Selección de Comportamientos básicos de Conducción usando MDP-ProbLog y Webots[1]

Introducción a MDP-ProbLog y Webots

Héctor Avilés, Karina Arévalo

havilesa@upv.edu.mx

Universidad Politécnica de Victoria

Escuela de Invierno de Robótica 2022-2023

11-12 de enero del 2023

[1]En trabajo conjunto con Marco Negrete (FI-UNAM), Rubén Machucho (UPV), Alberto Reyes (IEE)

Cadena de Markov

- Una cadena de Markov es una tupla $\lambda = (\mathcal{S}, P, \pi)$ donde:
  - $\mathcal{S}$ es un conjunto finito de $n > 0$ estados[2] observables
  - $P : \mathcal{S} \times \mathcal{S} \to [0, 1]$ es la posibilidad de que un estado futuro s' sea alcanzado a partir del estado actual conocido $s \in \mathcal{S}$, tal que $\sum_{s' \in \mathcal{S}} P(s'|s) = 1$
  - $\pi : \mathcal{S} \to [0, 1]$ donde $\sum_{s \in \mathcal{S}} \pi(s) = 1$ es la probabilidad inicial para cada estado s

  que modela evolución de los estados de un sistema en tiempo discreto[3] considerando transiciones inciertas entre estados[4] a través del tiempo

[2] El estado de un sistema es una situación o condición específica distinguible del resto

[3] Tiempos espaciados y distintos

[4] Estos modelos suponen: i) que el estado actual es observable, ii) la propiedad de Markov $p(s_{t+1}|s_t) = p(s_{t+1}|s_t, s_{t-1}, ..., s_0)$ (el pasado es independiente del futuro dado el presente) y que el proceso es estacionario, ie, $p(s_{t+1} = s'|s_t = s) = p(s_1 = s'|s_0 = s)$ para $t = 1, 2, ..., T$ (la probabilidad de transición entre cada par de estados s y s' no cambia en el tiempo, así que las referencias al índice t se pueden remover como arriba)

Cadena de Markov

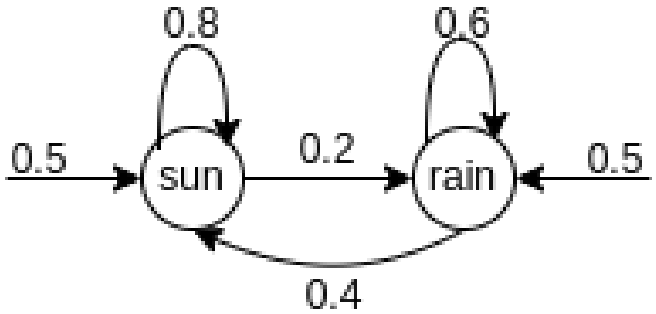
- La función de probabilidad conjunta de una secuencia de estados $s_{0:T} \in \mathcal{S}^{T+1}$ de longitud $T+1$ de una cadena de Markov λ es:

$$P(s_{0:T}; \lambda) = \pi(s_0; \lambda) \prod_{t=0}^{T-1} P(s_{t+1}|s_t; \lambda),$$

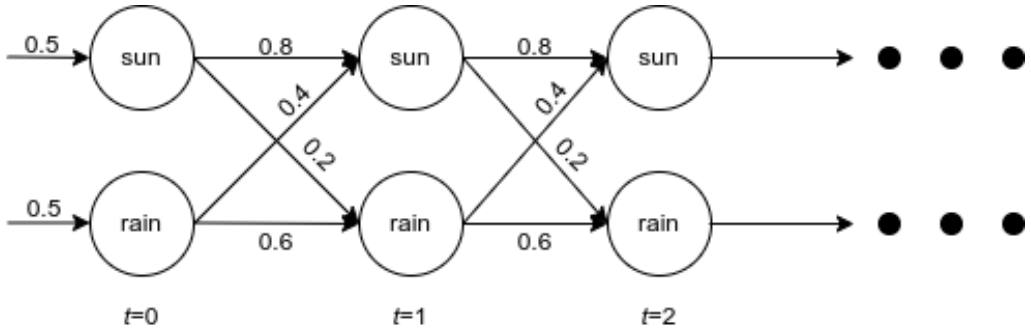
y la probabilidad $P(s_T = s; \lambda)$ de terminar en un estado s en el tiempo T es:

$$P(s_T = s; \lambda) = \sum_{s_{0:T}: s_T = s} \left[\pi(s_0; \lambda) \prod_{t=0}^{T-1} P(s_{t+1}|s_t; \lambda) \right]$$

Cadenas de Markov



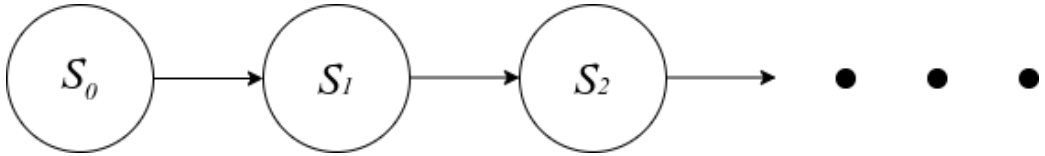
Topología de transición de la cadena de Markov para el ejemplo "weather" con dos estados posibles {sun, rain}



Rutas de estados de la cadena de Markov para el ejemplo "weather"

Archivo "markov_chain.pblg"

Cadenas de Markov



Cadena de Markov como una red Bayesiana dinámica
($S_0, S_1, S_2 \in \{\text{sun, rain}\}$ son variables aleatorias "de estado")

Procesos de decisión de Markov

- Un proceso de decisión de Markov (*MDP*, por sus siglas en inglés) es una extensión a las cadenas de Markov para la toma secuencial de decisiones y que involucra: i) un agente que ejecuta acciones, ii) un ambiente de operación[5,6] y iii) retroalimentaciones numéricas llamadas recompensas[7] otorgadas al agente para cada par estado-acción

[5] Se supone que el estado actual es conocido pero hay incertidumbre sobre el estado resultante después de ejecutar una acción

