatario y obtiene el destino de su módulo de planificación. El navegador empieza a moverlo hacia el destino.
	Persona cerca=false Loc. incierta = true	**N**: localizarse **D**: - **G**: -	Durante la navegación, la posición de HOMER se volvió incierta. El MDP se navegación se localiza. Los estados de los otros dos MDPs no se ven afectados.
	en_destino=true Voz escuchada=true yes/no=yes	**N**: espera **D**: entrega mensaje **G**: feliz	HOMER llego a su destino, detectó una persona y confirmó que era el receptor (a través de *yes/no*). Entrega mensaje y sonrie.
	tiene_mensaje = false En la meta=false Batería baja=true	**N**: ve a casa **D**: - **G**: -	Las baterías de HOMER se detectan bajas, el MDP navegador regresa a HOMER a casa.

10.8 Conclusiones

Un MDP modela un problema de decisión sequencial en donde el sistema evoluciona en el tiempo y es controlado por un agente. La dinámica del sistema está determinada por una función de transición de probabilidad que mapea estados y acciones a otros estados. El problema fundamental en MDPs es encontrar una política óptima; es decir, aquella que maximiza la recompensa que espera recibir el agente a largo plazo. La popularidad de los MDPs radica en su capacidad de poder decidir cual es la mejor acción a realizar en cada estado para lograr cierta meta, en condiciones inciertas.

En este capítulo hemos revisado los conceptos básicos de los procesos de decisión de Markov y descrito las principales técnicas de solución. Existen diferentes técnicas para solucionar MDPs dependiendo de si se conoce un modelo del ambiente o no se conoce. Independientemente de esto, la obtención de una solución en tiempo razonable se ve fuertemente afectada por el tamaño del espacio de estados y acciones posibles. Para esto se han planteado propuestas como factorizaciones, abstracciones y descomposiciones del problema, que buscan escalar las técnicas de solución a problemas más complejos. Estos avances han servido para controlar elevadores, jugar *backgammon*, controlar aviones y helicópteros, controlar plantas de generación de electricidad, servir como planificadores para robots, seleccionar portafolios financieros, y control de inventarios, entre otros.

El área de MDPs ha mostrado un fuerte desarrollo en los últimos años y se espera que se siga trabajando en extensiones; en particular en cómo combinar varias soluciones de MDPs diferentes un una solución global, en como reutilizar las soluciones obtenidas otros problemas similares, y en aplicaciones a dominios más complejos. Adicionalmente, se están desarrollando técnicas aproximadas para resolver POMDPs, en los cuales existe incertidumbre respecto al estado, por lo que se vuelven mucho más complejos.

Referencias

[1] D. Andre and S. Russell. Programmable reinforcement learning agents. In *Advances in Neural Information Processing Systems*, volume 13, 2000.

[2] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.

[3] R.E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, 1957.

[4] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artifical Intelligence Research*, 11:1–94, 1999.

[5] Craig Boutilier, Richard Dearden, and Moise's Goldszmidt. Exploiting structure in policy construction. In Chris Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111, San Francisco, 1995. Morgan Kaufmann.

[6] I. Bratko, T. Urbančič, and C. Sammut. Behavioural cloning: phenomena, results and problems. automated systems based on human skill. In *IFAC Symposium*, Berlin, 1998.

[7] D. Chapman and L. Kaelbling. Input generalization in delayed reinforcement learning: an algorithm and performance comparison. In *Proc. of the International Joint Conference on Artificial Intelligence*, pages 726–731, San Francisco, 1991. Morgan Kaufmann.

[8] N. Charness. Human chess skill. In P.W. Frey, editor, *Chess skill in man and machine*, pages 35–53. Springer-Verlag, 1977.

[9] G.F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–348, 1992.

[10] A. de Groot. *Thought and Choice in Chess*. Mouton, The Hague, 1965.

[11] T. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[12] T.G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[13] P. Elinas, J. Hoey, D. Lahey, J. Montgomery, D. Murray, S. Se, and J.J. Little. Waiting with Jose, a vision based mobile robot. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA '02)*, Washington, D.C., May 2002.

