**Proyecto:** **Implementación de un modelo didáctico de Procesos de Decisión de Markov Parcialmente Observables (POMDP)**

Autor: Leonardo Romero Muñoz
Fecha: Mayo de 1999

## 1. Introducción

Los Procesos de Decisión de de Markov Parcialmente Observables (POMDP, por sus siglas en inglés) se presentan como una alternativa para el modelado de problemas de decisión secuencial. Considérese un agente (v.g. un robot) que puede situarse en un número finito de estados. De acuerdo a las acciones que realiza (v.g. avanzar un metro), el agente puede cambiar de estado. En un POMDP estas acciones se modelan en forma no determinista asignando **probabilidades a las transiciones**. Para conocer su entorno, el agente puede realizar observaciones de objetos (v.g. usando sensores) para estimar su estado actual. En un POMDP estas observaciones se modelan también en forma no determinista asignando **probabilidades de observación** de los objetos dado el estado en que se encuentre el agente. El modelo incluye una recompensa (número real) asociada a la ejecución de una acción en cierto estado.

Al inicio, se puede asignar una distribución de probabilidad del estado inicial; permitiendo especificar tanto certeza del estado inicial, asignando un valor de probabilidad unitaria al estado inicial y cero a los demás, como incertidumbre total, asignando un mismo valor a cada estado. Algunos de los problemas que se pueden plantear en un POMDP son los siguientes:

problema 1. Dada un modelo POMDP, y una acción o una observación, actualizar la distribución del estado actual.

problema 2. Dada un estado meta, en forma implícita por la función de recompensa. Determinar las acciones a ejecutar para cumplir la meta.

problema 3. Dada una secuencia de observaciones-acciones, determinar la probabilidad de que el modelo genere dicha secuencia.

problema 4. Aprender un POMDP, dada una estructura del modelo y una secuencia de observaciones-acciones.

En este proyecto se implementó una versión didáctica de POMDP para resolver los problemas mencionados.

## 2. Teoría.

Un POMDP es similar a un Modelo de Markov Oculto (HMM) y consiste de:

- Un conjunto finito de estados S.
- Un conjunto finito de observaciones O.
- Una distribución del estado inicial Pi (una distribución de probabilidad sobre S).
- Cada estado *s* de S tiene un conjunto finito de acciones A(*s*) que pueden ejecutarse en *s*.
- Una función de transición *p* (de S x A -> S), donde p(*s'* |*s,a*) denota la probabilidad de que el proceso realice una transición del estado *s* al estado *s'* cuando la acción *a* se ejecuta.
- Una función de observación q (una función de S a distribuciones de probabilidad sobre O), donde q(*o*|*s*) denota la probabilidad de observar *o* en el estado *s*.
- Una función de recompensa inmediata *r* (una función de S x A a los números reales) donde r(*s,a*) denota la recompensa inmediata resultado de la ejecución de la acción *a* en el estado *s*.

Enseguida se abordan las soluciones a cada uno de los problemas planteados en la sección 1 según la referencia [Koenig y Simmons, 1998].

Para los problemas 3, 4 se sigue algoritmos similares a los empleados en el caso de HMM, teniendo en cuenta la acción realizada en cada paso puesto que de ella dependen las probabilidades de transición de estados. En el caso de HMM no se contemplan acciones. Enseguida se aborda la solución a los problemas 1 y 2.

**2.1. problema 1: Estimar el estado actual**

En este modelo el agente no puede conocer directamente su estado actual, sino que para ello puede realizar observaciones. Sea E(*s*) la probabilidad de que el agente se encuentre en el estado *s*. Al inicio E(s) = Pi(s). Después, si la distribución $E_{prior}$ es la distribución actual, la distribución después de realizar la acción *a* esta dada por:

$$E_{post}(s) = \frac{1}{escala} \sum_{s' \in S | a \in A(s')} p(s|s', a) E_{prior}(s')$$

Donde *escala* es una constante de normalización que garantiza que $\sum_{s \in S} E_{post}(s) = 1$. La normalización es necesaria solamente si no se tienen las mismas acciones posibles para cada estado. Por otro lado, la actualización de la distribución después de realizar una observación es:

$$E_{post}(s) = \frac{1}{escala} q(o|s) E_{prior}(s)$$

Donde *escala*, de nuevo, es una constante de normalización que garantiza que $\sum_{s \in S} E_{post}(s) = 1$.

**2.2. Problema 2: Determinar una política óptima para alcanzar la meta.**

El problema de calcular una política óptima para alcanzar una meta se puede simplificar si el agente asume que se encuentra en el estado actual más probable [Nourbakhsh, Powers y Birchfield, 1995]. Con esta suposición, el modelo se puede transformar en un Proceso de Decisión de Markov (MDP, por sus siglas en inglés) completamente observable. Una política óptima puede obtenerse a través de un método de programación dinámica llamada “iteración de valor” [Bellman, 1957]:

Paso 1. $V_1(s) := 0$ para $s \in S$. t := 1.
Paso 2. $v_{t+1}(s) := \max_{a \in A(s)} [r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) v_t(s')]$ para todo $s \in S$. t := t+1
Paso 3. Ir al paso 2 hasta que los valores de v no cambien.

Donde $\gamma$ es un “factor de descuento” y si se asume un valor menor de la unidad, se garantiza que la recompensa total esperada es finita. Después de calculada la función v(s), la acción óptima para ejecutar, dado que el agente se encuentra en el estado s, esta dada por:

$$a(s) := \operatorname{argmax}_{a \in A(s)} [r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) v_t(s')]$$

**3. Metodología**

Se desarrolló un programa que implementa un POMDP usando el lenguaje de programación C. Para demostrar la ejecución del programa se diseñó un caso de prueba de un robot móvil simple que se desplaza en una dimensión.

Supóngase que el robot en el escenario planteado en la figura 1. El robot está marcado con la letra “R” y puede ejecutar tres posibles acciones: avanzar un metro hacia la izquierda, avanzar un metro hacia la derecha o detenerse. El robot está equipado con un sensor de proximidad infrarojo que le indica si existe pared cercana a él. El robot también dispone de un odómetro para estimar cuando ha recorrido un metro de distancia en alguna dirección y por lo tanto parar la actividad de los motores de sus ruedas. Inicialmente se asume que se tiene incertidumbre de la posición inicial del robot.Se modela la incertidumbre asociada a las acciones permitiendo que con probabilidad 0.9 se realize la acción deseada y con probabilidad 0.1 el robot conserve su estado. De manera similar, se asume que el sensor con probabilidad 0.9 da la observación correcta y con probabilidad 0.1 se equivoca. Como función de recompensa se penaliza las acciones *izq* y *der* con un valor de -0.1, la acción de paro se penaliza para todos los estado que no sean la meta (en este caso, el estado correspondiente a p5). Por simplicidad se asume que el robot puede ejecutar las tres acciones en cualquier estado. El modelo contiene 8 estados (p0 a p8) cuyas posiciones están representadas en la figura 1.

