Course on Automated Planning: MDP & POMDP Planning; Reinforcement Learning

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, Spain

Models, Languages, and Solvers

- A **planner** is a **solver over a class of models;** it takes a model description, and computes the corresponding controller

$$\textit{Model} \Longrightarrow \boxed{\textit{Planner}} \Longrightarrow \textit{Controller}$$

- Many models, many solution forms: uncertainty, feedback, costs, . . .

- Models described in suitable **planning languages** (Strips, PDDL, PPDDL, . . .) where **states** represent interpretations over the language.

Planning with Markov Decision Processes: Goal MDPs

MDPs are **fully observable, probabilistic** state models:

- a state space S
- initial state $s_0 \in S$
- a set $G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each state $s \in S$
- **transition probabilities** $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
- action costs $c(a, s) > 0$

– **Solutions** are **functions (policies)** mapping states into actions

– **Optimal** solutions minimize **expected cost** from s_0 to goal

Discounted Reward Markov Decision Processes

Another common formulation of MDPs . . .

- a state space S
- initial state $s_0 \in S$
- actions $A(s) \subseteq A$ applicable in each state $s \in S$
- transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
- **rewards** $r(a, s)$ positive or negative
- a **discount factor** $0 < \gamma < 1$; **there is no goal**

– **Solutions** are **functions (policies)** mapping states into actions

– **Optimal** solutions max **expected discounted accumulated reward** from s_0

Partially Observable MDPs: Goal POMDPs

POMDPs are **partially observable, probabilistic** state models:

- states $s \in S$
- actions $A(s) \subseteq A$
- transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
- initial **belief state** b_0
- set of **observable target** states S_G
- action costs $c(a, s) > 0$
- **sensor model** given by probabilities $P_a(o|s)$, $o \in Obs$

– **Belief states** are probability distributions over S

– **Solutions** are policies that map belief states into actions

– **Optimal** policies minimize **expected** cost to go from b_0 to target bel state.

Discounted Reward POMDPs

A common alternative formulation of POMDPs:

- states $s \in S$
- actions $A(s) \subseteq A$
- transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
- initial **belief state** b_0
- **sensor model** given by probabilities $P_a(o|s)$, $o \in Obs$
- **rewards** $r(a, s)$ positive or negative
- **discount factor** $0 < \gamma < 1$; **there is no goal**

– **Solutions** are **policies** mapping states into actions

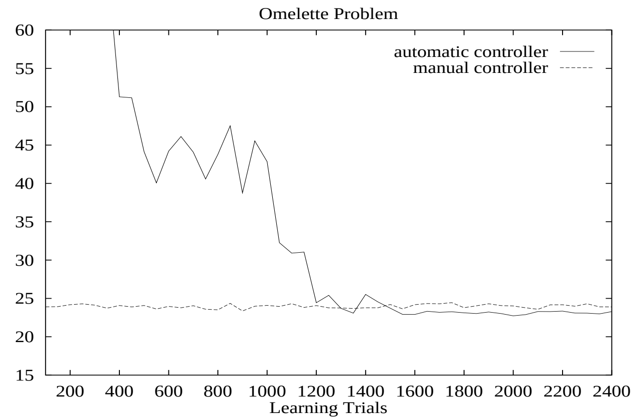
– **Optimal** solutions max **expected discounted accumulated reward** from b_0

Example: Omelette

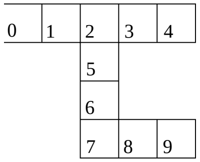
- Representation in GPT (incomplete):

Action:	$\mathbf{grab-egg}()$
Precond:	$\neg holding$
Effects:	$holding := \mathbf{true}$
	$good? := (\mathbf{true}\ 0.5\ ;\ \mathbf{false}\ 0.5)$
Action:	**clean**(bowl:BOWL)
Precond:	$\neg holding$
Effects:	$ngood(bowl) := 0\ \ , nbad(bowl) := 0$
Action:	$\mathbf{inspect}(bowl : BOWL)$
Effect:	$\mathbf{obs}(nbad(bowl) > 0)$

- Performance of resulting controller (2000 trials in 192 sec)



