Busca em Ambientes com Incerteza

Problemas de Decisões Seqüenciais

- Problemas de decisões seqüenciais:
  - Envolvem utilidades e incertezas

- **Podem ser vistos como uma generalização do problema de busca e planejamento**

Funções de Utilidade

- **Funções de Utilidade associam um valor a um estado**
  - Indica o “desejo” por estar nesse estado
  - U(S) = utilidade estado S de acordo com o agente
  - Ex.: s_1 = {rico, famoso}, s_2 = {pobre, famoso}

  $U(s_1) = 10$

  $U(s_2) = 5$

Funções de Utilidade

- **$Result_i(A)$: Todos os possíveis estados de saída de uma ação não-determinista A**
- **Para cada saída possível é associada uma probabilidade:**
  - $P(Result_i(A) \mid Do(A), E)$
  - Onde, **E** resume a evidência que o agente possuí do mundo

    Do(A) indica que a ação **A** foi executada no estado atual

- **Utilidade esperada de uma ação A dado a evidência do mundo E:**

$$EU(A|E) = \Sigma_i \, P(Result_i(A)|Do(A),E) \, U(Result_i(A))$$

- **Principio da Maximização da Utilidade: agente racional deve escolher ação que maximiza sua utilidade esperada !!!**

Exemplo: Cálculo da Utilidade Esperada

- Robô deve transportar uma caixa
  E = caixa é de metal

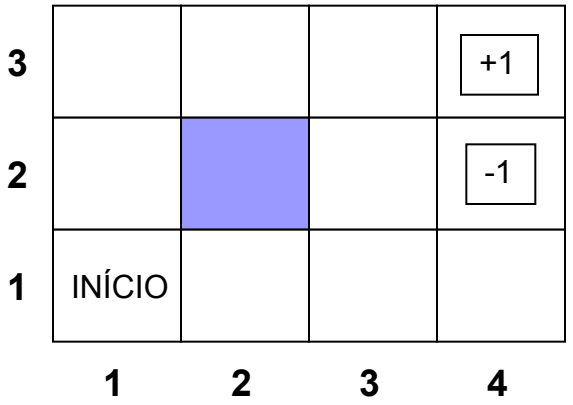
  a_1 = Chutar: s_1, caixa no destino 20% $U(s_1) = 10$
  s_2, caixa no meio do caminho 30% $U(s_2) = 5$
  s_3, caixa longe destino 50% $U(s_3) = 0$

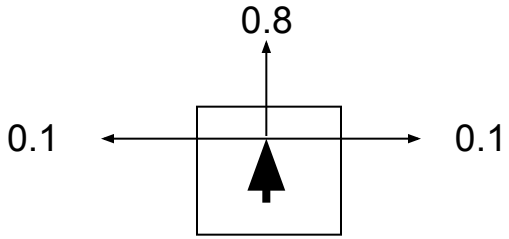
  a_2 = Carregar: s_1, caixa no destino 80% $U(s_1) = 10$
  s_2, caixa na origem 20% $U(s_2) = 0$

  $EU(a_1) = 0{,}20 \times 10 + 0{,}30 \times 5 + 0{,}50 \times 0 =$ **3,5**
  $EU(a_2) = 0{,}80 \times 10 + 0{,}20 \times 0 =$ **8**

Exemplo: Ambiente 4x3

- **Interação termina quando o agente alcança um dos estados finais (+1 ou -1)**
- **Ações disponíveis:**
  - Up, Down, Left e Right
- **Ambiente totalmente observável**
  - Agente sabe onde está!
- **Ações não confiáveis**
  - Locomoção estocástica
- **Se o agente bater em uma parede permanecerá no mesmo quadrado**
- **Em cada estado *s, o* agente recebe uma Recompensa R(s):**
  - R(s) = -0.04 para todos estados não terminais
  - Dois estados finais R(s) = +1 ou R(s) = -1
- **Por enquanto, utilidade pode ser dada pela soma das recompensas recebidas!**





Funções de Utilidade para Problemas Seqüenciais

- **Como definir funções de utilidade para problemas seqüenciais?**

$$U([s_0, s_1, \ldots, s_n])$$

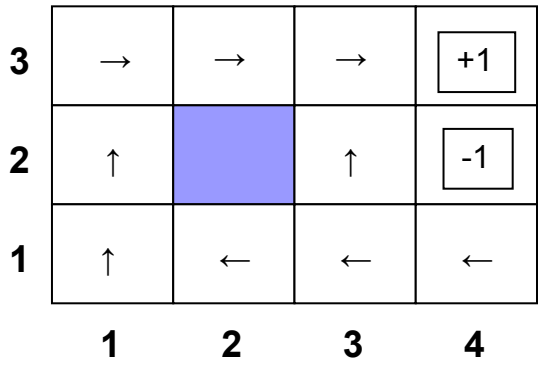
- **Resposta: atribuir recompensas aos estados visitados na seqüência**
  - Recompensas aditivas
  - Recompensas descontadas