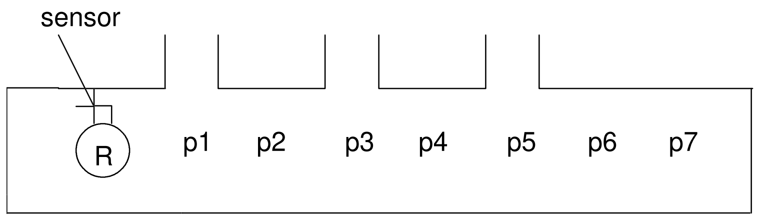


Figura 1. Ejemplo de un robot móvil en una dimensión

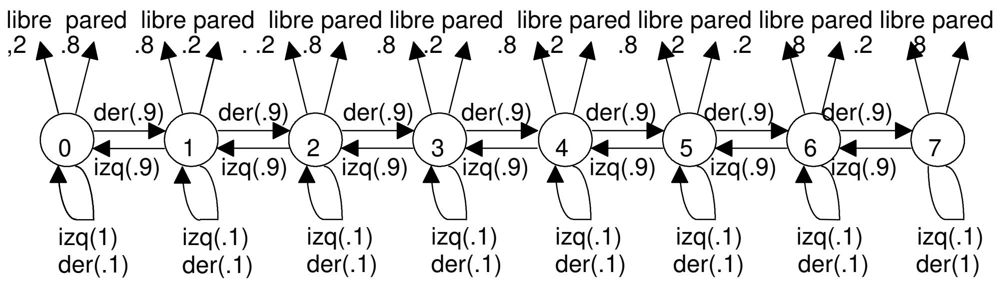


Figura 2. POMDP del ejemplo del robot móvil

El modelo POMDP asociado al robot móvil se muestra gráficamente en la figura 2 y consiste de los siguientes elementos:

1. S = {0,1,2,3,4,5,6,7}

2. O = {libre, pared}

3. Pi = (0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125)

4. A = {izq, der, paro}

5. p(0|0,izq) = 1, p(i|0, izq) = 0 si i > 0 (el robot rebota)
p(0|1,izq) = 0.9, p(1|1,izq) = 0.1, p(i|1,izq) = 0 si i>1
p(0|2,izq) = 0, p(1|2,izq) = 0.9, p(2|2,izq) = 0.1, p(i|2,izq) = 0 si i>2
p(i|3,izq) = 0 si i < 2, p(2|3,izq) = 0.9, p(3|3,izq) = 0.1, p(i|3,izq) = 0 si i>3
p(i|4,izq) = 0 si i < 3, p(3|4,izq) = 0.9, p(4|4,izq) = 0.1, p(i|4,izq) = 0 si i>4
p(i|5,izq) = 0 si i < 4, p(4|5,izq) = 0.9, p(5|5,izq) = 0.1, p(i|5,izq) = 0 si i>5
p(i|6,izq) = 0 si i < 5, p(5|6,izq) = 0.9, p(6|6,izq) = 0.1, p(i|6,izq) = 0 si i>6
p(i|7,izq) = 0 si i < 6, p(6|7,izq) = 0.9, p(7|7,izq) = 0.1
p(0|0,der) = 0.1, p(1|0, der) = 0.9, p(i|0,der) = 0 si i > 1
p(0|1,der) = 0, p(1|1,der) = 0.1, p(2|1,der) = 0.9, p(i|1,der) = 0 si i>2
p(i|2,der) = 0 si i < 2, p(2|2,der) = 0.1, p(3|2,der) = 0.9, p(i|2,der) = 0 si i>3
p(i|3,der) = 0 si i < 3, p(3|3,der) = 0.1, p(3|3,der) = 0.9, p(i|3,der) = 0 si i>3
p(i|4,der) = 0 si i < 4, p(4|4,der) = 0.1, p(4|4,der) = 0.9, p(i|4,der) = 0 si i>4
p(i|5,der) = 0 si i < 5, p(5|5,der) = 0.1, p(5|5,der) = 0.9, p(i|5,der) = 0 si i>5
p(i|6,der) = 0 si i < 6, p(6|6,der) = 0.1, p(6|6,der) = 0.9, p(i|6,der) = 0 si i>6
p(i|7,der) = 0 si i < 7, p(7|7,der) = 1 (el robot rebota)
p(i|i,paro) = 1 para toda i, p(i|j,paro) = 0 para i diferente de j

6. q(libre|0) = .2, q(pared|0) = .8

| | | | |
|---|---|---|---|
| | q(libre\|1) = .8, | q(pared\|1) = .2 | |
| | q(libre\|2) = .2, | q(pared\|2) = .8 | |
| | q(libre\|3) = .8, | q(pared\|3) = .2 | |
| | q(libre\|4) = .2, | q(pared\|4) = .8 | |
| | q(libre\|5) = .8, | q(pared\|5) = .2 | |
| | q(libre\|6) = .2, | q(pared\|6) = .8 | |
| | q(libre\|7) = .2, | q(pared\|7) = .8 | |
| 7. | r(0,izq) = -.1 | r(0,der) = - .1 | r(0,paro) = -10 |
| | r(1,izq) = -.1 | r(1,der) = - .1 | r(1,paro) = -10 |
| | r(2,izq) = -.1 | r(2,der) = - .1 | r(2,paro) = -10 |
| | r(3,izq) = -.1 | r(3,der) = - .1 | r(3,paro) = -10 |
| | r(4,izq) = -.1 | r(4,der) = - .1 | r(4,paro) = -10 |
| | r(5,izq) = -.1 | r(5,der) = - .1 | r(5,paro) = 10 |
| | r(6,izq) = -.1 | r(6,der) = - .1 | r(6,paro) = -10 |
| | r(7,izq) = -.1 | r(7,der) = - .1 | r(7,paro) = -10 |

**4. Resultados**

Los parámetros del ejemplo del robot móvil quedan codificados en un archivo llamado robot.pom que contiene lo siguiente:

---

```
POMDP
N
8
Q
0 1 2 3 4 5 6 7
M
2
V
libre pared
X
3
W
izq der paro
Pi
0.125 0.125 0.125 0.125 0.125 0.125 0.125 0.125
A
1  0  0  0  0  0  0  0
.9 .1 0  0  0  0  0  0
0  .9 .1 0  0  0  0  0
0  0  .9 .1 0  0  0  0
0  0  0  .9 .1 0  0  0
0  0  0  0  .9 .1 0  0
0  0  0  0  0  .9 .1 0
0  0  0  0  0  0  .9 .1
.1 .9 0  0  0  0  0  0
0  .1 .9 0  0  0  0  0
0  0  .1 .9 0  0  0  0
0  0  0  .1 .9 0  0  0
0  0  0  0  .1 .9 0  0
0  0  0  0  0  .1 .9 0
0  0  0  0  0  0  .1 .9
0  0  0  0  0  0  0  1
1  0  0  0  0  0  0  0
0  1  0  0  0  0  0  0
0  0  1  0  0  0  0  0
0  0  0  1  0  0  0  0
0  0  0  0  1  0  0  0
0  0  0  0  0  1  0  0
0  0  0  0  0  0  1  0
0  0  0  0  0  0  0  1
```

```
B
.2 .8
.8 .2
.2 .8
.8 .2
.2 .8
.8 .2
.2 .8
.2 .8
R
-.1 -.1 -10
-.1 -.1 -10
-.1 -.1 -10
-.1 -.1 -10
-.1 -.1 -10
-.1 -.1  10
-.1 -.1 -10
-.1 -.1 -10
```
---------------------------------------------------------------------------------------------------------------------------

### 4.1. Resultados de la determinación del estado actual

Para solicitar esta operación, se ejecuta el programa de la siguiente forma:

$ pomdp robot1.dat

Donde robot1.dat contiene lo siguiente:

```
-m P_Q
-r robot.pom
-l log.dat
-so 5
-o robot.so
-sa 4
-a robot.sa
-t archivo
```