Example: Hell or Paradise; Info Gathering

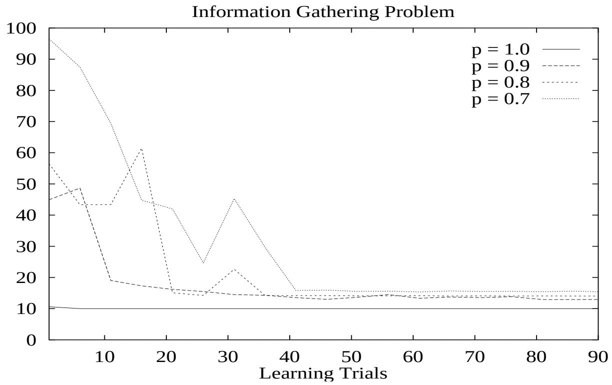


- initial position is 6
- $goal$ and $penalty$ at either 0 or 4; which one not known
- noisy map at position 9

Action: $\mathbf{go-up}()$; same for down,left,right
Precond: $\textsc{free}(\textsc{up}(pos))$
Effects: $pos := \textsc{up}(pos)$

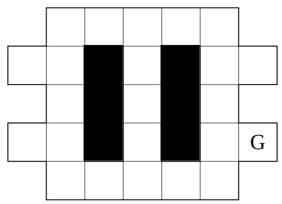
Action: $*$
Effects: $pos = pos9 \ \rightarrow\ \mathbf{obs}(ptr)$
$pos = goal \ \rightarrow\ \mathbf{obs}(goal)$
Costs: $pos = penalty \ \rightarrow\ 50.0$

Ramif: $\mathbf{true} \ \rightarrow\ ptr = (goal\ p\ ;\ penalty\ 1-p)$
Init: $pos = pos6\ ;\ goal = pos0\ \vee\ goal = pos4$
$penalty = pos0\ \vee\ penalty = pos4\ ;\ goal \neq penalty$
Goal: $pos = goal$



Examples: Robot Navigation as a POMDP

- **states:** $[x, y; \theta]$
- **actions** $rotate$ $+90$ and -90, $move$
- **costs:** uniform except when hitting walls
- **transitions:** e.g, $P_{move}([2, 3; 90] \,|\, [2, 2; 90]) = .7$, if $[2, 3]$ is empty, . . .



- **initial** b_0: e.g,, uniform over set of states
- **goal** G: cell marked G
- **observations:** presence or absence of wall with probs that depend on position of robot, walls, etc

Expected Cost/Reward of Policy (MDPs)

- In Goal MDPs, **expected cost of policy** π **starting in** s, denoted as $V^\pi(s)$, is

$$V^\pi(s) = E_\pi[\sum_{s_i} c(a_i, s_i) \mid s_0 = s, a_i = \pi(s_i)]$$

where expectation is **weighted sum** of **cost** of possible state trajectories **times** their **probability** given π

- In Discounted Reward MDPs, **expected discounted reward from** s is

$$V^\pi(s) = E_\pi[\sum_{s_i} \gamma^i r(a_i, s_i) \mid s_0 = s, a_i = \pi(s_i)]$$

Equivalence of (PO)MDPs

- Let the **sign** of a POMDP be **positive** if cost-based and **negative** if reward-based
- Let $V^{\pi}_{M}(b)$ be expected cost (reward) from b in positive (negative) POMDP M
- Define **equivalence** of any two POMDPs as follows; assuming goal states are absorbing, cost-free, and observable:

Definition 1. *POMDPs R and M* **equivalent** *if have same set of non-goal states, and there are constants α and β s.t. for every π and non-target bel b,*

$$V^{\pi}_{R}(b) = \alpha V^{\pi}_{M}(b) + \beta$$

with $\alpha > 0$ if R and M have same sign, and $\alpha < 0$ otherwise.

Intuition: If R and M are equivalent, they have same optimal policies and same 'preferences' over policies

Equivalence Preserving Transformations

- A transformation that maps a POMDP M into M' is **equivalence-preserving** if M and M' are equivalent.

- Three **equivalence-preserving transformation** among POMDP's

  1. $R \mapsto R + C$: addition of C ($+$ or $-$) to all rewards/costs
  2. $R \mapsto kR$: multiplication by $k \neq 0$ ($+$ or $-$) of rewards/costs
  3. $R \mapsto \overline{R}$: elimination of discount factor by adding goal state t s.t.

$$P_a(t|s) = 1 - \gamma \; , \; P_a(s'|s) = \gamma P_a^R(s'|s) \; ; \; O_a(t|t) = 1 \; , \; O_a(s|t) = 0$$

Theorem 1. *Let R be a* **discounted reward-based** POMDP, *and C a constant that bounds all rewards in R from above; i.e. $C > \max_{a,s} r(a,s)$. Then, $M = \overline{-R + C}$ is a* **goal** POMDP *equivalent to R.*