[6] Acto, tarea

[7] Pueden ser positivas o negativas

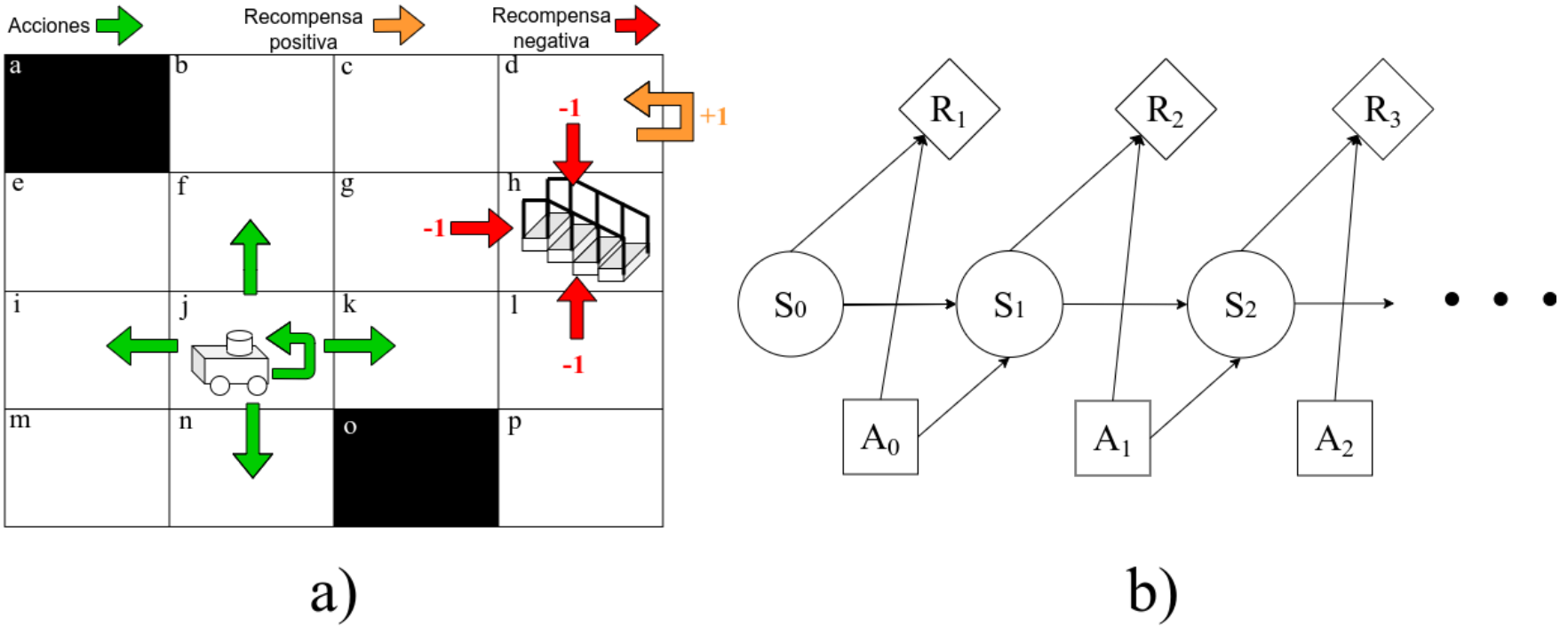
Procesos de decisión de Markov

- Un MDP[8] es una tupla $(\mathcal{S}, \mathcal{A}, P, r)$ donde:
  - $\mathcal{S}$ es un conjunto finito de $n > 0$ estados observables
  - $\mathcal{A}$ es un conjunto finito de $m > 0$ acciones
  - $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ es la probabilidad de que el estado siguiente s' sea alcanzado dado que se ejecuta la acción $a \in \mathcal{A}$ en el estado actual $s \in \mathcal{S}$, tal que $\sum_{s' \in \mathcal{S}} P(s'|s, a) = 1$
  - $r : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ es la *recompensa inmediata* que el agente recibe al ejecutar la acción $a \in \mathcal{A}$ en el estado $s \in \mathcal{S}$ y donde $\mathcal{R} \subset \mathbb{R}$ es un conjunto finito no vacío de recompensas disponibles

Al igual que las cadenas de Markov, un MDP es un proceso estacionario que sigue la propiedad Markoviana

[8] Aquí se consideran sólo MDPs de horizonte infinito ($T = \infty$) en donde las decisiones no dependen del tiempo en que se toma la decisión y no hay preferencia sobre el estado inicial ó un estado-objetivo explícito

Procesos de decisión de Markov



Aplicación de un MDP: a) agente robótico en su ambiente de operación con 4 acciones y recompensas, y b) MDP como una red Bayesiana dinámica que modela posibles secuencias de observaciones de estado, acciones y recompensas:

$$S_0 = s_0, A_0 = a_0, R_1 = r_1, S_1 = s_1, A_1 = a_1, R_2 = r_2, S_2 = s_2, A_2 = a_2, R_3 = r_3, \ldots$$

Procesos de decisión de Markov

- El objetivo de un agente es seleccionar acciones que maximicen el retorno acumulado a largo plazo ó a futuro con descuento[9]:

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

y donde $G_{t+1} = R_{t+2} + \gamma R_{t+3} + \cdots$ es el retorno acumulado descontado obtenido a partir del tiempo $t+2$

[9] El factor $\gamma \in [0, 1)$ es un factor de descuento útil para que la suma infinita de recompensas (presupuesta teóricamente por el horizonte infinito) no sume a infinito. Otra forma de entender a γ es que pondera más al efecto de las recompensas recientes que a las recompensas futuras

Procesos de decisión de Markov - Política determinista

- Una política determinista $\pi : \mathcal{S} \rightarrow \mathcal{A}$, es una función (ó un conjunto de reglas de decisión), que asocia una acción $a \in \mathcal{A}$ con cada estado $s \in \mathcal{S}$ (ie, $\pi(s) = a$)