En este caso, la opción -m con una valor de "P_Q" indica la operación de determinación del estado actual; "robot.pom" es el archivo de la red; "log.dat" es un archivo para guardar los resultados; 5 es el número de observaciones de la secuencia; "robot.so" es el archivo de las observaciones; 4 es el número de acciones realizadas; "robot.sa" es el archivo de acciones; y finalmente "-t archivo" indica que todas las probabilidades son obtenidas a partir de archivos (la otra opción es "azar"). Para el ejemplo, el archivo de observaciones (robot.so) contiene la secuencia:

libre pared libre pared pared

y el archivo de acciones (robot.sa) contiene la secuencia:

der der der der

En el archivo "log.dat" se obtuvieron los siguientes resultados:

```
*** SOLUCION ***
Distribucion del estado actual:
0.125000 0.125000 0.125000 0.125000 0.125000 0.125000 0.125000 0.125000
despues de o:libre
0.058824 0.235294 0.058824 0.235294 0.058824 0.235294 0.058824 0.058824
despues de a:der
0.005882 0.076471 0.217647 0.076471 0.217647 0.076471 0.217647 0.111765
despues de o:pared
```

```
0.007105 0.023091 0.262877 0.023091 0.262877 0.023091 0.262877 0.134991
despues de a:der
0.000710 0.008703 0.047069 0.238899 0.047069 0.238899 0.047069 0.371581
despues de o:libre
0.000289 0.014155 0.019138 0.388532 0.019138 0.388532 0.019138 0.151080
despues de a:der
0.000029 0.001675 0.014653 0.056077 0.351592 0.056077 0.351592 0.168304
despues de o:pared
0.000032 0.000458 0.016021 0.015328 0.384410 0.015328 0.384410 0.184013
despues de a:der
0.000003 0.000074 0.002014 0.015951 0.052236 0.347502 0.052236 0.529983
despues de o:pared
0.000004 0.000026 0.002769 0.005483 0.071817 0.119440 0.071817 0.728644
```

que concuerdan con los resultados esperados; esto es, el estado actual más probable es el estado 7 despues de la secuencia de observaciones-acciones.

**4.2. Resultados de la determinación de la política óptima de acciones**

Para solicitar esta operación, se ejecuta el programa de la siguiente forma:

$ pomdp robot2.dat

Donde robot2.dat contiene lo siguiente:

```
-m P_A
-r robot.pom
-l log.dat
-so 5
-o robot.so
-sa 4
-a robot.sa
-t archivo
-tol 0.0001
```

En este caso, la operación "P_A" indica la determinación de la política óptima de acciones, y "tol 0.0001" indica que se fija un umbral de 0.0001 para la determinación de los cambios de valor de la función v del algoritmo iteración de valor. En el programa se considera un valor de descuento $\gamma$ de 0.9. Al correr el programa, en el archivo "log.dat" se obtuvieron los siguientes resultados:

```
*** SOLUCION ***
Funcion v(s):
55.43 62.40 70.23 79.02 88.90 100.00 88.90 79.02
Politica de acciones optima:
der der der der der paro izq izq
```

Que corresponde con la solución intuitiva para llevar el robot a la posición p5.

**4.3. Resultados de la determinación de la probabilidad de observación de una secuencia**

Supóngase que el robot se encuentra inicialmente en la posición p3 y sabemos que la secuencia de observación-acción es la siguiente:

libre der pared der libre der pared der pared

Para determinar la probabilidad de que esa secuencia haya sido generada por el modelo, se ejecuta el programa de la siguiente forma:

$ pomdp robot3.dat

Donde robot3.dat contiene lo siguiente:

```
-m P_O
-r robot3.pom
-l log.dat
-so 5
-o robot.so
-sa 4
-a robot.sa
-t archivo
```

En este caso, la operación “P_O” indica la determinación de la probabilidad de la secuencia de observación-acción; el archivo robot3.pom se diferencía de robot.pom en que el vector PI toma los valores (0 0 0 1 0 0 0 0). Al correr el programa, en el archivo “log.dat” se obtuvieron los siguientes resultados:

```
*** SOLUCION ***
alfa
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.800000 0.016000 0.001280 0.000026 0.000001
0.000000 0.576000 0.014400 0.002074 0.000184
0.000000 0.000000 0.414720 0.010886 0.000591
0.000000 0.000000 0.000000 0.298598 0.031726
0.000000 0.000000 0.000000 0.000000 0.214991
P(O) = 0.247493
```

**4.4. Resultados del entrenamiento**

Para generar una secuencia larga observación-acción se implementó una opción de simulación del modelo. Para correr esta opción se ejecutó el programa pomdp de la forma:

$ pomdp robot_s.dat

Donde robot_s.dat contiene lo siguiente:

```
-m SIMULA
-r robot_s.pom
-l log.dat
-so 51
-o robot_s.so
-sa 50
-a robot_s.sa
-t archivo
```

En este caso, la opción “SIMULA” efectúa la operación deseada, y la secuencia de observaciones y acciones se deposita en los archivos “robot_s.so” y robot_s.sa” respectivamente. El archivo robot_s.pom modela el robot móvil con actuadores y sensores perfectos, iniciando en la posición p0. A continuación se muestra el resultado del programa cuando se solicita la opción de entrenamiento:

$ pomdp robot4.dat

Donde robot4.dat contiene lo siguiente:

```
-m ENTRENA
-r robot4.pom
-l log.dat
-so 5
```

```
-o robot_s.so
-sa 4
-a robot_s.sa
-t archivo
-tol 0.0001
-i 10
```

En este caso, la operación “ENTRENA” indica la opción de entrenamiento; el archivo robot4.pom se diferencia de robot.pom en que los valores de PI, A y B son arbitrarios; la opcion “i 10” fija el número máximo de iteraciones del algoritmo a 10. Al correr el programa, en el archivo “log.dat” se obtuvieron los siguientes resultados:

```
POMDP
N
8
Q
0 1 2 3 4 5 6 7
M
2
V
libre pared
X
3
LX
izq der paro
PI
0.125000 0.125000 0.125000 0.125000 0.125000 0.125000 0.125000 0.125000
A
1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.400000 0.600000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.800000 0.200000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.700000 0.300000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.200000 0.800000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.300000 0.700000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.500000 0.500000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.200000 0.800000
0.200000 0.800000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.300000 0.700000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.400000 0.600000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.500000 0.500000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.400000 0.300000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.200000 0.800000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.500000 0.500000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000
1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000
B
0.549773 0.450227
0.869091 0.130909
0.560082 0.439918
0.680536 0.319464
0.497600 0.502400
0.711739 0.288261
0.434959 0.565041
0.374375 0.625625
R
-0.100000 -0.100000 -10.000000
-0.100000 -0.100000 -10.000000
```

```
-0.100000 -0.100000 -10.000000
-0.100000 -0.100000 -10.000000
-0.100000 -0.100000 -10.000000
-0.100000 -0.100000 10.000000
-0.100000 -0.100000 -10.000000
-0.100000 -0.100000 -10.000000
-----
Entrena
SO
pared libre libre pared libre pared libre pared libre pared pared libre pared
libre pared pared libre pared libre libre pared libre pared pared pared pared
pared pared pared pared pared pared pared pared pared pared pared pared pared
pared pared pared pared pared pared pared pared pared pared pared pared
SA
der paro der izq der izq izq der der paro der izq izq der paro der der der paro
izq der der der der paro paro der der paro der izq paro der der der der der der
der der izq der paro der paro paro der izq der izq

*** SOLUCION ***
P(O|M )=0.000000000000073542950925054352
P(O|M')=0.000000014686119428925121000000
P(O|M')=0.015100452107289190000000000000
P(O|M')=0.352861017362385950000000000000
P(O|M')=0.962645930967017890000000000000
P(O|M')=0.999891015519840830000000000000
P(O|M')=0.999999998061582130000000000000
P(O|M')=0.999999999999999560000000000000
P(O|M')=0.999999999999999000000000000000
PI
1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
A
1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.074580 0.925420
0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.200330 0.799670
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000
1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000
B
0.000000 1.000000
1.000000 0.000000
0.000000 1.000000
1.000000 0.000000
0.000000 1.000000
1.000000 0.000000
0.000000 1.000000
0.000000 1.000000
```

Se observa que el valor de probabilidad de observación de la secuencia cambia 0.000000000000073 a un valor de 0.999999999999999 en 10 iteraciones del algoritmo. Las matrices A, B y el vector PI se acercan a los correspondientes que se utilizaron para realizar la simulación. Es conveniente señalar que el algoritmo de entrenamiento no cambia los valores de probabilidad de 0 o de 1 y es posible que quede atrapado en mínimos locales.

