Artificial Intelligence

Dynamic Programming

Marc Toussaint
University of Stuttgart
Winter 2018/19

Motivation:

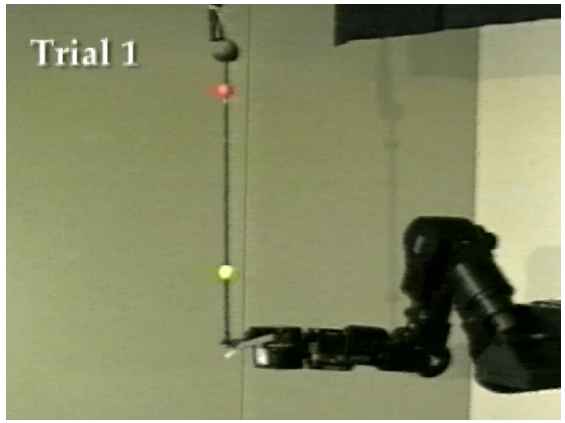
So far we focussed on tree search-like solvers for decision problems. There is a second important family of methods based on dynamic programming approaches, including Value Iteration. The Bellman optimality equation is at the heart of these methods.

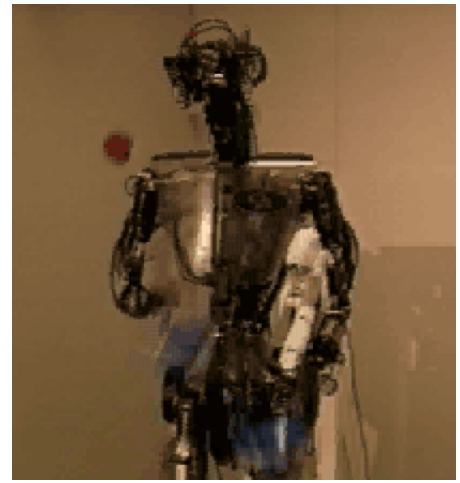
Such dynamic programming methods are important also because standard Reinforcement Learning methods (learning to make decisions when the environment model is initially unknown) are directly derived from them.

Markov Decision Process

MDP & Reinforcement Learning

- MDPs are the basis of Reinforcement Learning, where $P(s'|s,a)$ is not know by the agent

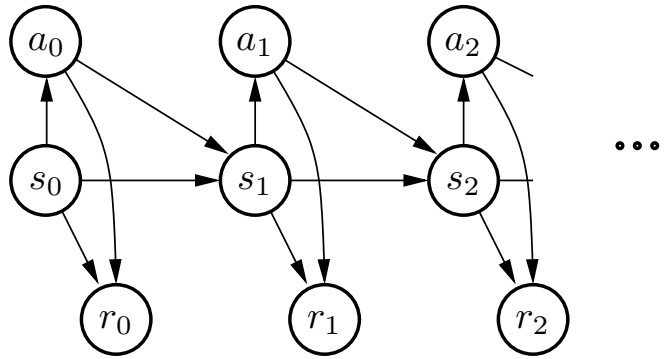




(around 2000, by Schaal, Atkeson, Vijayakumar)

(2007, Andrew Ng et al.)

Markov Decision Process



$$P(s_{0:T+1}, a_{0:T}, r_{0:T}; \pi) = P(s_0) \prod_{t=0}^{T} P(a_t|s_t;\pi)\ P(r_t|s_t,a_t)\ P(s_{t+1}|s_t,a_t)$$

- world's initial state distribution $P(s_0)$
- world's transition probabilities $P(s_{t+1} \mid s_t, a_t)$
- world's reward probabilities $P(r_t \mid s_t, a_t)$
- agent's *policy* $\pi(a_t \mid s_t) = P(a_0|s_0;\pi)$ (or deterministic $a_t = \pi(s_t)$)

- **Stationary MDP:**
  - We assume $P(s' \mid s, a)$ and $P(r|s,a)$ independent of time
  - We also define $R(s,a) := \mathsf{E}\{\}r|s,a = \int r\ P(r|s,a)\ dr$