[14] P. Elinas, L.E. Sucar, J. Hoey, and A. Reyes. A decision-theoretic approach for task coordination in mobile robots. In *Proc. IEEE Robot Human Interaction (RoMan)*, Japan, September 2004.

[15] Pantelis Elinas, Jesse Hoey, and James J. Little. Human oriented messenger robot. In *Proc. of AAAI Spring Symposium on Human Interaction with Autonomous Systems*, Stanford, CA, March 2003.

[16] Pantelis Elinas and James J. Little. A robot control architecture for guiding a vision-based mobile robot. In *Proc. of AAAI Spring Symposium in Intelligent Distributed and Embedded Systems*, Stanford, CA, March 2002.

[17] The Elvira Consortium. Elvira: An environment for creating and using probabilistic graphical models. In *Proceedings of the First European Workshop on Probabilistic graphical models (PGM'02)"*, pages 1–11, Cuenca, Spain, 2002.

[18] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of International Conference on Uncertainty in Artificial Intelligence (UAI '99)*, Stockholm, 1999.

[19] R. A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, Mass., U.S.A., 1960.

[20] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2), 1998.

[21] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2), 1998.

[22] D. Michie, M. Bain, and J. Hayes-Michie. Cognitive models from subcognitive skills. In M. Grimble, J. McGhee, and P. Mowforth, editors, *Knowledge-Based Systems in Industrial Control*, pages 71–90. Peter Peregrinus, 1990.

[23] E. Morales. On learning how to play. In H.J. van den Herik and J.W.H.M. Uiterwijk, editors, *Advances in Computer Chess 8*, pages 235–250. Universiteit Maastricht, The Netherlands, 1997.

[24] E. Morales. Scaling up reinforcement learning with a relational representation. In *Proc. of the Workshop on Adaptability in Multi-agent Systems (AORC-20 03)*, pages 15–26, 2003.

[25] D. Murray and J. Little. Using real-time stereo vision for mobile robot navigation. *Autonomous Robots*, 8:161–171, 2000.

[26] A.E. Nicholson and J.M. Brady. Dynamic belief networks for discrete monitoring. *IEEE Trans. on Systems, Man, and Cybernetics*, 24(11):1593–1610, 1994.

[27] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1997.

[28] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann, San Francisco, CA., 1988.

[29] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, New York, NY., 1994.

[30] J. R. Quinlan. *C 4.5: Programs for machine learning*. The Morgan Kaufmann Series in Machine Learning, San Mateo, CA: Morgan Kaufmann, —c1993, 1993.

[31] A. Reyes, P. H. Ibarguengoytia, and L. E. Sucar. Power plant operator assistant: An industrial application of factored MDPs. *MICAI 2004: Advances in Artificial Intelligence*, pages 565–573, 2004.

[32] A. Reyes, L. E. Sucar, E. Morales, and Pablo H. Ibarguengoytia. Abstraction and refinement for solving Markov Decision Processes. In *Workshop on Probabilistic Graphical Models PGM-2006*, pages 263–270, Chezch Republic, 2006.

[33] C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *Proc. of the Ninth International Conference on Machine Learning*, pages 385–393. Morgan Kaufmann, 1992.

[34] S. Se, D. Lowe, and J.J. Little. Mobile robot localization and mapping with uncertainty using scale-invariant landmarks. *International Journal of Robotics Research*, 21(8):735–758, August 2002.

[35] L.E. Sucar. Parallel markov decision processes. In *2nd European Workshop on Probabilistic Graphical Models*, pages 201–208, Holland, 2004.

[36] R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[37] R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[38] G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning hierarchical partially observable Markov decision process models for robot navigation. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA '01)*, Seoul, Korea, May 2001.

[39] S. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109, 2000.

[40] Ian H. Witten and Eibe Frank. *Data mining: practical machine learning tools and techniques with Java implementations.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.