En otras pruebas de entrenamiento realizadas con valores de PI, A y B al azar (opción -t con valor "AZAR" en el archivo robot4.dat), se obtuvieron otros valores de PI, A y B completamente diferentes. En unas ocasiones la probabilidad de observación de la secuencia alcanzaba un valor cercano a la unidad, mientras que en otras ocasiones alcanzaba valores significativamente menores. Cuando se aumento el número de observaciones-acciones (v.g. 500) la probabilidad de observación de la secuencia alcanzaba un valor muy pequeño (< 1E-6), probablemente debido a errores numéricos acumulados.

En el siguiente ejemplo que se muestra, se puede observar que las acciones permiten transiciones a varios estados no vecinos (con referencia a la figura 1), el estado inicial es diferente al utilizado en la simulación, así como también son diferentes las probabilidades de observación a las utilizadas en la simulación.

```
POMDP
N
8
Q
0 1 2 3 4 5 6 7
M
2
V
libre pared
X
3
LX
izq der paro
PI
0.224534 0.019642 0.240006 0.142014 0.098746 0.059070 0.002611 0.213377
A
0.067895 0.099276 0.300990 0.134409 0.003089 0.063784 0.134472 0.196085
0.056607 0.214563 0.210738 0.286833 0.001922 0.173431 0.000763 0.055143
0.030909 0.236522 0.059313 0.145872 0.142049 0.132832 0.235841 0.016660
0.074329 0.233905 0.079433 0.223331 0.023071 0.174201 0.159716 0.032012
0.108281 0.059041 0.091134 0.205831 0.067439 0.078519 0.187545 0.202211
0.150450 0.185673 0.139371 0.000695 0.062549 0.044630 0.134878 0.281753
0.142383 0.110948 0.028553 0.267693 0.032907 0.060674 0.188585 0.168256
0.180290 0.129707 0.011838 0.213140 0.038640 0.111051 0.137261 0.178073
0.048770 0.062701 0.168955 0.157238 0.136955 0.173247 0.086786 0.165349
0.018555 0.172724 0.018930 0.287307 0.166020 0.160704 0.077333 0.098429
0.184360 0.163271 0.054354 0.168498 0.154206 0.049257 0.054964 0.171089
0.076839 0.281035 0.057422 0.218156 0.032073 0.017115 0.099905 0.217455
0.106807 0.087030 0.035739 0.247963 0.132209 0.191078 0.169773 0.029400
0.137674 0.031230 0.144004 0.219225 0.150368 0.034171 0.129041 0.154287
0.170463 0.019874 0.172277 0.107416 0.081360 0.083523 0.163404 0.201681
0.253023 0.118376 0.172596 0.014092 0.156559 0.026286 0.223995 0.035074
0.052248 0.150435 0.099253 0.022859 0.228039 0.094969 0.175820 0.176377
0.133924 0.093394 0.056894 0.068743 0.125926 0.149056 0.248375 0.123688
0.107631 0.015178 0.107888 0.243389 0.131105 0.055138 0.235660 0.104012
0.081616 0.140766 0.148663 0.000599 0.195213 0.210825 0.024026 0.198291
0.110679 0.126321 0.170012 0.006659 0.156328 0.070309 0.182122 0.177570
0.175102 0.177020 0.158875 0.149124 0.031493 0.096724 0.099244 0.112418
0.108584 0.118678 0.053702 0.167924 0.096393 0.145985 0.123155 0.185579
0.065814 0.046008 0.223443 0.135580 0.186962 0.095242 0.086842 0.160109
B
0.526572 0.473428
0.353645 0.646355
0.397696 0.602304
0.446571 0.553429
0.732926 0.267074
0.556206 0.443794
```

```
0.167259 0.832741
0.698199 0.301801
R
-0.100000 -0.100000 -10.000000
-0.100000 -0.100000 -10.000000
-0.100000 -0.100000 -10.000000
-0.100000 -0.100000 -10.000000
-0.100000 -0.100000 -10.000000
-0.100000 -0.100000 10.000000
-0.100000 -0.100000 -10.000000
-0.100000 -0.100000 -10.000000
-----
Entrena
SO
pared libre libre pared libre pared libre pared libre pared pared libre pared
libre pared pared libre pared libre libre pared libre pared pared pared pared
pared pared pared pared pared pared pared pared pared pared pared pared pared
pared pared pared pared pared pared pared pared pared pared pared pared
SA
der paro der izq der izq izq der der paro der izq izq der paro der der der paro
izq der der der der paro paro der der paro der izq paro der der der der der der
der der izq der paro der paro paro der izq der izq

*** SOLUCION ***
P(O|M )=0.0000000000000001076304608798611
P(O|M')=0.0000000000031158848414403320000
P(O|M')=0.0000000000040946445521015305000
...
P(O|M')=0.9999999999587016570000000000000
PI
0.000000 0.000000 0.998295 0.000000 0.000000 0.000000 0.001705 0.000000
A
0.000000 0.000000 0.000000 0.000000 0.003592 0.000000 0.000000 0.996408
0.000000 0.249312 0.000000 0.495997 0.000000 0.254690 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.654330 0.000000 0.000000 0.345670
0.000000 0.413915 0.000000 0.292429 0.000000 0.293655 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000
0.000000 0.813929 0.000000 0.002594 0.000000 0.183477 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.023287 0.000000 0.000000 0.976713
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000
0.000020 0.198085 0.000000 0.551191 0.000000 0.250704 0.000000 0.000000
0.000000 0.359767 0.000000 0.537809 0.000000 0.102424 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 1.000000 0.000000 0.000000 0.000000
0.000000 0.648255 0.000000 0.339860 0.000000 0.011885 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000
0.000000 0.139548 0.000000 0.802348 0.000000 0.058104 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000
1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.583960 0.000000 0.173027 0.000000 0.243013 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.559825 0.000000 0.001717 0.000000 0.438458 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.133571 0.000000 0.000000 0.866429
0.000000 0.624321 0.000000 0.284495 0.000000 0.091184 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.162984 0.000000 0.000000 0.837016
B
0.000000 1.000000
0.000000 1.000000
0.000000 1.000000
0.000000 1.000000
1.000000 0.000000
0.000000 1.000000
0.000000 1.000000
1.000000 0.000000
```

**5. Conclusiones**

Con referencia al problema de determinación del estado actual (problema 1) se puede observar que una adecuada secuencia de acciones puede ayudar a reducir la incertidumbre del estado actual. En el caso de ejemplo presentado, el problema de localización del robot móvil se resuelve satisfactoriamente. Las técnicas de elección de acción, de acuerdo a la reducción de entropía asociada a las acciones parece prometedora.

Con referencia al problema de determinar una política óptima de acción, el algoritmo de iteración de valores trabaja satisfactoriamente. El problema de seguir dicha política consiste precisamente en que el robot tenga certeza con respecto al estado en el que se encuentra; es decir, que solamente un estado tenga una alta probabilidad de ser el estado actual.

La determinación de la probabilidad de una secuencia observación-acción dado un modelo se usa para evaluar el aprendizaje de los parámetros del modelo. El algoritmo de aprendizaje programado (Baum-Welch) no cambia los valores de probabilidad que son 0 o 1. Usando esto, es posible facilitar el aprendizaje asignando valores diferentes de 0 y 1 a los valores de probabilidad que se desea aprender. Si se restringen de esta manera los parámetros a aprender, se obtiene un desempeño satisfactorio del algoritmo. En el caso no restringido el aprendizaje se complica y frecuentemente se encuentran mínimos locales. Además el algoritmo programado parece presentar problemas numéricos cuando se ingresan secuencias de observación-acción con mas de 500 elementos.