Procesos de decisión de Markov - Evaluación de la política

- La política π puede evaluarse por la "conveniencia" que produce a cada estado, llamado retorno acumulado descontado *esperado* en el largo plazo siguiendo π a partir de s:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\left[G_t | S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} | S_t = s\right] + \gamma \mathbb{E}_\pi\left[G_{t+1} | S_t = s\right] \end{aligned}$$

  para todo estado $s \in \mathcal{S}$

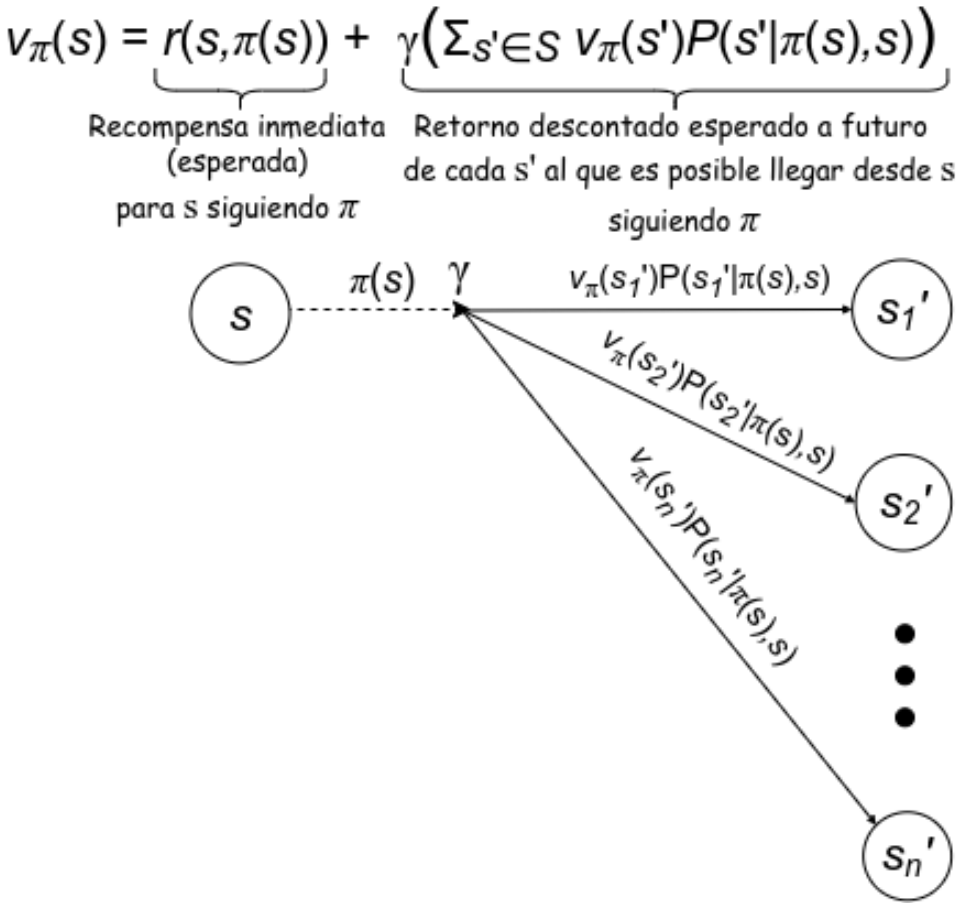
Procesos de decisión de Markov - Evaluación de la política

- La función $v_\pi(s)$ se puede calcular por medio de la ecuación de Bellman:

$$v_\pi(s) = r(s, \pi(s)) + \gamma \left(\sum_{s' \in \mathcal{S}} v_\pi(s') P(s'|s, \pi(s)) \right),$$

para todo estado $s \in \mathcal{S}$

Procesos de decisión de Markov - Evaluación de la política



Cálculo de $v_\pi(s)$ para cada $s \in \mathcal{S}$ mediante la ecuación de Bellman

Procesos de decisión de Markov - Solución

- La solución del MDP es una política óptima $\pi_* \doteq \text{argmax}_{\pi \in \mathcal{A}^{|\mathcal{S}|}} v_\pi(s)$ para todo $s \in \mathcal{S}$, donde $\mathcal{A}^{|\mathcal{S}|}$ es el conjunto de todas las políticas posibles[10], y que genera la función de retorno esperado acumulado descontado máximo[11] $v_* = \max_{\pi \in \mathcal{A}^{|\mathcal{S}|}} v_\pi$

[10] Cuyo tamaño es $|\mathcal{A}|^{|\mathcal{S}|}$, eg, si $|\mathcal{A}|= 2$ y $|\mathcal{S}|= 3$, $2^3 = 8$, pero si $|\mathcal{A}|= 4$ y $|\mathcal{S}|= 100$, $4^{100} = 1.60693804410^{60}$

[11] Puede haber más de una política óptima π_* pero hay una única función de retorno máximo v_*

Procesos de decisión de Markov - Solución

- Sin embargo, en vez de buscar directamente a π_*, es usual aproximar iterativamente primero a la función v_* por medio de la ecuación de actualización de Bellman[12]:

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \left(\sum_{s' \in \mathcal{S}} v_k(s') P(s'|s,a) \right) \right],$$

para cada estado $s \in \mathcal{S}$, a partir de alguna función inicial v_0 arbitraria, hasta que se cumple algún criterio de paro (eg, cuando $\max_{s \in \mathcal{S}} |v_{k+1}(s) - v_k(s)| < \epsilon$, donde ϵ es un umbral suficientemente pequeño)

[12] Puede mostrarse que v_* es único y que la ecuación de actualización de Bellman $v_{k+1}(s)$ mejora v_k hasta converger a $v_*(s)$. Ver: https://towardsdatascience.com/why-does-the-optimal-policy-exist-29f30fd51f8c

Procesos de decisión de Markov

- La obtención de π_* (ó una política útil $\widetilde{\pi} \approx \pi_*$, si la última $v_k \approx v_*$), requiere recuperación de la acción $a \in \mathcal{A}$ que maximiza $v_k(s)$ para cada estado $s \in \mathcal{S}$:

$$\pi_*(s) = \underset{a \in \mathcal{A}}{\text{argmax}} \left[r(s,a) + \gamma \left(\sum_{s' \in \mathcal{S}} v_*(s') P(s'|s,a) \right) \right]$$

  La aproximación iterativa de v_* y la recuperación de $\pi_*(s)$ como paso final se denominan algoritmo de iteración de valor

Procesos de decisión de Markov - Iteración de valor

Algoritmo 1 Iteración de valor para estimar $\pi \approx \pi_*$

Require: Parámetro del algoritmo: un umbral pequeño $\theta > 0$ que determina la exactitud de la estimación.