Computation: Solving MDPs

Conditions that ensure **existence** of optimal policies and **correctness** (convergence) of some of the methods we'll see:

- For **discounted MDPs**, $0 < \gamma < 1$, none needed as everything is bounded; e.g. discounted cumulative reward no greater than $C/1 - \gamma$, if $r(a, s) \leq C$ for all a, s

- For **goal MDPs**, absence of **dead-ends** assumed so that $V^*(s) \neq \infty$ for all s

Basic Dynamic Programming Methods: Value Iteration (1)

- **Greedy policy** π_V for $V = V^*$ is **optimal**:

$$\pi_V(s) = \arg\min_{a \in A(s)}[c(s,a) + \sum_{s' \in S} P_a(s'|s)V(s')]$$

- Optimal V^* is unique solution to **Bellman's optimality equation** for MDPs

$$V(s) = \min_{a \in A(s)}[c(s,a) + \sum_{s' \in S} P_a(s'|s)V(s')]$$

where $V(s) = 0$ for goal states s

- For **discounted reward MDPs**, Bellman equation is

$$V(s) = \max_{a \in A(s)}[r(s,a) + \gamma \sum_{s' \in S} P_a(s'|s)V(s')]$$

Basic DP Methods: Value Iteration (2)

- **Value Iteration** finds V^* solving Bellman eq. by **iterative procedure:**
  - ▷ Set V_0 to arbitrary value function; e.g., $V_0(s) = 0$ for all s
  - ▷ Set V_{i+1} to result of Bellman's **right hand side** using V_i in place of V:

$$V_{i+1}(s) := \min_{a \in A(s)} [c(s,a) + \sum_{s' \in S} P_a(s'|s) V_i(s')]$$

- $V_i \mapsto V^*$ as $i \mapsto \infty$
- $V_0(s)$ must be initialized to 0 for all goal states s

(Parallel) Value Iteration and Asynchronous Value Iteration

- Value Iteration (VI) converges to **optimal value function** V^* asympotically
- Bellman eq. for **discounted reward** MDPs similar, but with **max** instead of **min**, and sum multiplied by γ
- In practice, VI stopped when **residual** $R = \max_s |V_{i+1}(s) - V_i(s)|$ is small enough
- Resulting greedy policy π_V has **loss** bounded by $2\gamma R/1-\gamma$
- **Asynchronous Value Iteration** is **asynchronous** version of VI, where states **updated in any order**
- Asynchronous VI also converges to V^* when **all states updated infinitely often**; it can be **implemented** with single V vector

Policy Evaluation

- **Expected cost** of policy π from s to goal, $V^\pi(s)$, is weighted avg of **cost** of **state trajectories** $\tau : s_0, s_1, \ldots,$ times their **probability** given π
- **Trajectory cost** is $\sum_{i=0,\infty} cost(\pi(s_i), s_i)$ and **probability** $\prod_{i=0,\infty} P_{\pi(s_i)}(s_{i+1}|s_i)$
- Expected costs $V^\pi(s)$ can also be characterized as solution to Bellman equation

$$V^\pi(s) \;=\; c(a,s) \;+\; \sum_{s' \in S} P_a(s'|s) V^\pi(s')$$

  where $a = \pi(s)$, and $V^\pi(s) = 0$ for goal states
- This set of **linear equations** can be solved analytically, or by VI-like procedure
- **Optimal expected cost** $V^*(s)$ is $\min_\pi V^\pi(s)$ and **optimal policy** is the $\arg\min$
- For **discounted reward** MDPs, all similar but with $r(s,a)$ instead of $c(a,s)$, max instead of min, and sum discounted by γ

Policy Iteration (Howard)

- Let $Q^\pi(a,s)$ be **expected cost** from s when doing a first and then π

$$Q^\pi(a,s) \;=\; c(a,s) \;+\; \sum_{s' \in S} P_a(s'|s) V^\pi(s')$$

- When $Q^\pi(a,s) < Q^\pi(\pi(s),s)$, π **strictly improved** by changing $\pi(s)$ to a

- **Policy Iteration (PI)** computes π^* by seq. of **evaluations** and **improvements**

  1. Starting with arbitrary policy π
  2. Compute $V^\pi(s)$ for all s (**evaluation**)
  3. Improve π by setting $\pi(s)$ to $a = \arg\min_{a \in A(s)} Q^\pi(a,s)$ (**improvement**)
  4. If π changed in 3, go back to 2, else **finish**

- PI finishes with π^* after **finite** number of iterations, as # of policies is **finite**

Dynamic Programming: The Curse of Dimensionality

- **VI** and **PI** need to deal with value vectors V of size $|S|$

- **Linear programming** can also be used to get V^* but $O(|A||S|)$ constraints:

$$\max_V \sum_s V(s) \text{ subject to } V(s) \leq c(a,s) + \sum_{s'} P_a(s'|s)V(s') \text{ for all } a, s$$

  with $V(s) = 0$ for goal states

- MDP problem is thus **polynomial** in S but **exponential** in $\#$ vars

- Moreover, **this is not worst case**; vectors of size $|S|$ needed **to get started!**

Question: Can we do better?