**6. Referencias**

Bellman, R. E. 1957.
"Dynamic Programming". Princeton University Press, Princeton, NJ, 1957.

Koenig, S.; y Simmmons, G. 1998.
"Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models". En "Artificial Intelligence and Mobile Robots", edited by David Kortenkamp, R. Peter Bonasso and Robin Murphy, AAAI Press/The MIT Press.

Nourbakhsh, I.; Powers, R.; y Birchfield, S. 1995.
"Dervish: An Office-Navigating Robot". AI Magazine 16(2): 53-60.

## Apendice A. Listado del programa

```
/*
   Archivo:    pomdp.c
   Propósito:  Programa de entrenamiento y reconocimiento de un Proceso
               de Decisión Parcialmente observable (POMDP)
   Autor:      Leonardo Romero Muñoz
   Ultima modificacion: 4 de mayo de 1999
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>


/* Archivo de parametros de entrada para la configuraci¢n de la red*/
#define MAX_S   40                  /* Caracteres de las cadenas */

#define MAX_N   10                  /* numero de nodos */
#define MAX_O   10                  /* numero de observaciones */
#define MAX_SO  2001                /* numero de observaciones de la secuencia*/
#define MAX_A   10                  /* numero de acciones */
#define MAX_SA  2000                /* numero de acciones de la secuencia*/

double PI[MAX_N];                   /* vector de probabilidades iniciales*/
double A[MAX_A][MAX_N][MAX_N];      /* matriz de transicion de estados */
double B[MAX_N][MAX_O];             /* matriz de probs. de observaciones */
double R[MAX_N][MAX_A];             /* matriz de recompensas */
char   *LQ[MAX_N];                  /* etiquetas de los estados */
char   *LO[MAX_O];                  /* etiquetas de las observaciones */
char   *LA[MAX_O];                  /* etiquetas de las acciones */
int    O[MAX_SO];                   /* secuencia de observaciones (indices)*/
int    X[MAX_SA];                   /* secuencia de acciones */

double a[MAX_N][MAX_SO];                     /* valores alfa */
double b[MAX_N][MAX_SO];                     /* valores beta */
double xC [MAX_N][MAX_N][MAX_SO];
double xN [MAX_N][MAX_SO];

enum Control { P_Q, P_A, P_O, SIMULA, ENTRENA } ;
enum Tipo { AZAR, ARCHIVO } ;

typedef struct datos_entrada
{
   enum Control modo;
   int n_n, n_o, n_so;  /* nodos, observaciones, sec. observaciones */
   int n_a, n_sa;       /* acciones, sec. acciones */
   double tol;          /* tolerancia permitida para el entrenamiento */
   int iteraciones;     /* maximo numero de iteraciones permitidas */
   int iter_show;       /* iteraciones para mostrar avance */
   int iter_guarda;     /* iteraciones para guardar valores en archivo */
   enum Tipo ini_probs; /* probs de archivo o aleatorios */
   char arch_red[MAX_S];
   char arch_O[MAX_S];
   char arch_A[MAX_S];
   char arch_log[MAX_S];
} PARAM;

#define p(text) printf(text);
void help()
{
p("Forma de ejecutarse:\n");
p("\n");
p("POMDP [-h]\n");
p("POMDP <archivo.dat>\n");
p("\n");
p("Donde <archivo.dat> tiene la forma:                     (Comentario)\n");
p("\n");
p("-m <P_Q|P_A|SIMULA|ENTRENA>                             operacion\n");
p("-r <archivo de definicion de la red>                    (v.g. robot.pomdp");
p("-l <archivo de resultados>                              (v.g. log.dat)");
p("-so #observaciones_de_la_secuencia                      (v.g. 5)\n");
p("-sa #acciones_de_la_secuencia                           (v.g. 4)\n");
p("-tol #tolerancia_durante_el_aprendizaje                 (v.g. 0.001)\n");
p("-i #maximo_de_iteraciones                               (v.g. 10000000)\n");
p("-s #iteraciones_para_mostrar_resultados_en_pantalla     (v.g. 100)\n");
p("-g #iteraciones_para_guardar_valores_calculados         (v.g. 1000)\n");
p("-t <archivo|azar>                                       Generaci¢n de PI,A,B\n");
```

```
p("Autor: Leonardo Romero M., 4\\mayo\\1999\n");
getchar();
}

enum Control interpreta_modo(char dato[])
{
   enum Control modo;
   if(strcmp(dato,"P_Q")==0 )          modo = P_Q;
   else if(strcmp(dato,"P_A")==0 )     modo = P_A;
   else if(strcmp(dato,"P_O")==0 )     modo = P_O;
   else if(strcmp(dato,"SIMULA")==0 )  modo = SIMULA;
   else if(strcmp(dato,"ENTRENA")==0 ) modo = ENTRENA;
   else {
      printf("Modo de operacion desconocido: %s\n", dato);
      exit(1);
   }
   return modo;
}

PARAM lee_parametros (char filename[])
{
   FILE *fp;
   char comando[255], dato[255];
   PARAM p;

   strcpy(p.arch_log, "");
   if( (fp=fopen(filename,"r"))!=NULL ) {
      while( fscanf(fp,"%s",comando)==1) {
        if (strcmp(comando,"-m")==0 && fscanf(fp,"%s",dato)==1) {
           p.modo = interpreta_modo(dato);        }
        else if (strcmp(comando,"-r")==0) fscanf(fp,"%s",p.arch_red);
        else if (strcmp(comando,"-l")==0) fscanf(fp,"%s",p.arch_log);
        else if (strcmp(comando,"-so")==0) fscanf(fp,"%d", &p.n_so);
        else if (strcmp(comando,"-o")==0) fscanf(fp,"%s",p.arch_O);
        else if (strcmp(comando,"-sa")==0) fscanf(fp,"%d", &p.n_sa);
        else if (strcmp(comando,"-a")==0) fscanf(fp,"%s",p.arch_A);
        else if (strcmp(comando,"-tol")==0) fscanf(fp,"%lf",&p.tol);
        else if (strcmp(comando,"-i")==0) fscanf(fp,"%d" ,&p.iteraciones);
        else if (strcmp(comando,"-s")==0) fscanf(fp,"%d" ,&p.iter_show);
        else if (strcmp(comando,"-g")==0) fscanf(fp,"%d" ,&p.iter_guarda);
        else if ( (strcmp(comando,"-t")==0)  && (fscanf(fp,"%s",dato)== 1)) {
           if(strcmp(dato,"azar")==0 ) p.ini_probs = AZAR; else p.ini_probs = ARCHIVO;
        }
        else printf("Error, comando :%s: desconocido\n",comando);
     }
     fclose(fp);
     if ( p.n_so > MAX_SO || p.n_sa > MAX_SA) {
        printf("Error, dimensiones de la red excedidas. Recompile el programa\n");
        exit(1);
     }
     return p;
   } else {
     printf("\nError en la apertura del archivo %s..",filename);
     exit(1);
   }
}

/* Lee un número de un archivo */
double lee_num(FILE *fp)
{
   double dato;

   if (fscanf(fp,"%lf",&dato) == 1)
      return dato;
   else {
      printf("Error, se esperaba un numero\n");
      exit(1);
      return 0;
   }
}

/* Regresa un número aleatorio entre 0 y 1 */
double random_double()
{
       return  (double) rand() / (double) RAND_MAX;
}

/* regresa un valor entero aleatorio entre 0 y max */
int random_int(int max)
{
```

```
        double n;

        n = random_double() * (double) max + 0.5;