MDP

- In basic discrete MDPs, the transition probability

$$P(s'|s,a)$$

  is just a table of probabilities

- The *Markov* property refers to how we defined state:
  History and future are conditionally independent given s_t

$$I(s_{t^+}, s_{t^-}|s_t)\ , \quad \forall t^+ > t, t^- < t$$

Dynamic Programming

State value function

- We consider a stationary MDP described by

$$P(s_0)\ , \quad P(s' \,|\, s, a)\ , \quad P(r \,|\, s, a)\ , \quad \pi(a_t \,|\, s_t)$$

- The **value** (*expected* discounted return) of policy π when started in state s:

$$V^\pi(s) = \mathsf{E}_\pi\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \,|\, s_0 = s\}$$

discounting factor $\gamma \in [0, 1]$

State value function

- We consider a stationary MDP described by

$$P(s_0)\ , \quad P(s' \,|\, s, a)\ , \quad P(r \,|\, s, a)\ , \quad \pi(a_t \,|\, s_t)$$

- The **value** (*expected* discounted return) of policy π when started in state s:

$$V^\pi(s) = \mathsf{E}_\pi\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \,|\, s_0 = s\}$$

discounting factor $\gamma \in [0, 1]$

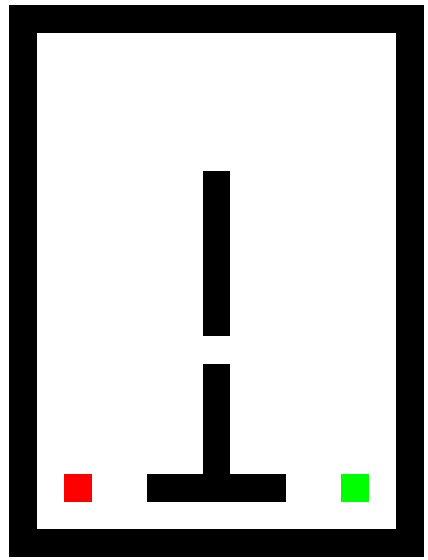
- Definition of **optimality**: A policy π^* is optimal iff

$$\forall s:\ V^{\pi^*}(s) = V^*(s) \qquad \text{where } V^*(s) = \max_\pi V^\pi(s)$$

(simultaneously maximising the value in all states)

(In MDPs there always exists (at least one) optimal deterministic policy.)

An example for a value function...



demo: `test/mdp runVI`

Values provide a gradient towards desirable states

Value function

- The value function V is a central concept in all of RL!

  Many algorithms can directly be derived from properties of the value function.

- In other domains (stochastic optimal control) it is also called *cost-to-go* function (cost $= -$reward)

Recursive property of the value function

$$
\begin{aligned}
V^{\pi}(s) &= \mathsf{E}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s; \pi\} \\
&= \mathsf{E}\{r_0 \mid s_0 = s; \pi\} + \gamma \mathsf{E}\{r_1 + \gamma r_2 + \cdots \mid s_0 = s; \pi\} \\
&= R(s, \pi(s)) + \gamma \textstyle\sum_{s'} P(s' \mid s, \pi(s)) \; \mathsf{E}\{r_1 + \gamma r_2 + \cdots \mid s_1 = s'; \pi\} \\
&= R(s, \pi(s)) + \gamma \textstyle\sum_{s'} P(s' \mid s, \pi(s)) \; V^{\pi}(s')
\end{aligned}
$$

Recursive property of the value function

$$
\begin{aligned}
V^\pi(s) &= \mathsf{E}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s; \pi\} \\
&= \mathsf{E}\{r_0 \mid s_0 = s; \pi\} + \gamma \mathsf{E}\{r_1 + \gamma r_2 + \cdots \mid s_0 = s; \pi\} \\
&= R(s, \pi(s)) + \gamma \textstyle\sum_{s'} P(s' \mid s, \pi(s))\ \mathsf{E}\{r_1 + \gamma r_2 + \cdots \mid s_1 = s'; \pi\} \\
&= R(s, \pi(s)) + \gamma \textstyle\sum_{s'} P(s' \mid s, \pi(s))\ V^\pi(s')
\end{aligned}
$$