Dynamic Programming and Heuristic Search

- **Heuristic search** algorithms like A* and IDA* manage to solve **optimally** problems with more than 10^{20} states, like Rubik's Cube and the 15-puzzle
- For this, **admissible heuristics** (lower bounds) used to **focus/prune** search
- Can admissible heuristics be used for **focusing updates** in DP methods?
- Often states **reachable** with **optimal policy** from s_0 much smaller than S
- Then convergence to V^* **over all** s not needed for **optimality** from s_0

Theorem 2. *If* V *is an* **admissible** *value function s.t. the* **residuals** *over the states reachable with* π_V *from* s_0 *are all zero, then* π_V *is an* **optimal policy** *from* s_0 *(i.e. it minimizes* $V^\pi(s_0)$*)*

Learning Real Time A* (LRTA*) Revisited

1. **Evaluate** each action a in s as: $Q(a, s) = c(a, s) + V(s')$
2. **Apply** action $\mathbf{a}$ that minimizes $Q(\mathbf{a}, s)$
3. **Update** $V(s)$ to $Q(\mathbf{a}, s)$
4. **Exit** if s' is goal, else go to 1 with $s := s'$

- LRTA* can be seen as **asynchronous value iteration** algorithm for **deterministic** actions that takes advantage of theorem above (i.e. updates = DP updates)
- **Convergence** of LRTA* to V implies residuals along π_V reachable states from s_0 are all zero
- Then 1) $V = V^*$ along such states, 2) $\pi_V = \pi^*$ from s_0, but 3) $V \neq V^*$ and $\pi_V \neq \pi^*$ over other states; yet this is irrelevant given s_0

Real Time Dynamic Programming (RTDP) for MDPs

RTDP is a generalization of LRTA* to MDPs due to (Barto et al 95); just adapt Bellman equation used in the **Eval** step

1. **Evaluate** each action a applicable in s as

$$Q(a, s) = c(a, s) + \sum_{s' \in S} P_a(s'|s) V(s')$$

2. **Apply** action $\mathbf{a}$ that minimizes $Q(\mathbf{a}, s)$
3. **Update** $V(s)$ to $Q(\mathbf{a}, s)$
4. **Observe** resulting state s'
5. **Exit** if s' is goal, else go to 1 with $s := s'$

Same properties as LRTA* but over MDPs: **after repeated trials**, greedy policy eventually becomes **optimal** if $V(s)$ initialized to admissible $h(s)$

Find-and-Revise: A General DP + HS Scheme

- Let $Res_V(s)$ be **residual** for s given **admissible** value function V

- **Optimal** π for MDPs from s_0 can be obtained for sufficiently small $\epsilon > 0$:

  1. **Start** with admissible V; i.e. $V \leq V^*$
  2. **Repeat:** find s reachable from π_V & s_0 with $Res_V(s) > \epsilon$, and **Update** it
  3. **Until** no such states left

- V remains **admissible** (**lower bound**) after updates

- **Number of iterations** until convergence bounded by $\sum_{s \in S}[V^*(s) - V(s)]/\epsilon$

- Like in **heuristic search**, convergence achieved **without visiting or updating** many of the states in S; LRTDP, LAO*, ILAO*, HDP, LDFS, etc. are algorithms of this type

POMDPs are MDPs over Belief Space

- Beliefs b are **probability distributions** over S

- An action $a \in A(b)$ maps b into b_a

$$b_a(s) = \sum_{s' \in S} P_a(s|s')b(s')$$

- The probability of observing o then is:

$$b_a(o) = \sum_{s \in S} P_a(o|s)b_a(s)$$

- . . . and the new belief is

$$b_a^o(s) = P_a(o|s)b_a(s)/b_a(o)$$

RTDP for POMDPs

Since POMDPs are MDPs over belief space algorithm for POMDPs becomes

1. **Evaluate** each action a applicable in b as

$$Q(a,b) = c(a,b) + \sum_{o \in O} b_a(o) V(b_a^o)$$

2. **Apply** action $\mathbf{a}$ that minimizes $Q(a,b)$
3. **Update** $V(b)$ to $Q(\mathbf{a},b)$
4. **Observe** o
5. **Compute** new belief state b_a^o
6. **Exit** if b_a^o is a final belief state, else set b to b_a^o and go to 1