        return (int) n;
}

/* Regresa un valor de probabilidad a partir de un archivo o generado al azar */
double lee_prob(FILE *fp, enum Tipo t)
{
  double dato;

  dato = lee_num(fp);
  if (t == AZAR)
     dato = random_double();
  return dato;
}

void lee_mat_A(FILE *fp, PARAM p)
{
  int a, i, j;

  for(a=0; a < p.n_a; a++)
     for(i=0; i < p.n_n; i++)
        for(j=0; j < p.n_n; j++)
           A[a][i][j] = lee_prob(fp, p.ini_probs);
}

void lee_mat_B(FILE *fp, PARAM p)
{
  int n, o;

  for(n=0; n < p.n_n; n++)
     for(o=0; o < p.n_o; o++)
           B[n][o] = lee_prob(fp, p.ini_probs);
}

void lee_mat_R(FILE *fp, PARAM p)
{
  int n, a;

  for(n=0; n < p.n_n; n++)
     for(a=0; a < p.n_a; a++)
           R[n][a] = lee_num(fp);
}

void lee_vector_int(FILE *fp, int n, int V[])
{
  int i;

  for(i=0; i < n; i++)
       V[i] = (int) lee_num(fp);
  fclose(fp);
}

void lee_vector_double(FILE *fp, int n, double V[], enum Tipo t)
{
  int i;

  for(i=0; i < n; i++)
     V[i] = lee_prob(fp, t);
}

void lee_etiquetas(FILE *fp, int n, char *V[])
{
  int i;
  char string[MAX_S];

  for(i=0; i < n; i++) {
        fscanf(fp, "%s", string);
        V[i] = (char *) malloc(strlen(string) + 1);
        strcpy(V[i],string);
  }
}

int ind_eti(char eti[], int n, char *L[])
{
  int i, i_o;

  i_o = -1;
  for(i = 0; i < n; i++)
```

```
      if(strcmp(eti, L[i]) == 0)
         i_o = i;
   if (i_o < 0) {
      printf("etiqueta desconocida: %s\n",eti);
      exit(1);
   }
   return i_o;
}

/*
    Lee el archivo "file" de n_s etiquetas y coloca sus indices en el
    arreglo a[]. El diccionario de etiquetas L contiene un maximo de n
*/
void lee_sec_eti(char file[], int n, char *L[], int n_s, int a[])
{
   FILE *fp;
   int i;
   char o[MAX_S];

   if ((fp=fopen(file,"r")) == NULL) {
      printf("\nError en la apertura del archivo %s..", file);
      exit(1);
   }
   for(i=0; i < n_s; i++) {
         fscanf(fp, "%s", o);
         a[i] = ind_eti(o, n, L);
   }
   fclose(fp);
}

void normaliza(int n, double PI[])
{
    int i;
    double sum = 0.0;

    for(i=0; i<n; i++)
           sum += PI[i];
    for(i=0; i<n; i++)
           PI[i] /= sum;
}

void normaliza_A(PARAM p)
{
    int a, i, j;
    double acc;

    for(a= 0; a < p.n_a; a++)
           for(i = 0; i < p.n_n; i++) {
                  acc = 0.0;
               for(j = 0; j < p.n_n; j++)
                         acc += A[a][i][j];
               for(j = 0; j < p.n_n; j++)
                         A[a][i][j] /= acc;
           }
}

void normaliza_B( PARAM p)
{
    int n,o;
    double acc;

    for(n= 0; n < p.n_n; n++) {
           acc = 0.0;
           for(o = 0; o < p.n_o; o++)
                        acc += B[n][o];
           for(o = 0; o < p.n_o; o++)
                        B[n][o] /= acc;
    }
}

PARAM lee_red(PARAM p)
{
    FILE *fp;
    char comando[255];

    /* Seed the random-number generator with current time so that
     * the numbers will be different every time we run.    */
    srand( (unsigned)time( NULL ) );

    if( (fp=fopen(p.arch_red,"r"))!=NULL ) {
```

```
      while( fscanf(fp,"%s",comando)==1) {
        if (strcmp(comando,"POMDP")==0) ;
        else if (strcmp(comando,"N")==0) fscanf(fp,"%d",&p.n_n);
        else if (strcmp(comando,"Q")==0) lee_etiquetas(fp, p.n_n, LQ);
        else if (strcmp(comando,"M")==0) fscanf(fp,"%d",&p.n_o);
        else if (strcmp(comando,"V")==0) lee_etiquetas(fp, p.n_o, LO);
        else if (strcmp(comando,"X")==0) fscanf(fp,"%d",&p.n_a);
        else if (strcmp(comando,"W")==0) lee_etiquetas(fp, p.n_a, LA);
        else if (strcmp(comando,"Pi")==0) lee_vector_double(fp, p.n_n, PI, p.ini_probs);
        else if (strcmp(comando,"A")==0)  lee_mat_A(fp, p);
        else if (strcmp(comando,"B")==0)  lee_mat_B(fp, p);
        else if (strcmp(comando,"R")==0)  lee_mat_R(fp, p);
        else printf("Error, comando :%s: desconocido\n",comando);
     }
     fclose(fp);
     if ( p.n_n > MAX_N || p.n_o > MAX_O  || p.n_a > MAX_A) {
        printf("Error, dimensiones de la red excedidas. Recompile el programa\n");
        exit(1);
     }
   } else {
          printf("Error, al abrir el archivo de red: %s\n",p.arch_red);
          exit(1);
   }
   if (p.ini_probs == AZAR) {
      normaliza(p.n_n, PI);
      normaliza_A(p);
         normaliza_B(p);
   }
   return p;
}

void modo_procesamiento(enum Control m, char letrero[])
{
   switch (m) {
    case P_Q: strcpy(letrero, "Estimacion de estado actual E(Q|O,X,M)");  break;
    case P_A: strcpy(letrero, "Determinar una politica optima de acciones");break;
    case P_O: strcpy(letrero, "Determinar la prob. de una secuencia de observaciones");break;
    case SIMULA: strcpy(letrero, "Simulacion");   break;
    case ENTRENA: strcpy(letrero, "Entrena"); break;
   }
}

void init_log(char file[])
{
   FILE *f;

   f = fopen(file, "w");
   fclose(f);
}

void print_log(char file[], char texto[])
{
   FILE *f;

   printf("%s",texto);
   if(strcmp(file,"")!= 0) {
      f = fopen(file, "a");
      fprintf(f,"%s", texto);
      fclose(f);
   }
}

void print_eti(char file[], char label[], int n, char *L[])
{
   FILE *f;
   int i;

   f = fopen(file, "a");
   printf("%s\n",label);
   fprintf(f,"%s\n", label);
   for(i=0; i < n; i++) {
      printf("%s ",L[i]);
      fprintf(f,"%s ", L[i]);
   }
   fprintf(f,"\n");
   fclose(f);
}

void  print_vector_double(char file[], char label[], int n, double a[])
{
```

```
  int i;
  char texto[255];

  sprintf(texto,"%s\n", label);
  print_log(file,texto);
  for(i=0; i < n; i++) {
     sprintf(texto,"%lf ", a[i]);
     print_log(file, texto);
  }
  print_log(file, "\n");
}

void print_mat_A(PARAM p)
{
  int a, i, j;
  char texto[255];

  print_log(p.arch_log, "A\n");
  for(a=0; a < p.n_a; a++)
     for(i=0; i < p.n_n; i++) {
        for(j=0; j < p.n_n; j++) {
           sprintf(texto, "%lf ", A[a][i][j]);
           print_log(p.arch_log, texto);
        }
        print_log(p.arch_log, "\n");
     }
}

void print_mat_B( PARAM p)
{
  int n,o;
  char texto[255];

  print_log(p.arch_log, "B\n");
  for(n=0; n < p.n_n; n++) {
     for(o=0; o < p.n_o; o++) {
        sprintf(texto, "%lf ", B[n][o]);
        print_log(p.arch_log, texto);
     }
     print_log(p.arch_log, "\n");
  }
}

void print_mat_R( PARAM p)
{
  int n,a;
  char texto[255];

  print_log(p.arch_log, "R\n");
  for(n=0; n < p.n_n; n++) {
     for(a=0; a < p.n_a; a++) {