- We can write this in vector notation $\boldsymbol{V}^\pi = \boldsymbol{R}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{V}^\pi$ with vectors $\boldsymbol{V}^\pi_s = V^\pi(s)$, $\boldsymbol{R}^\pi_s = R(s, \pi(s))$ and matrix $\boldsymbol{P}^\pi_{ss'} = P(s' \mid s, \pi(s))$

- For stochastic $\pi(a|s)$:
$$V^\pi(s) = \textstyle\sum_a \pi(a|s) R(s, a) + \gamma \sum_{s', a} \pi(a|s) P(s' \mid s, a)\ V^\pi(s')$$

Bellman optimality equation

- Recall the recursive property of the value function

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s' \mid s, \pi(s))\ V^{\pi}(s')$$

- Bellman optimality equation

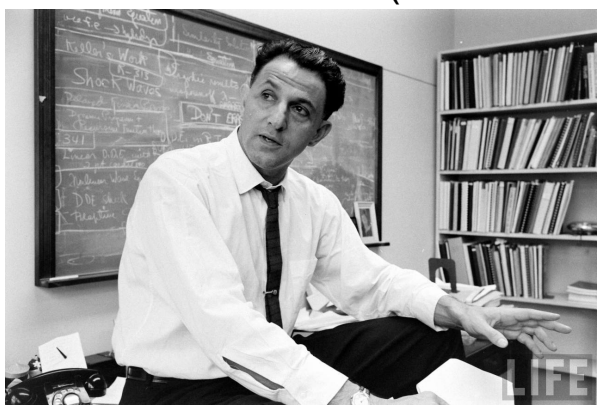
$$V^{*}(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s, a)\ V^{*}(s') \right]$$

with $\pi^{*}(s) = \operatorname{argmax}_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s, a)\ V^{*}(s') \right]$

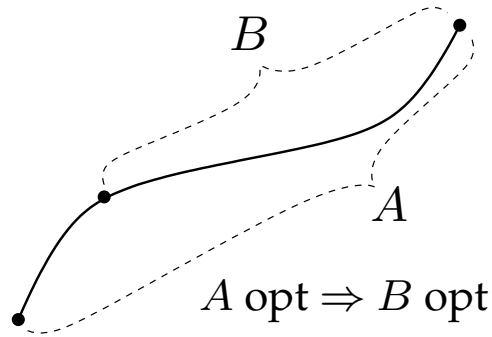
(Sketch of proof: If π would select another action than $\operatorname{argmax}_a[\cdot]$, then π' which $= \pi$ everywhere except $\pi'(s) = \operatorname{argmax}_a[\cdot]$ would be better.)

- This is the **principle of optimality** in the stochastic case

Richard E. Bellman (1920-1984)



Bellman's principle of optimality



$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\, V^*(s') \right]$$

$$\pi^*(s) = \operatorname*{argmax}_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\, V^*(s') \right]$$

Value Iteration

- *How can we use this to compute V^*?*
- Recall the Bellman optimality equation:

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\ V^*(s') \right]$$

- **Value Iteration:** (initialize $V_{k=0}(s) = 0$)

$$\forall s:\ V_{k+1}(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a)\ V_k(s') \right]$$

stopping criterion: $\max_s |V_{k+1}(s) - V_k(s)| \leq \epsilon$

- Note that V^* is a **fixed point** of value iteration!
- Value Iteration converges to the optimal value function V^* (proof below)

`demo: test/mdp runVI`

State-action value function (Q-function)

- *We repeat the last couple of slides for the Q-function...*
- The *state-action value function* (or **Q-function**) is the expected discounted return when starting in state s and taking first action a:

$$\begin{aligned} Q^\pi(s,a) &= \mathsf{E}_\pi\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s, a_0 = a\} \\ &= R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\, Q^\pi(s', \pi(s')) \end{aligned}$$

(Note: $V^\pi(s) = Q^\pi(s, \pi(s))$.)