- Resulting algorithm, called RTDP-Bel, **discretizes** beliefs b for writing to and reading from hash table
- RTDP-Bel competitive in quality and performance with **Point-based POMDP** based algorithms that do not (see paper at IJCAI-09)

Variations on RTDP : Reinforcement Learning

Q-learning is a **model-free** version of RTDP; Q-values initialized arbitrarily and **learned by experience**

1. **Apply** action $\mathbf{a}$ that minimizes $Q(\mathbf{a}, s)$ with probability $1 - \epsilon$, with probability ϵ, choose $\mathbf{a}$ randomly
2. **Observe** resulting state s' and collect cost c
3. **Update** $Q(\mathbf{a}, s)$ to

$$Q(\mathbf{a}, s) + \alpha[c + \text{min}_a Q(a, s') - Q(\mathbf{a}, s)]$$

4. **Exit** if s' is goal, else with $s := s'$ go to 1

- Q-learning converges asympotically to **optimal** Q-values, when all actions and states visited **infinitely often**
- Q-learning solves MDPs optimally without model parameters (probabilities, costs)

Variations on RTDP : Reinforcement Learning (2)

More familiar **Q-learning** algorithm formulated for **discounted reward MDPs:**

1. **Apply** action $\mathbf{a}$ that maximizes $Q(\mathbf{a}, s)$ with probability $1 - \epsilon$, with probability ϵ, choose $\mathbf{a}$ randomly
2. **Observe** resulting state s' and collect **reward** r
3. **Update** $Q(\mathbf{a}, s)$ to

$$Q(\mathbf{a}, s) + \alpha[r + \gamma \, \text{max}_a Q(a, s') \; - \; Q(a, s)]$$

4. **Exit** if s' is goal, else with $s := s'$ go to 1

- Q-values initialized arbitrarily
- This version solves **discounted reward MDPs**

Why RL works? Intuitions

N-armed bandit problem: simpler problem without **state**:

- Choose repeatedly one of n actions a (levers)
- Get 'stochastic' reward r_t at time t that depends on action chosen
- **How to play to maximize reward in long term**; e.g. 10000 plays?
- Need to find out value of actions (**exploration**) and then play best (**exploitation**)
- For this, choose 'greedy' a that maximizes $Q_t(a)$ with probability $1 - \epsilon$, where
  - ▷ Average: $Q_{t+1}(a) = r_1 + r_2 + \ldots + r_{t+1}/t + 1$
  - ▷ Incremental: $Q_{t+1}(a) = Q_t(a) \, + \, [r_{t+1} - Q_t(a)]/(t+1)$
  - ▷ Recency Weighted Avg: $Q_{t+1}(a) = Q_t(a) + \alpha \, [r_{t+1} - Q_t(a)]$
- Last expression similar to the one for Q-learning, except for states . . .

Monte Carlo RL Prediction and Learning

Assuming underlying **discounted reward MDP** with **unknown pars:**

- Eval policy π by sampling executions s_0, s_1, . . . ,
- For each state s_t visited, collect return $R_t = \sum_{k \geq 0} \gamma^k r(a_{t+k}, s_{t+k})$
- Approximate $V^\pi(s_t)$ to **average** of returns R_t)
- In order to learn **control** not just **values**, approx $Q^\pi(a, s_t)$

Monte Carlo vs. TD Predictions (Sutton & Barto)

- Incremental **Monte Carlo** updates for prediction are

$$V(s_t) := V(s_t) + \alpha[R_t - V(s_t)]$$

- **TD Methods** as used in Q-learning, **bootstrap**:

$$V(s_t) := V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)]$$

- Other types of **returns** can be used as well; e.g. n-step return R_t^n

$$V(s_t) := V(s_t) + \alpha[r_t + \gamma r_{t+1} + \cdots + \gamma r_{t+n-1} + \gamma^n V(s_{t+n}) - V(s_t)]$$

- $TD(\lambda)$, $0 \leq \lambda \leq 1$, uses linear combination of returns R_t^n for all n

$$V(s_t) := V(s_t) + \alpha[R_t^\lambda - V(s_t)]$$

where $R_t^\lambda = (1 - \lambda) \sum_{n=1,\infty} \lambda^{n-1} R_t^n$