Inicializar $v(s)$, $\forall s \in \mathcal{S}$ arbitrariamente

repeat

$\Delta \leftarrow 0$

for $s \in \mathcal{S}$ **do**

$v_{temp} \leftarrow v(s)$

$$v(s) \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \left(\sum_{s' \in \mathcal{S}} v_k(s') P(s'|s,a) \right) \right]$$

$\Delta \leftarrow \max(\Delta, |v_{temp} - v(s)|)$

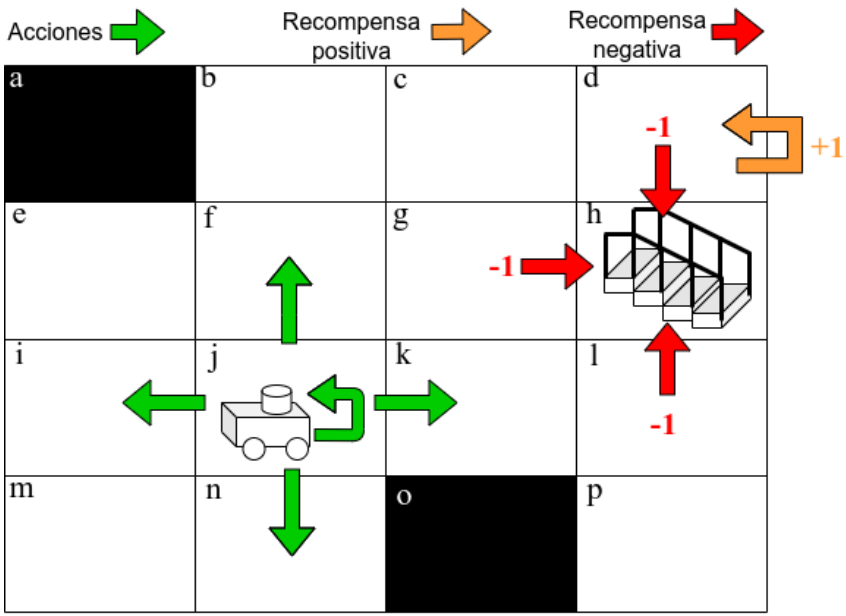
end for

until $\Delta < \theta$

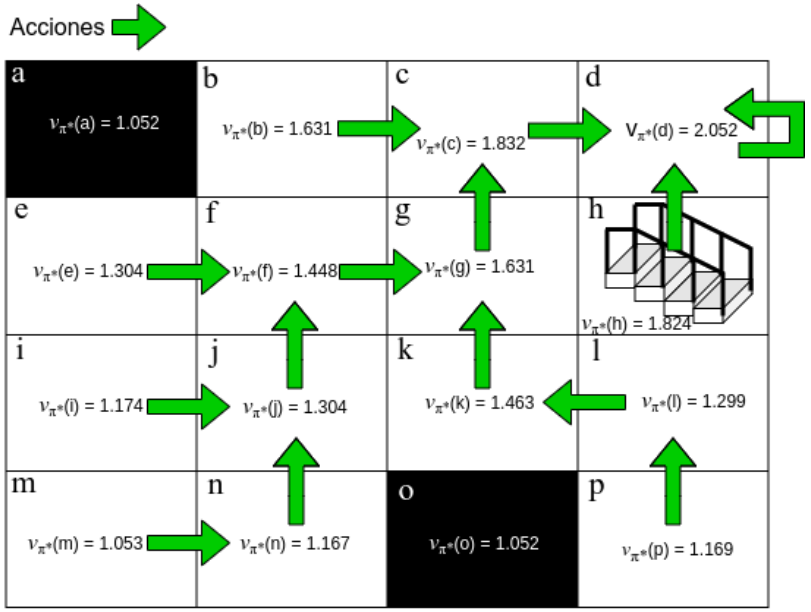
Generar la política $\pi \approx \pi_*$ de tal manera que:

$$\pi(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left[r(s,a) + \gamma \left(\sum_{s' \in \mathcal{S}} v_*(s') P(s'|s,a) \right) \right]$$

Procesos de decisión de Markov



a)



b)

Ejemplo de una política obtenida mediante un MDP: a) agente robótico, y b) política de acción y función $v_{\pi^*}(s)$ para cada estado del sistema: i) a medida que los estados se alejan de d (único con recompensa positiva) su valor decrece, y ii) la acción $\leftarrow$ se prefiere a sobre $\uparrow$, aunque $v_{\pi^*}(\mathrm{h}) > v_{\pi^*}(\mathrm{k})$, ya que $r(\mathrm{l}, \uparrow) = -1$ reduce la "aportación" de esa acción al valor de l)

MDPs factorizados

- Si se consideran conjuntos de k variables de estado[13] discretas $\mathbf{X} = \{X_1, ..., X_k\}$ de pre-acción y $X' = \{X'_1, ..., X'_k\}$ de post-acción con *asignaciones conjuntas*[14] $\mathbf{x} = (x_1, ..., x_k)$ y $\mathbf{x}' = (x'_1, ..., x'_k)$, tal que cada posible asignación define a un estado $s, s' \in \mathcal{S}$, respectivamente, entonces la función de transición se puede factorizar como sigue:

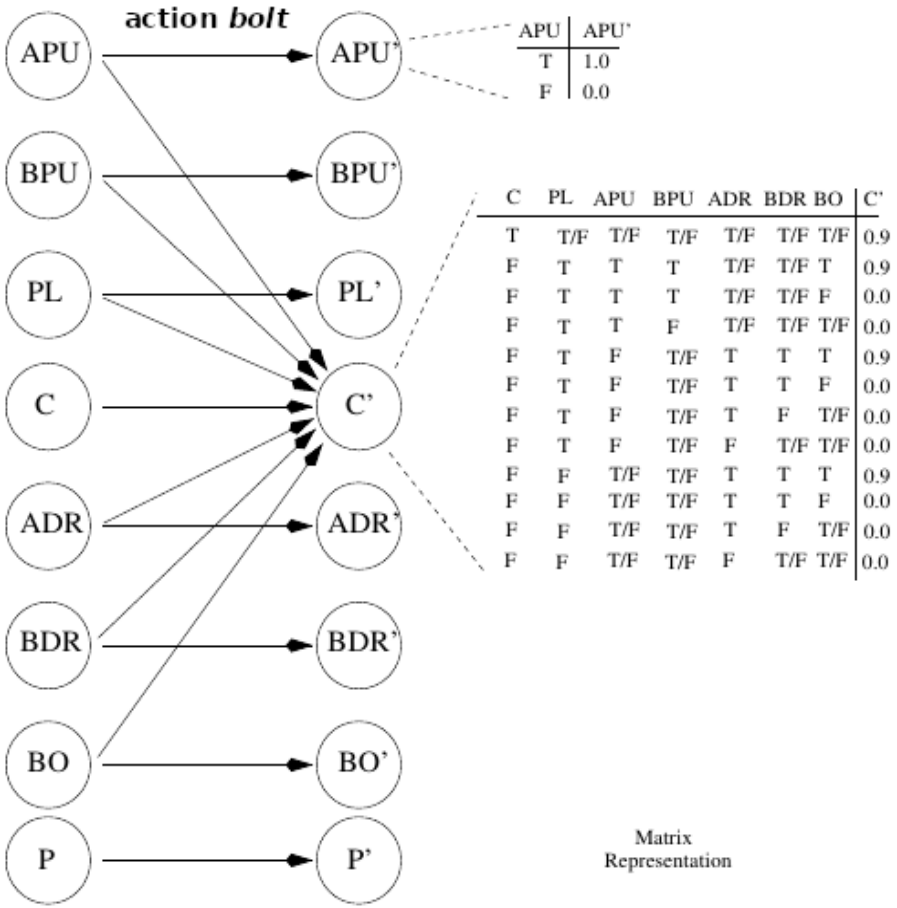
$$P(s'|s,a) = P(\mathbf{x}'|\mathbf{x},a) = \prod_{i=1}^{k} P(X'_i = x'_i|\mathbf{x}_{\mathrm{Pa}(X'_i)}, a)$$