        sprintf(texto, "%lf ", R[n][a]);
        print_log(p.arch_log, texto);
     }
     print_log(p.arch_log, "\n");
  }
}

void print_matriz_double(char file[], char titulo[], int n_r, int n_c, int max_c, double V[])
{
  int r, c;
  char texto[255];

  sprintf(texto,"%s\n",titulo);
  print_log(file, texto);
  for(r=0; r < n_r; r++) {
     for(c=0; c < n_c; c++) {
        sprintf(texto, "%lf ", V[r*max_c+c] );
        print_log(file, texto);
     }
     print_log(file, "\n");
  }
}


void print_header(PARAM p)
{
   char modo[255];
   char texto[255];
```

```
   print_log(p.arch_log,"POMDP\n");
   sprintf(texto,"N\n%d\n",p.n_n);
   print_log(p.arch_log,texto);
   print_eti(p.arch_log, "Q", p.n_n, LQ);
   sprintf(texto,"M\n%d\n",p.n_o);
   print_log(p.arch_log,texto);
   print_eti(p.arch_log, "V", p.n_o, LO);
   sprintf(texto,"X\n%d\n",p.n_a);
   print_log(p.arch_log,texto);
   print_eti(p.arch_log, "LX", p.n_a, LA);
   print_vector_double(p.arch_log,"PI", p.n_n, PI);
   print_mat_A(p);
   print_mat_B(p);
   print_mat_R(p);
   print_log(p.arch_log,"-----\n");
   modo_procesamiento (p.modo, modo);
   sprintf(texto,"%s\n",modo);
   print_log(p.arch_log,texto);
}

void print_sec_eti(char file[], char label[], int n, char *L[], int n_s, int a[])
{
  int i;
  char texto[MAX_S];

  if (strcmp(label, "") != 0) {
     sprintf(texto,"%s\n", label);
     print_log(file,texto);
  }
  for(i=0; i < n_s; i++) {
     sprintf(texto,"%s ", L[a[i]]);
     print_log(file, texto);
  }
  print_log(file, "\n");
}

void copy_vec_double(int n, double O[], double I[])
{
        int i;

        for(i=0;i<n;i++)
               O[i] = I[i];
}

void actualiza_edo_por_accion(int a, int n, double PI[])
{
   double E[MAX_N], sum;
   int i, j;

   for(j=0; j < n; j++) {
          sum = 0.0;
          for(i=0; i < n; i++)
                 sum += A[a][i][j] * PI[i];
          E[j] = sum;
   }
   normaliza(n,E);
   copy_vec_double(n, PI,E);
}

void actualiza_edo_por_observacion(int o, int n, double PI[])
{
   int i;

   for(i=0; i < n; i++) {
          PI[i] = B[i][o] * PI[i];
   }
   normaliza(n,PI);
}

/*-------------------------------------------------------------------
   Cálculo de la distribución del estado actual dadas observaciones y
   acciones
-------------------------------------------------------------------*/
void calcula_edo_actual(PARAM p)
{
   int t = 0;
   char texto[255];

   sprintf(texto,"Distribucion del estado actual:");
   print_vector_double(p.arch_log,texto, p.n_n, PI);
```

```
   while(t < p.n_so || t < p.n_sa) {
        if (t < p.n_so) {
        actualiza_edo_por_observacion(O[t], p.n_n, PI);
               sprintf(texto,"despues de o:%s",LO[O[t]]);
        print_vector_double(p.arch_log,texto, p.n_n, PI);
        }
        if (t < p.n_sa) {
        actualiza_edo_por_accion(X[t], p.n_n, PI);
               sprintf(texto,"despues de a:%s",LA[X[t]]);
        print_vector_double(p.arch_log,texto, p.n_n, PI);
        }
        t++;
   }
}

/*----------------------------------------------------------------------
   Cálculo de la política óptima de acciones
----------------------------------------------------------------------*/
void calcula_politica_acciones(PARAM p)
{
       double V[MAX_N], Vn[MAX_N], sum, max_v, v, g=0.9;
       int n, i,j, a, no_cambios, max_a;
       char texto[255];

   for(n=0;n<p.n_n; n++)
         V[n] = 0.0;

   do {
       for(i=0;i<p.n_n; i++) {
              for(a=0; a<p.n_a; a++) {
                     sum = 0.0;
               for(j=0;j<p.n_n; j++)
                            sum += A[a][i][j] * V[j] ;
                     v = R[i][a] + g * sum;
                     if (a== 0) max_v = v;
                     if (v > max_v) max_v = v;
              }
              Vn[i] = max_v;
       }

       no_cambios = 0;
       for(i=0;i<p.n_n; i++) {
          if(fabs(V[i] - Vn[i]) >= p.tol)
                 no_cambios++;
          V[i] = Vn[i];
       }
   } while(no_cambios > 0);

   print_log(p.arch_log, "Funcion v(s):\n");
   for(i=0;i<p.n_n; i++) {
         sprintf(texto,"%.2lf ", V[i]);
         print_log(p.arch_log, texto);
   }
   print_log(p.arch_log, "\n");

   print_log(p.arch_log, "Politica de acciones optima:\n");
   for(i=0;i<p.n_n; i++) {
              for(a=0; a<p.n_a; a++) {
                     sum = 0.0;
               for(j=0;j<p.n_n; j++)
                            sum += A[a][i][j] * V[j];
                     v = R[i][a] + g * sum;
                     if (a== 0) { max_v = v; max_a = 0; }
                     if (v > max_v) { max_v = v; max_a = a; }
              }
              sprintf(texto,"%s ", LA[max_a]);
              print_log(p.arch_log, texto);
       }
       print_log(p.arch_log, "\n");
}

int genera_observacion(int n, double PI[], int n_o, double B[MAX_N][MAX_O])
{
       double ACC_PI[MAX_N], ACC_O[MAX_O], r;
       int i, j, q, o, encontre;

       for(i=0; i < n; i++)   /* Inicializa ACC_PI a ceros */
              ACC_PI[i] = 0;
       for(i=0; i < n_o; i++)  /* Inicializa ACC_O a ceros */
              ACC_O[i] = 0;
```

```
        for(i=0; i < n; i++)   /* ACC_PI es una funcion acumulada de PI*/
           for(j=i; j < n; j++)
                 ACC_PI[j] += PI[j];

        r = random_double();
        q = n - 1;
        encontre = 0;
    for(i=0; i < n && !encontre; i++) {
               if (r <= ACC_PI[i]) {
                       q = i;
                       encontre = 1;
               }
        }                                         /* q es el estado actual */

        for(i=0; i < n_o; i++) /* ACC_O es una funcion acumulada de B[q][]*/
           for(j=i; j < n_o; j++)
                 ACC_O[j] += B[q][j];

        r = random_double();
        o = n_o - 1;
        encontre = 0;
    for(i=0; i < n_o && !encontre; i++) {
               if (r <= ACC_O[i]) {
                       o = i;
                       encontre = 1;
               }
        }                                         /* o es el la observación realizada */
        return o;
}

/*---------------------------------------------------------------------
   Simulación del proceso para acciones aleatorias
---------------------------------------------------------------------*/
void simula(PARAM p)
{
   int t = 0;
   char texto[255];

   sprintf(texto,"Simulacion del proceso para acciones aleatorias:\n");
   print_log(p.arch_log,texto);
   while(t < p.n_so || t < p.n_sa) {