Recompensas

- **Recompensas Aditivas:**
  - $U_h([s_0, s_1, \ldots, s_n]) = R(s_0) + R(s_1) + R(s_2) + \ldots$
- **Recompensas Descontadas:**
  - $U_h([s_0, s_1, \ldots, s_n]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$
  - Onde γ é chamado *fator de desconto* e tem valor entre 0 e 1
- **Fator de desconto:**
  - Descreve a preferência de um agente com relação a recompensas atuais sobre recompensas futuras
    - γ próximo a 0 $\Rightarrow$ recompensas no futuro distante são irrelevantes
    - $\gamma = 1 \Rightarrow$ recompensa aditiva

Como são as soluções desse problema?

- **Uma solução deve especificar o que o agente deve fazer em qualquer estados em que possa chegar**
- **Seqüência fixa de ações não o resolvem:**
  - Ações não confiáveis não geram estados deterministicamente
- **Solução: construir uma Política (Policy):**
  - **π (s)** = ação recomendada para estado s
  - Assim, o agente sabe como atuar em qualquer estado
- **Utilidade esperada de uma política é dada pelas seqüências de ações que ela pode gerar**
- **Política Ótima π^*:**
  - Política que produz a mais alta utilidade esperada



Processo de Decisão de Markov (PDM)

Problema de decisão seqüencial em um ambiente totalmente observável com modelo de transição de Markov e recompensas aditivas

- **Definido pelos seguintes componentes:**
  - Estado Inicial: s_0
  - Modelo de Transição: T(s,a,s')
    - **Probabilidade de chegar a *s'* como resultado da execução da ação *a* no estado *s***
  - Função de Recompensa: R(s)
    - **Do estado *s* para o agente**

- Hipótese de transição Markoviana:
  - Próximo estado depende apenas da ação atual e do estado atual, não dependendo de estados passados

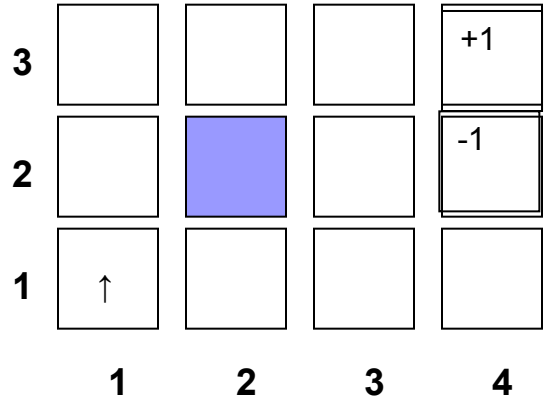
Algoritmo *Value Iteration*

- **Algoritmo para calcular políticas ótimas**
- **Idéia: calcular a utilidade de cada estado e utilizá-las para escolher uma ação ótima**
- **Utilidade de cada estado definida em termos da utilidade das seqüências de estados que podem se seguir a partir dele**
  - R(s): recompensa a “curto prazo” por se estar em s
  - U(s): recompensa total a “longo prazo” a partir de s

Algoritmo *Value Iteration*

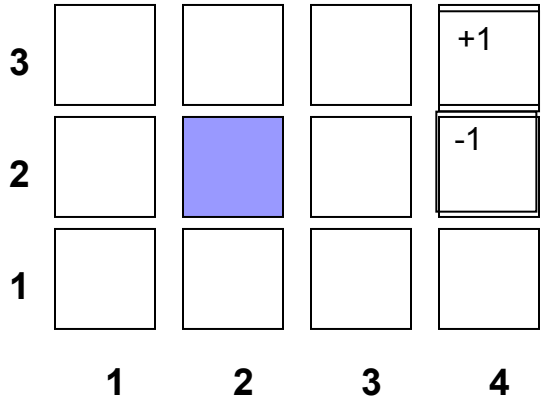
- **Utilidade de um estado é dada pela recompensa imediata para aquele estado mais a utilidade esperada descontada do próximo estado, assumindo que o agente escolhe a ação ótima**

  - U(1,1) = -0.04
    + γ [0.8 U(1,2) +
    0.1 U(2,1) +
    0.1 U(1,1)]



Algoritmo *Value Iteration*

- **Utilidade de um estado é dado pela equação de Bellman:**
  - $U(s) = R(s) + \gamma \max_a \Sigma_{s'} T(s,a,s') U(s')$