donde $\mathbf{x}_{\mathrm{Pa}(X'_i)}$ es una asignación conjunta de $\mathrm{Pa}(X'_i) \subseteq \mathbf{X}$ (ie, la probabilidad del valor de cada $X'_i \in \mathbf{X}'$ puede depender de sólo algunas de las variables de estado en $\mathbf{X}$)

[13] Específicamente, variables aleatorias "de estado"

[14] De tal manera que $X_i = x_i$ y $X'_i = x'_i$ para todo i

MDPs factorizados



La ejecución de una acción $a \in \mathcal{A}$ modifica sólo a algunas de las VA (en la imagen, $C^{'}$ depende de 7 VAs del tiempo actual y de la acción, y el resto de VAs X' sólo dependen de su valor previo). Imagen modificada sin permiso de: Jesse Hoey, Robert St-Aubin, Alan J Hu, Craig Boutilier, SPUDD: Stochastic Planning Using Decision Diagrams, Proceedings of Uncertainty in Artificial Intelligence. 1999.

MDP-ProbLog

- MDP-ProbLog V0.3.0 es una especialización de ProbLog para modelar MDPs factorizados de horizonte infinito con descuento que utiliza iteración de valor:
  - Disponible en: https://github.com/thiagopbueno/mdp-problog
  - Instalación: pip3 install mdpproblog
  - Ejecución: 'mdp-problog solve -m domain1.pl network1.pl'
- El objetivo es solucionar MDPs involucrando relaciones complejas entre sus elementos usando una descripción comprensible

MDP-ProbLog

- Un programa de MDP-ProbLog es una tupla $L_{MDP} = (\mathcal{A}_t, \mathcal{R}, \mathcal{F}_a)$, donde:
  1) $\mathcal{A}_t$ es un conjunto finito no vacío de átomos divididos en:
     - Un conjunto finito $\mathcal{SF}$ $n > 0$ predicados de estado fluente (VA de estado)
     - Un conjunto finito $\mathcal{A}$ no vacío de $m > 0$ predicados de acciones
     - Un conjunto finito $\mathcal{U}$ de predicados utilidades
  2) Un conjunto finito $\mathcal{R}$ no vacío de reglas divididas en:
     - Un conjunto finito $\mathcal{T}_r$ no vacío de reglas de transición factorizada
     - Un conjunto finito $\mathcal{R}_r$ de reglas de recompensa
  3) Un conjunto finito $\mathcal{F}_a$ hechos probabilísticos auxiliares

MDP-ProbLog - Átomos especiales

- Variable de estado Booleana $\{1, 0\}$:

```
state_fluent(var1). % var1=1 ó var1=0
```

- Acción:

```
action(accion1).
```

- Utilidad:

```
utility(var1(0), valor1). % Variable de estado pre-acción
utility(accion1, valor2). % Para una acción
utility(aux1, valor3). % Para un átomo auxiliar
```

donde valor1, valor2, valor3 $\in \mathbb{R}$

MDP-ProbLog - Átomos auxiliares

- Átomos deterministas ó probabilistas:

```
% Átomo y regla "deterministas"
aux1. % Equivale a 1.0::aux1.
aux2 :- aux1. % Equivale a 1.0::aux3 si aux1.
% Átomo y regla probabilistas
0.45::aux3.
% P(aux1|aux2)=0.1*0.45=0.045
0.1::aux4 :- aux3.
```

MDP-ProbLog - Definición intensional

- Definición intensional donde la variable X es sustituida por dos o más constantes y sirve para variables, acciones y utilidades:

```
aux1(obj1).
aux1(obj2).
% Genera dos variables de estado: var1(obj1), var1(obj2)
state_fluent(var1(X)) :-  aux1(X).
% Genera dos acciones: accion1(obj1) y accion1(obj2)
accion1(X) :-  aux1(X).
% Genera utilidades negativas de -5 para aux2(obj1)
% y aux2(obj2)
utility(aux2(X), -5) :-  aux1(X).
```

MDP-ProbLog - Definición intensional

- Definición intensional (*Cont.*):

```
% Definición intensional donde aux1(obj1) y
% aux1(obj2) suceden con probabilidad 0.1
aux2(obj1).
aux2(obj2).
0.1::aux1(X) :- aux2(X).
```

MDP-ProbLog - Partición temporal

- Partición temporal de átomos (fluentes, acciones, auxiliares):

```
% El argumento '(0)' indica que la variable
% de estado es "pre-acción"
var1(0).
% El argumento '(1)' que la variable
% de estado es "post-acción"
var1(1).
% Es posible particionar cualquier átomo
% aunque para las acciones posiblemente
% no tendría sentido
```

MDP-ProbLog - Modelo de transición factorizado

- Una regla de transición factorizada describe a los factores del tipo $P(X_i' = 1|\mathbf{x}_{\mathrm{Pa}(X_i')}, a)$:

```
% P(var1'|var1, ¬var2, accion1) = 0.95
% ie, la probabilidad de que var1' sea verdad,
% dados var1, ¬var2 y accion1
0.95::var1(1) :- var1(0), not(var2(0)), accion1.

% P(var1'|accion1) = 0.01 (sólo una acción, no importa
% el valor del resto de las variables)
0.01::var1(1) :- accion2.
```