- Bellman optimality equation for the Q-function

$$Q^*(s,a) = R(s,a) + \gamma \textstyle\sum_{s'} P(s' \mid s,a) \max_{a'} Q^*(s',a')$$

with $\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$

Q-Iteration

- Recall the Bellman equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \,|\, s,a) \; \max_{a'} Q^*(s',a')$$

- **Q-Iteration:** (initialize $Q_{k=0}(s,a) = 0$)

$$\forall_{s,a} : \; Q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \; \max_{a'} Q_k(s',a')$$

stopping criterion: $\max_{s,a} |Q_{k+1}(s,a) - Q_k(s,a)| \leq \epsilon$

- Note that Q^* is a **fixed point** of Q-Iteration!
- Q-Iteration converges to the optimal state-action value function Q^*

Proof of convergence

- Let $\Delta_k = ||Q^* - Q_k||_\infty = \max_{s,a} |Q^*(s,a) - Q_k(s,a)|$

$$
\begin{aligned}
Q_{k+1}(s,a) &= R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q_k(s',a') \\
&\leq R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} \Big[Q^*(s',a') + \Delta_k \Big] \\
&= \Big[R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') \Big] + \gamma \Delta_k \\
&= Q^*(s,a) + \gamma \Delta_k
\end{aligned}
$$

  similarly: $Q_k \geq Q^* - \Delta_k \;\Rightarrow\; Q_{k+1} \geq Q^* - \gamma \Delta_k$

- The proof translates directly also to value iteration

For completeness**

- **Policy Evaluation** computes V^π instead of V^*: Iterate:

$$\forall s: \; V^\pi_{k+1}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) \; V^\pi_k(s')$$

  Or use matrix inversion $\boldsymbol{V}^\pi = (\boldsymbol{I} - \gamma \boldsymbol{P}^\pi)^{-1} \boldsymbol{R}^\pi$, which is $O(|S|^3)$.

- **Policy Iteration** uses V^π to incrementally improve the policy:
  1. Initialise π_0 somehow (e.g. randomly)
  2. Iterate:
     - **Policy Evaluation:** compute V^{π_k} or Q^{π_k}
     - **Policy Update:** $\pi_{k+1}(s) \leftarrow \text{argmax}_a \, Q^{\pi_k}(s, a)$

`demo: test/mdp runPI`

Summary: Bellman equations

- Discounted infinite horizon:

$$V^*(s) = \max_a Q^*(s,a) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\ V^*(s') \right]$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\ \max_{a'} Q^*(s',a')$$

- With finite horizon T (non stationary MDP), initializing $V_{T+1}(s) = 0$

$$V_t^*(s) = \max_a Q_t^*(s,a) = \max_a \left[R_t(s,a) + \gamma \sum_{s'} P_t(s' \mid s,a)\ V_{t+1}^*(s') \right]$$

$$Q_t^*(s,a) = R_t(s,a) + \gamma \sum_{s'} P_t(s' \mid s,a)\ \max_{a'} Q_{t+1}^*(s',a')$$

- This recursive computation of the value functions is a form of **Dynamic Programming**

Comments & relations

- Tree search is a form of **forward** search, where heuristics (A^* or UCB) may optimistically estimate the value-to-go
- Dynamic Programming is a form of **backward** inference, which exactly computes the value-to-go backward from a horizon
- UCT also estimates $Q(s,a)$, but based on Monte-Carlo rollouts instead of exact Dynamic Programming

- In deterministic worlds, Value Iteration is the same as *Dijkstra backward*; it labels all nodes with the value-to-go ($\leftrightarrow$ cost-to-go).

- In *control theory*, the Bellman equation is formulated for continuous state x and continuous time t and ends-up:

$$-\frac{\partial}{\partial t}V(x,t) = \min_u \Big[c(x,u) + \frac{\partial V}{\partial x} f(x,u)\Big]$$

  which is called *Hamilton-Jacobi-Bellman* equation.
  For linear quadratic systems, this becomes the *Riccati equation*

Comments & relations

- The Dynamic Programming principle is applicable throughout the domains – but inefficient if the state space is large (e.g. relational or high-dimensional continuous)
- It requires iteratively computing a value function over the whole state space

Dynamic Programming in Belief Space

Back to the Bandits

- Can Dynamic Programming also be applied to the Bandit problem? We learnt UCB as the standard approach to address Bandits – but what would be the optimal policy?