- **Exemplo:**
  - U(1,1) = -0.04 + γ max {
    0.8 U(1,2) + 0.1 U(2,1) + 0.1 U(1,1), ***(Up)***
    0.9 U(1,1) + 0,1 U(1,2), ***(Left)***
    0.9 U(1,1) + 0.1 U(2,1), ***(Down)***
    0.8 U(2,1) + 0.1 U(1,2) + 0.1 U(1,1) } ***(Right)***

Algoritmo *Value Iteration*

INICIALIZAÇÃO

> rw = matrix(-0.04,1,11)
> rw[1,10] = -1
> rw[1,11] = 1
>
> value = matrix(0,1,11) ← Utilidades incializadas com valor 0

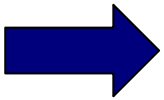
Algoritmo *Value Iteration*

> value = update_value(T_up, T_down, T_right, T_left, rw, value)

"INPUT UTILITIES"

	[,1]	[,2]	[,3]	[,4]
[1,]	0	0	0	0
[2,]	0	0	0	0
[3,]	0	0	0	0

1a iteração



"OUTPUT UTILITIES"

	[,1]	[,2]	[,3]	[,4]
[1,]	-0.04	-0.04	-0.04	1.00
[2,]	-0.04	0.00	-0.04	-1.00
[3,]	-0.04	-0.04	-0.04	-0.04

- $U(s) = R(s) + \gamma \max_a \Sigma_{s'} T(s,a,s') U(s')$

Algoritmo *Value Iteration*

> policy = return_policy(T_up,T_down,T_right,T_left,value)

RG LF RG +1
UP * LF -1
RG LF RG DW

Política resultante da 1a. iteração.
Ainda meio aleatória

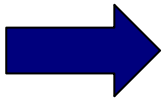
Algoritmo *Value Iteration*

> value = update_value(T_up, T_down, T_right, T_left, rw, value)

"INPUT UTILITIES"

	[,1]	[,2]	[,3]	[,4]
[1,]	-0.04	-0.04	-0.04	1.00
[2,]	-0.04	0.00	-0.04	-1.00
[3,]	-0.04	-0.04	-0.04	-0.04

2a iteração



"OUTPUT UTILITIES"

	[,1]	[,2]	[,3]	[,4]
[1,]	-0.08	-0.08	0.752	1.00
[2,]	-0.08	0.00	-0.080	-1.00
[3,]	-0.08	-0.08	-0.080	-0.08

RG RG RG +1
UP * UP -1
RG LF UP DW

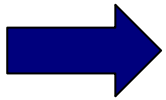
Bom estado (1,3) identificado!

Algoritmo *Value Iteration*

> value = update_value(T_up, T_down, T_right, T_left, rw, value)

"INPUT UTILITIES"

	[,1]	[,2]	[,3]	[,4]
[1,]	-0.08	-0.08	0.752	1.00
[2,]	-0.08	0.00	-0.080	-1.00
[3,]	-0.08	-0.08	-0.080	-0.08



"OUTPUT UTILITIES"

	[,1]	[,2]	[,3]	[,4]
[1,]	-0.12	0.5456	0.8272	1.00
[2,]	-0.12	0.0000	0.4536	-1.00
[3,]	-0.12	-0.1200	-0.1200	-0.12

RG RG RG +1
UP * UP -1
UP UP UP DW

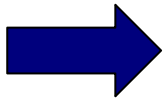
Estados (2,3) e (1,2) são bons porque levam ao estado (1,3) com as ações UP e RG respectivamente

Algoritmo *Value Iteration*

```
> value = update_value(T_up, T_down, T_right, T_left, rw, value)
```

"INPUT UTILITIES"

	[,1]	[,2]	[,3]	[,4]
[1,]	-0.12	0.5456	0.8272	1.00
[2,]	-0.12	0.0000	0.4536	-1.00
[3,]	-0.12	-0.1200	-0.1200	-0.12



"OUTPUT UTILITIES"

	[,1]	[,2]	[,3]	[,4]
[1,]	0.372	0.730	0.888	1.00
[2,]	-0.160	0.000	0.567	-1.00
[3,]	-0.160	-0.160	0.298	-0.16

RG	RG	RG	+1
UP	*	UP	-1
RG	RG	UP	LF

Estados (3,4) deve ser evitado

Algoritmo *Value Iteration*

Depois de várias iterações….

"UTILITIES"

	[,1]	[,2]	[,3]	[,4]
[1,]	0.8115	0.8678	0.9178	+1
[2,]	0.7615		0.6602	-1
[3,]	0.7053	0.6553	0.6114	0.3879

“POLICY”

RG	RG	RG	+1
UP	*	UP	-1
UP	LF	LF	LF

A política indicar que se deve ir na direção do objetivo com recompensa +1 mas se evitar o estado com recompensa -1