MDP-ProbLog - Modelo de recompensa

- El modelo de recompensa (opcional) indica cómo asignar utilidades:

```
% La utilidad de 3 a aux1 que depende de otras
% variables, acciones y átomos auxiliares
utility(aux1, 3).
0.95::aux1 :- not(var1), action1, not(aux2).
0.01::aux2 :- var2, action2.

% Utilidad no constante
lista([1,2]).
lista([1]).
utility(aux1(X), C) :- lista(X), length(X,L),
                       C is -0.5*L.
```

MDP-ProbLog - Asociaciones cíclicas

- Asociaciones cíclicas (ó recursivas):

```
aux2(obj1,obj2).
aux2(obj1,obj3).
0.2::aux1(obj2).
0.1::aux1(obj3).
% X cumple la propiedad aux1 con probabilidad 0.95
% si se cumple la relación aux2(X,Y) y aux1(Y)
0.95::aux1(X) :- aux2(X,Y), aux1(Y).
% eg, aux1(X) equivale a fuma(X)
% y aux2(X,Y) equivale a amigo_de(X,Y)
```

Ejemplos

- Archivos:

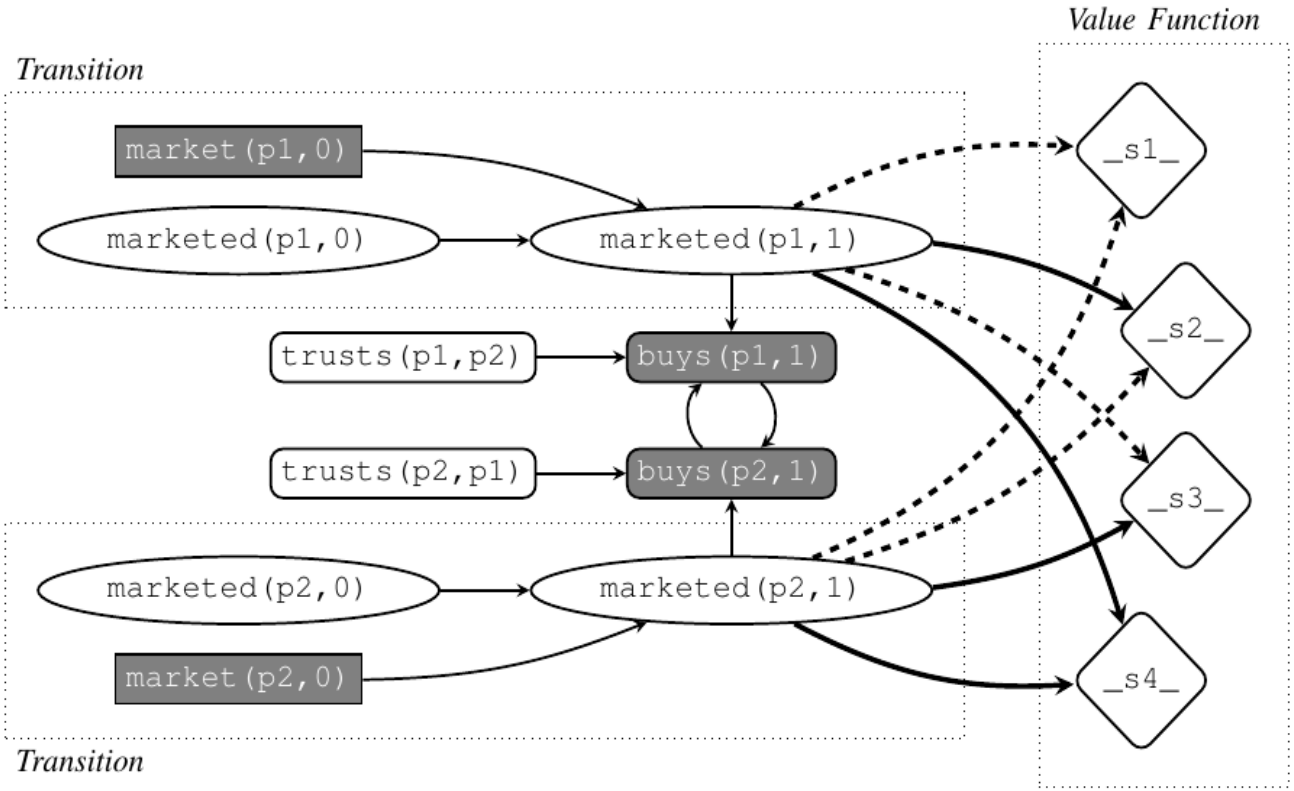
"mdp-problog solve -m simplest_example.pl dummy.pl"

Ejemplo

- Considere el ejemplo "viral marketing": una compañía desea obtener una política de acción sobre cómo marketear a un grupo de personas en una red social tomando en cuenta la utilidad y costo del marketeo, las relaciones de confianza entre estas personas y si compran por marketeo y/o por confianza en un comprador previo

Archivos (dentro de la carpeta "Viral_marketing"):
"mdp-problog solve -m domain1.pl network1.pl"

MDP-ProbLog



Grafo de dependencia aterrizado de "viral marketing" para dos personas. Los rombos s_1, s_2, s_3, s_4 son la función de valor por estado (línea sólida si cada persona ha sido marqueteada y línea punteada si no) (Reproducido sin permiso de: Thiago P. Bueno, Denis D. Mauá, Leliane N. de Barros Fabio G. Cozman. Markov Decision Processes Specified by Probabilistic Logic Programming: Representation and Solution. 5th Brazilian Conference on Intelligent Systems. Págs. 337-342. 2016)

Ejemplos

- Archivos:

"mdp-problog solve -m mobile_robot.pl dummy.pl"