Bandits recap

- Let $a_t \in \{1, .., n\}$ be the choice of machine at time t
  Let $y_t \in \mathbb{R}$ be the outcome with mean $\langle y_{a_t} \rangle$
  A policy or strategy maps all the history to a new choice:

$$\pi : \ [(a_1, y_1), (a_2, y_2), ..., (a_{t\text{-}1}, y_{t\text{-}1})] \mapsto a_t$$

- Problem: Find a policy π that

$$\max \langle \sum_{t=1}^{T} y_t \rangle$$

  or

$$\max \langle y_T \rangle$$

- "Two effects" of choosing a machine:
  – You collect more data about the machine $\rightarrow$ knowledge
  – You collect reward

The Belief State

- "Knowledge" can be represented in two ways:
  - as the full history

$$h_t = [(a_1, y_1), (a_2, y_2), ..., (a_{t\text{-}1}, y_{t\text{-}1})]$$

  - as the **belief**

$$b_t(\theta) = P(\theta|h_t)$$

    where θ are the unknown parameters $\theta = (\theta_1, .., \theta_n)$ of all machines

- In the bandit case:
  - The belief factorizes $b_t(\theta) = P(\theta|h_t) = \prod_i b_t(\theta_i|h_t)$
    e.g. for Gaussian bandits with constant noise, $\theta_i = \mu_i$

$$b_t(\mu_i|h_t) = \mathcal{N}(\mu_i|\hat{y}_i, \hat{s}_i)$$

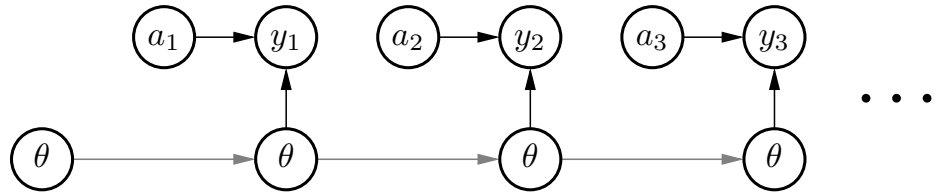
    e.g. for binary bandits, $\theta_i = p_i$, with prior $\mathrm{Beta}(p_i|\alpha, \beta)$:

$$b_t(p_i|h_t) = \mathrm{Beta}(p_i|\alpha + a_{i,t}, \beta + b_{i,t})$$

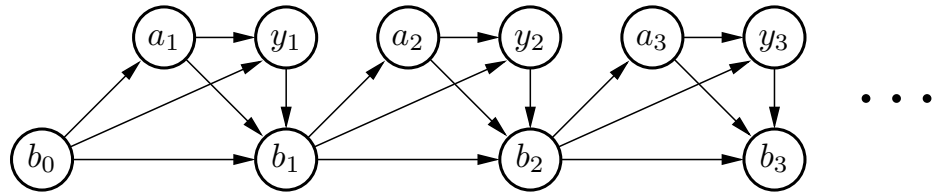
$$a_{i,t} = \textstyle\sum_{s=1}^{t-1} [a_s = i][y_s = 0] \;, \quad b_{i,t} = \sum_{s=1}^{t-1} [a_s = i][y_s = 1]$$

The Belief MDP

- The process can be modelled as



  or as Belief MDP



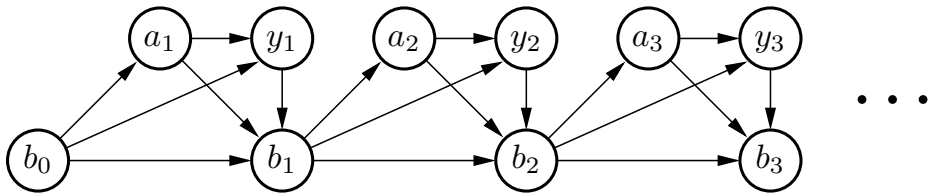
$$P(b'|y,a,b) = \begin{cases} 1 & \text{if } b' = b'_{[b,a,y]} \\ 0 & \text{otherwise} \end{cases}, \quad P(y|a,b) = \int_{\theta_a} b(\theta_a)\, P(y|\theta_a)$$

- The Belief MDP describes a *different* process: the interaction between the information available to the agent (b_t or h_t) and its actions, where *the agent uses his current belief to anticipate observations,* $P(y|a,b)$*.*
- The belief (or history h_t) is all the information the agent has avaiable; $P(y|a,b)$ the "best" possible anticipation of observations. If it acts optimally in the Belief MDP, it acts optimally in the original problem.