      if (t < p.n_so) {
         O[t] = genera_observacion(p.n_n, PI, p.n_o, B);
         actualiza_edo_por_observacion(O[t], p.n_n, PI);
      }
      if (t < p.n_sa) {
         X[t] = random_int(p.n_a - 1);
         actualiza_edo_por_accion(X[t], p.n_n, PI);
      }
      t++;
   }
   init_log(p.arch_O);
   init_log(p.arch_A);
   print_log(p.arch_log,"SO\n");
   print_sec_eti(p.arch_O, "",p.n_o, LO, p.n_so, O);
   print_log(p.arch_log,"SA\n");
   print_sec_eti(p.arch_A, "", p.n_a, LA, p.n_sa, X);
}

void calc_alfas(PARAM p)
{
  int n, t, i;
  double acc;

  for(n=0;n < p.n_n; n++)
    a[n][0] = PI[n] * B[n][O[0]];

  for(t=1; t < p.n_so; t++)
     for(n=0;n < p.n_n; n++) {
        acc = 0.0;
        for(i=0; i < p.n_n; i++)
           acc += a[i][t-1] * A[X[t -1]][i][n];
        a[n][t] = acc * B[n][O[t]];
     }
}

void calc_betas(PARAM p)
{
  int n, t, i, T;
  double acc;
```

```
  T = p.n_so - 1;
  for(n=0;n < p.n_n; n++)
    b[n][T] = 1;

  for(t=T-1; t >= 0; t--)
     for(n=0;n < p.n_n; n++) {
        acc = 0.0;
        for(i=0; i < p.n_n; i++)
           acc += b[i][t+1] * A[X[t]][n][i] * B[i][O[t+1]];
        b[n][t] = acc;
     }
}

/*--------------------------------------------------------------------
   Calculo de la probabilidad de una secuencia de observaciones
--------------------------------------------------------------------*/
double calc_p_O (PARAM p)
{
   int n, T;

   double prob = 0.0;

   calc_alfas(p);
   T = p.n_so - 1;
   for(n=0; n< p.n_n; n++)
      prob += a[n][T];
   return prob;
}

double calc_xC(PARAM p)
{
   int i, j, t;
   double acc;

   for(t = 0; t < p.n_so - 1; t++) {
      acc = 0.0;
      for(i = 0; i < p.n_n; i++)
         for(j=0; j < p.n_n; j++)
            acc += xC[i][j][t] = a[i][t] * A[X[t]][i][j] * B[j][O[t+1]] * b[j][t+1];
      for(i = 0; i < p.n_n; i++)
         for(j=0; j < p.n_n; j++)
            xC[i][j][t] /= acc;
   }
   return 0;
}

double calc_xN(PARAM p)
{
   int i, j, t;
   double acc;

   for(t = 0; t < p.n_so - 1; t++)
      for(i = 0; i < p.n_n; i++) {
         acc = 0.0;
         for(j=0; j < p.n_n; j++)
            acc += xC[i][j][t];
         xN[i][t] = acc;
      }
   return 0;
}

void actualiza_PI(PARAM p)
{
  int n;

  for(n=0; n< p.n_n; n++)
    PI[n] = xN[n][0];
}

void actualiza_A(PARAM p)
{
  int i,j,t,a;
  double num, den;

  for(i = 0; i < p.n_n; i++) {
     for(j=0; j < p.n_n; j++) {
        for(a=0; a < p.n_a; a++) {
           num = den = 0;
           for(t=0; t < p.n_so -1; t++) {
```

```
            if(a == X[t]) {
               num += xC[i][j][t];
               den += xN[i][t];
            }
         }
              if (den != 0.0)
            A[a][i][j] = num / den;
      }
    }
  }
}

void actualiza_B(PARAM p)
{
  int n,o,t;
  double num, den;

  for(n = 0; n < p.n_n; n++) {
     for(o=0; o < p.n_o; o++) {
        num = den = 0;
        for(t=0; t < p.n_so ; t++) {
           if(O[t] == o)
              num += xN[n][t];
           den += xN[n][t];
        }
            if (den != 0.0)
           B[n][o] = num / den;
     }
  }
}

/*-----------------------------------------------------------------------
   Entrenamiento
-----------------------------------------------------------------------*/
void entrena (PARAM p, double prob_ini)
{
  double prob1, prob0;
  char texto[255];
  int i = 0;

  prob1 = prob_ini;
  do {
     prob0 = prob1;
     calc_alfas(p);
     calc_betas(p);
     calc_xC(p);
     calc_xN(p);
     actualiza_PI(p);
     actualiza_A(p);
     actualiza_B(p);
     prob1 = calc_p_O(p);
     i++;
     sprintf(texto,"i=%d P(O|M')=%.30lf\n",i, prob1);
     print_log(p.arch_log, texto);
  } while ( ((prob1 - prob0) / prob1) > p.tol && i < p.iteraciones);
  print_vector_double(p.arch_log,"PI", p.n_n, PI);
  print_mat_A(p);
  print_mat_B(p);
  sprintf(texto,"i=%d P(O|M')=%.30lf\n",i, prob1);
  print_log(p.arch_log, texto);
}

/*-----------------------------------------------------------------------
   Procesamiento de la operación solicitada
-----------------------------------------------------------------------*/
void procesa (PARAM p)
{
   double prob;
   char texto[255];

   init_log(p.arch_log);
   print_header(p);
   switch (p.modo) {
    case P_Q: lee_sec_eti(p.arch_O, p.n_o, LO, p.n_so, O);
              print_sec_eti(p.arch_log, "SO",p.n_o, LO, p.n_so, O);
              lee_sec_eti(p.arch_A, p.n_a, LA, p.n_sa, X);
              print_sec_eti(p.arch_log, "SA", p.n_a, LA, p.n_sa, X);
              print_log(p.arch_log,"\n*** SOLUCION ***\n");
              calcula_edo_actual(p);
              break;
```

```
    case P_A: print_log(p.arch_log,"\n*** SOLUCION ***\n");
              calcula_politica_acciones(p);
              break;
    case P_O: lee_sec_eti(p.arch_O, p.n_o, LO, p.n_so, O);
              print_sec_eti(p.arch_log, "SO",p.n_o, LO, p.n_so, O);
              lee_sec_eti(p.arch_A, p.n_a, LA, p.n_sa, X);
              print_sec_eti(p.arch_log, "SA", p.n_a, LA, p.n_sa, X);
              print_log(p.arch_log,"\n*** SOLUCION ***\n");
              prob = calc_p_O(p);
              print_matriz_double(p.arch_log,"alfa", p.n_n,p.n_so, MAX_SO,(double *) a);
              sprintf(texto,"P(O) = %.30lf\n", prob);
              print_log(p.arch_log, texto);
              break;
    case SIMULA: print_log(p.arch_log,"\n*** SOLUCION ***\n");
              simula(p);
              break;
    case ENTRENA: lee_sec_eti(p.arch_O, p.n_o, LO, p.n_so, O);
              print_sec_eti(p.arch_log, "SO",p.n_o, LO, p.n_so, O);
              lee_sec_eti(p.arch_A, p.n_a, LA, p.n_sa, X);
              print_sec_eti(p.arch_log, "SA", p.n_a, LA, p.n_sa, X);
              print_log(p.arch_log,"\n*** SOLUCION ***\n");
              prob = calc_p_O(p);
              sprintf(texto,"i=0 P(O|M )=%.30lf\n", prob);
              print_log(p.arch_log, texto);
              entrena(p, prob);
              break;
   }
}

/*---------------------------------------------------------------------
   Función principal
---------------------------------------------------------------------*/
int main(int argc, char *argv[])
{
   PARAM p;

   if (argc >= 2){
      if(strcmp(argv[1],"-h") == 0) {
          help();
          exit(0);
      }
      p = lee_parametros(argv[1]);
      p = lee_red(p);
      procesa(p);
   } else {
      printf("Sintaxis: pomdp parametros.dat\n");
      printf("Sintaxis: pomdp -h\n");
   }
   return 0;
}
```