MDP-ProbLog - Recompensa inmediata

- Para cada par estado-acción (s, a) y dados los pares átomo-utilidad $(a_t, u) \in \mathcal{U}$ donde $a_t \in \mathcal{A}_t$ y $u \in \mathbb{R}$, la recompensa inmediata se calcula en MDP-ProbLog de la siguiente manera:

$$r(s,a) = \sum_{(X,u)\in\mathcal{U}:X\in\mathcal{SF}} uP(X|L_p;s,a) + \sum_{(\text{acc},u)\in\mathcal{U}:\text{acc}\in\mathcal{A}} uP(\text{acc}|L_p;s,a) + \sum_{(\text{aux},u)\in\mathcal{U}:\text{aux}\in\mathcal{F}_a} uP(\text{aux}|L_p;s,a)$$

donde u es la utilidad asociada a cada átomo, $P(X|L_p;s,a)$ y $P(\text{aux}|L_p;s,a)$ se obtienen de ProbLog, $P(\text{acc}|L_p;s,a) = 1$ *ssi* $\text{acc} = a$ y L_p es el programa en ProbLog

MDP-ProbLog - Valor futuro esperado descontado

- El retorno esperado descontado a partir del tiempo $t+2$ para cada par estado-acción (s, a) se obtiene como:

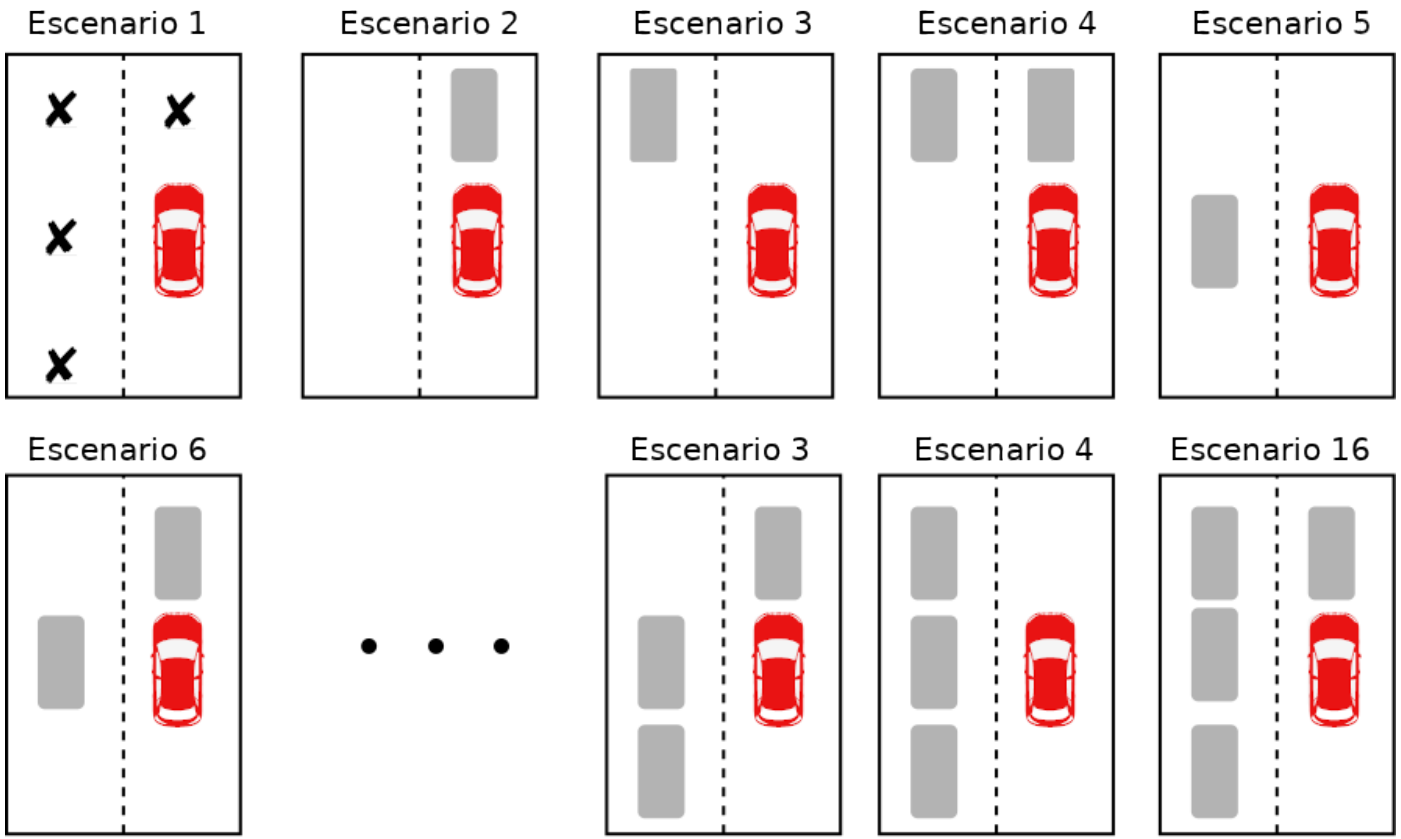
$$\begin{aligned}\gamma\mathbb{E}[v_k(S_{t+1})|S_t = s, A_t = a] &= \gamma\left(\sum_{s'\in\mathcal{S}} v_k(s')P(s'|L_p; s, a)\right)\\ &= \sum_{\forall \mathbf{x}'} \gamma v_k(\mathbf{x}')P(\mathbf{x}'|L_p; \mathbf{x}, a)\\ &= \sum_{\forall \mathbf{x}'} \gamma v_k(\mathbf{x}') \prod_{i=1}^{k} P(X_i' = x_i'|L_p; \mathbf{x}, a)\\ &= \sum_{\forall \mathbf{x}'} \mathcal{U}_k(\mathbf{x}') \prod_{i=1}^{k} P(X_i' = x_i'|L_p; \mathbf{x}, a)\end{aligned}$$

donde $\gamma v_k(\mathbf{x}') = \mathcal{U}_k(\mathbf{x}')$ (el valor descontado del estado siguiente s' es la utilidad de cada estado $\mathbf{x}'$ calculada internamente por MDP-ProbLog)