  Optimality in the Belief MDP $\Rightarrow$ *optimality in the original problem*

Optimal policies via Dynamic Programming in Belief Space

- The Belief MDP:



$$P(b'|y,a,b) = \begin{cases} 1 & \text{if } b' = b'_{[b,a,y]} \\ 0 & \text{otherwise} \end{cases}, \quad P(y|a,b) = \int_{\theta_a} b(\theta_a)\ P(y|\theta_a)$$

- Belief Planning: Dynamic Programming on the value function

$$\begin{aligned} \forall_b :\ V_{t\text{-}1}(b) &= \max_{\pi} \langle \textstyle\sum_{t=t}^{T} y_t \rangle \\ &= \max_{a_t} \int_{y_t} P(y_t|a_t,b)\ \left[y_t + V_t(b'_{[b,a_t,y_t]}) \right] \end{aligned}$$

$$V_t^*(h) := \max_\pi \int_\theta P(\theta|h)\ V_t^{\pi,\theta}(h) \tag{1}$$

$$V_t^\pi(b) := \int_\theta b(\theta)\ V_t^{\pi,\theta}(b) \tag{2}$$

$$V_t^*(b) := \max_\pi V_t^\pi(b) = \max_\pi \int_\theta b(\theta)\ V_t^{\pi,\theta}(b) \tag{3}$$

$$= \max_\pi \int_\theta P(\theta|b) \left[R(\pi(b), b) + \int_{b'} P(b'|b, \pi(b), \theta)\ V_{t+1}^{\pi,\theta}(b')\right] \tag{4}$$

$$= \max_a \max_\pi \int_\theta P(\theta|b) \left[R(a, b) + \int_{b'} P(b'|b, a, \theta)\ V_{t+1}^{\pi,\theta}(b')\right] \tag{5}$$

$$= \max_a \left[R(a, b) + \max_\pi \int_\theta \int_{b'} P(\theta|b)\ P(b'|b, a, \theta)\ V_{t+1}^{\pi,\theta}(b')\right] \tag{6}$$

$$P(b'|b, a, \theta) = \int_y P(b', y|b, a, \theta) \tag{7}$$

$$= \int_y \frac{P(\theta|b, a, b', y)\ P(b', y|b, a)}{P(\theta|b, a)} \tag{8}$$

$$= \int_y \frac{b'(\theta)\ P(b', y|b, a)}{b(\theta)} \tag{9}$$

$$V_t^*(b) = \max_a \left[R(a, b) + \max_\pi \int_\theta \int_{b'} \int_y b(\theta)\ \frac{b'(\theta)\ P(b', y|b, a)}{b(\theta)}\ V_{t+1}^{\pi,\theta}(b')\right] \tag{10}$$

$$= \max_a \left[R(a, b) + \max_\pi \int_{b'} \int_y P(b', y|b, a) \int_\theta b'(\theta)\ V_{t+1}^{\pi,\theta}(b')\right] \tag{11}$$

$$= \max_a \left[R(a, b) + \max_\pi \int_y P(y|b, a) \int_\theta b'_{[b,a,y]}(\theta)\ V_{t+1}^{\pi,\theta}(b'_{[b,a,y]})\right] \tag{12}$$

Optimal policies

- The value function assigns a value (maximal achievable expected return) to a state of knowledge
- While UCB approximates the value of an action by an optimistic estimate of immediate return; Belief Planning acknowledges that this really is a sequencial decision problem that requires to plan
- Optimal policies "navigate through belief space"
  - This automatically implies/combines "exploration" and "exploitation"
  - There is no need to explicitly address "exploration vs. exploitation" or decide for one against the other. Optimal policies will automatically do this.
- Computationally heavy: b_t is a probability distribution, V_t a function over probability distributions

- The term $\int_{y_t} P(y_t|a_t, b) \left[y_t + V_t(b'_{[b,a_t,y_t]}) \right]$ is related to the *Gittins Index*: it can be computed for each bandit separately.

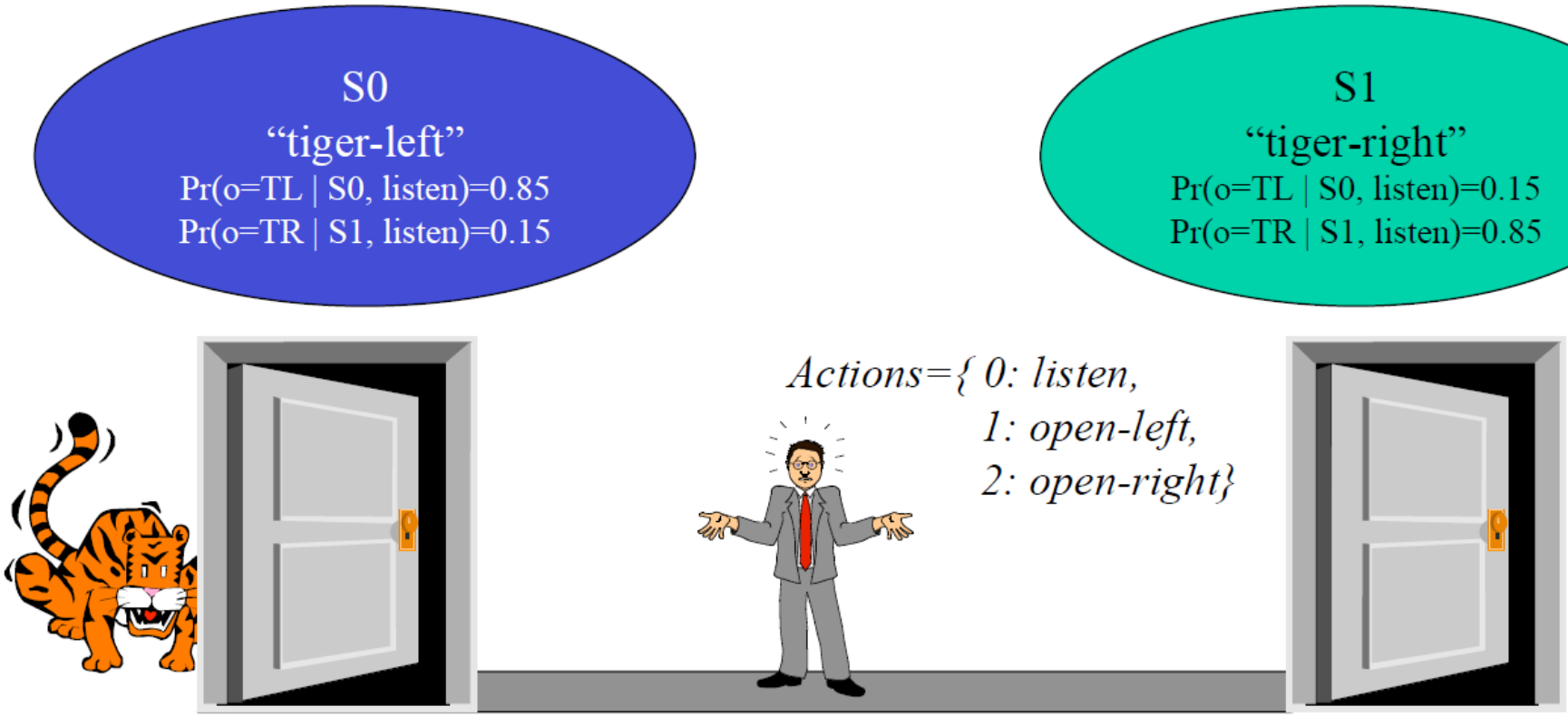
Example exercise

- Consider 3 binary bandits for $T = 10$.
  - The belief is 3 Beta distributions $\text{Beta}(p_i|\alpha + a_i, \beta + b_i)$ $\rightarrow$ 6 integers
  - $T = 10$ $\rightarrow$ each integer ≤ 10
  - $V_t(b_t)$ is a function over $\{0, .., 10\}^6$

- Given a prior $\alpha = \beta = 1$,
  a) compute the optimal value function and policy for the final reward and the average reward problems,
  b) compare with the UCB policy.