Actividad

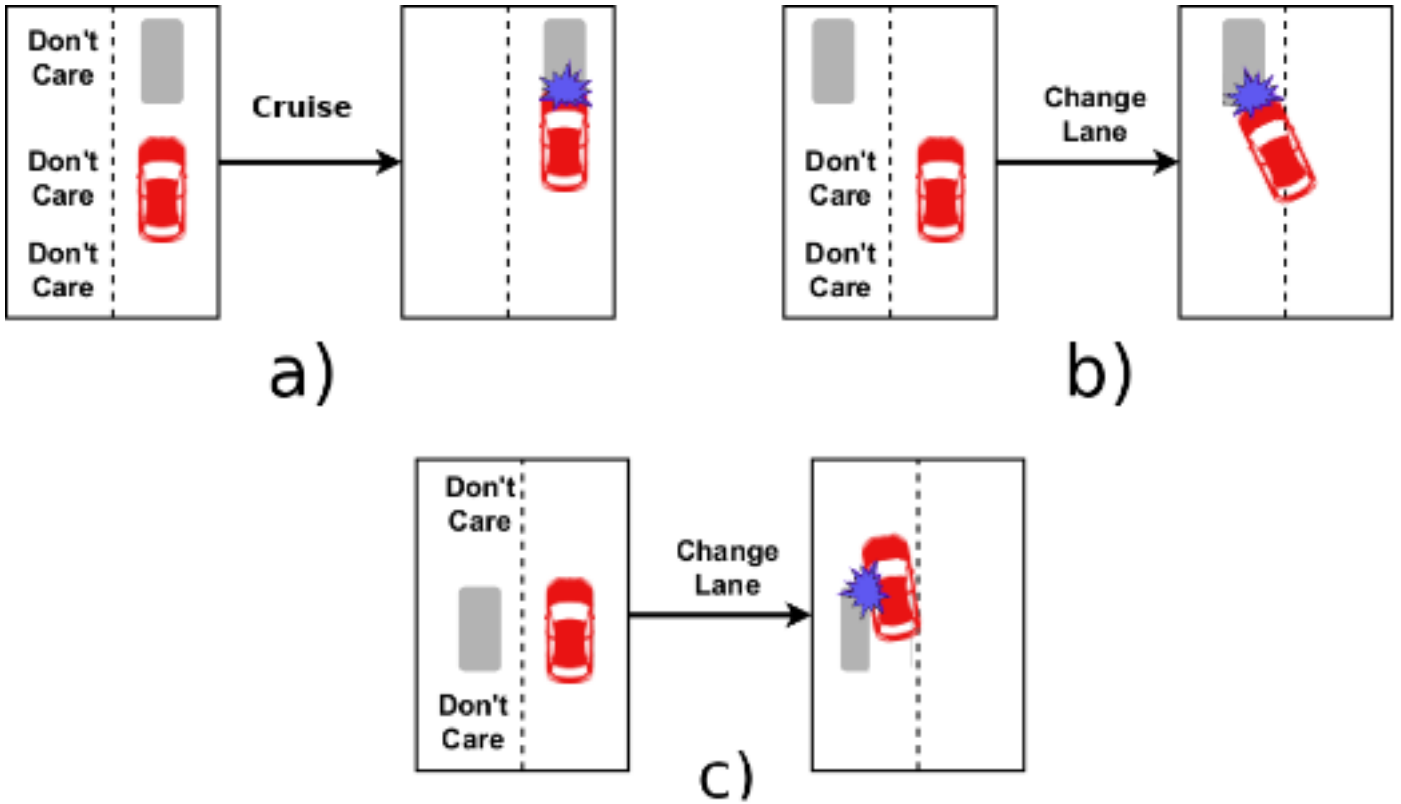
- De manera individual o por equipos (hasta 5 personas) diseñar un MDP en MDP-ProbLog para generar una política de acciones reactivas para un coche autónomo:
  - Hasta 4 coches vecinos alrededor de un coche autónomo en dos carriles de un solo sentido.
  - Se proponen 3 acciones básicas: movimiento de crucero, mantener distancia y cambiar de carril
  - Se supondrá que la percepción del coche autónomo incluye la identificación de la presencia/ausencia de los otros vehículos; el estado del sistema (del coche autónomo y del ambiente) es conocido en cada momento
  - Se supone también que el coche inicia movimiento en línea recta en el carril derecho detrás de los vecinos y a velocidad mayor
  - La prueba termina cuando el coche autónomo tome la primera decisión con la que sea evidente que se evita un accidente

Estados del sistema



Posición supuesta para los coches vecinos (✗ en Escenario 1 donde no hay coches vecinos) en el entorno del coche autónomo: i) adelante, ii) adelante a la izquierda, iii) izquierda, iv) atrás a la izquierda (la ocupación se marca con rectángulos en el resto de los escenarios)

Estados del sistema



Se pueden considerar diferentes tipos de colisión, eg: a) y b) choque por alcance, y c) choque lateral

Sugerencias para generar la política

- Para obtener la política con MDP-ProbLog se sugieren los siguientes pasos:
  1) Identificar el agente, el ambiente, los estados y acciones
  2) Definir las variables de estado (binarias)
  3) Considerar átomos auxiliares para el modelo de recompensa (pueden servir para reducir el número de variables de estado)
  4) Definir utilidades para variables de estado[15], acciones y átomos auxiliares
  5) Identificar las variables de estado post-acción que serán modificadas y no modificadas por cada acción y sus variables-padre pre-acción
  6) Proponer las probabilidades de cada variable aleatoria post-acción en forma de regla
  7) Ejecutar mdp-problog, generar la política y depurar

[15] Se sugiere asignar utilidad sólo a variables de estado pre-acción (no para variables post-acción) dado que se busca construir $r(s, a)$

Código Webots

- Repositorio (seguir instrucciones de instalación):
  https://github.com/hector-aviles/CodigosEIR22-23
- Ejecutar mundo:
  $ roslaunch eir2023 eir.launch world:=01
  (número de mundo 00-15, Correr: 00, 01, 03)
- Ejecutar archivo de política:
  $ rosrun eir2023 policy.py

Resumen

- Se revisaron conceptos básicos de Prolog, ProbLog y ejemplos MDP-ProbLog como herramientas útiles en IA y en robótica
- Los MDPs permiten modelar la toma de decisiones de un agente cuando hay incertidumbre sobre el estado que resultará en cada par estado-acción y hay recompensas para el tomador de decisiones
- En particular, MDP-ProbLog sigue lógico-probabilista para describir MDPs con una descripción ¿clara y flexible?
- Se construyó de una política de decisiones reactivas para un coche autónomo usando ROS y Webots
- Hay mucho por explorar (eg, otros modelos gráfico-probabilistas para los coches autónomos, aprendizaje, VA continuas, nuevas aplicaciones) pero los sitios de ProbLog y MDP-ProbLog son un buen lugar para continuar

¡Gracias!

¿Preguntas?

Comentarios y preguntas a:
havilesa@upv.edu.mx