- The concept of Belief Planning transfers to other uncertain domains: Whenever decisions influence also the state of knowledge
  - Active Learning
  - Optimization
  - Reinforcement Learning (MDPs with unknown environment)
  - POMDPs

- The tiger problem: a typical POMDP example:



Reward Function

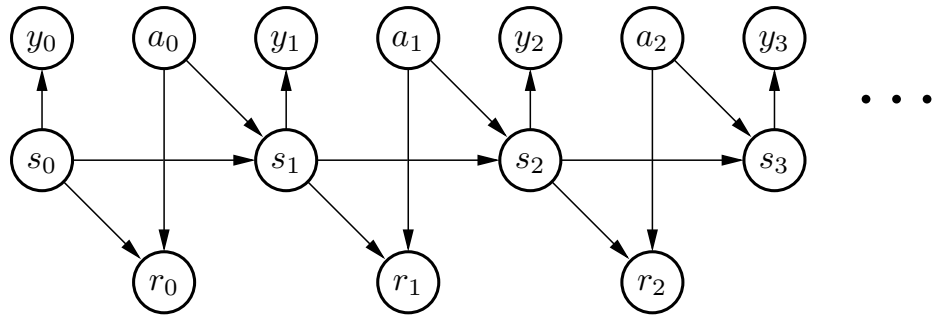
- *Penalty for wrong opening: -100*
- *Reward for correct opening: +10*
- *Cost for listening action: -1*

Observations

- *to hear the tiger on the left (TL)*
- *to hear the tiger on the right(TR)*

(from the a "POMDP tutorial")

Solving POMDPs via Dynamic Programming in Belief Space



- Again, the value function is a function over the belief

$$V(b) = \max_a \left[R(b,s) + \gamma \sum_{b'} P(b'|a,b)\, V(b') \right]$$

- Sondik 1971: V is piece-wise linear and convex: Can be described by m vectors $(\alpha_1, .., \alpha_m)$, each $\alpha_i = \alpha_i(s)$ is a function over discrete s

$$V(b) = \max_i \sum_s \alpha_i(s) b(s)$$

  Exact dynamic programming possible, see Pineau et al., 2003

Approximations & Heuristics

- Point-based Value Iteration (Pineau et al., 2003)
  – Compute $V(b)$ only for a finite set of belief points

- Discard the idea of using belief to "aggregate" history
  – Policy directly maps history (window) to actions
  – Optimize finite state controllers (Meuleau et al. 1999, Toussaint et al. 2008)

Further reading

- *Point-based value iteration: An anytime algorithm for POMDPs.* Pineau, Gordon & Thrun, IJCAI 2003.
- The standard references on the "POMDP page"
  `http://www.cassandra.org/pomdp/`
- *Bounded finite state controllers.* Poupart & Boutilier, NIPS 2003.
- *Hierarchical POMDP Controller Optimization by Likelihood Maximization.* Toussaint, Charlin & Poupart, UAI 2008.

Conclusions

- We covered two basic types of planning methods
  - Tree Search: forward, but with backward heuristics
  - Dynamic Programming: backward

- Dynamic Programming explicitly describes optimal policies. Exact DP is computationally heavy in large domains $\rightarrow$ approximate DP
  Tree Search became very popular in large domains, esp. MCTS using UCB as heuristic

- Planning in Belief Space is fundamental
  - Describes optimal solutions to Bandits, POMDPs, RL, etc
  - But computationally heavy
  - Silver's MCTS for POMDPs annotates nodes with history and belief representatives