Markov Decision Process

based on slides of Branislav Bošanský and Jan Mrkos

Markov Decision Processes

- Main formal model
- $\langle S, A, D, T, R \rangle$
  - states – a finite set of states of the world
  - actions – a finite set of actions the agent can perform
  - horizon – a finite/infinite set of time steps $(1,2, \ldots)$
  - transition function
    - $T: S \times A \times S \to [0,1]; \sum_{s' \in S} T(s, a, s') = 1$
  - reward function
    - $R: S \times A \times S \to \mathbb{R}$
    - typically bounded

MDP – policy

- history-dependent policy
  - $\pi: H \times A \rightarrow [0,1]; \sum_{a \in A} \pi(h, a) = 1$
- for simple cases we do not need history and randomization
  - Markov assumption
  - finite-horizon MDPs
  - infinite-horizon MDPs with reward discount factor $0 \leq \gamma < 1$
  - stochastic shortest path
  - (… and some others)
- from now on, policy is an assignment of an action in each state and time

MDP – policy (2)

- $\pi: S \rightarrow A$
- **stationary policy**
  - when the policy is same every time state s is visited
  - otherwise – **nonstationary policy**
- **positional policy**
  - deterministic and stationary policy

MDP – value of a policy

- we can express an expected reward for every state and time-step when specific policy is followed
- $V_\pi^k(s) = \mathbb{E}\left[\sum_{t=0}^{k} \gamma^t \cdot R(s_t, a_t, s_{t+1}) \,|s_0 = s, a_t = \pi(s_t)\right]$
  - optimal policy : $\pi^{*,k}(s) = \underset{\pi}{\operatorname{argmax}}\, V_\pi^k(s)$
- for large (infinite) k we can approximate the value by dynamic programming
  - $V_\pi^0(s) = 0$
  - $V_\pi^k(s) = \sum_{s' \in S} T(s, a, s')\,[R(s, a, s') + \gamma V_\pi^{k-1}(s')] \qquad a = \pi(s)$

MDP – towards finding optimal policy

- we can exploit the concept of dynamic programming to find an optimal policy
- basic algorithm for solving MDPs based on Bellman's equation
- **value iteration**
  - $V^0(s) = 0 \quad \forall s \in S$
  - $V^k(s) = \max_{a \in A} \underbrace{\sum_{s' \in S} T(s,a,s')\left[R(s,a,s') + \gamma V^{k-1}(s')\right]}_{\text{Q-function } (Q(s,a))}$
- for $k \to \infty$ values converge to optimum $V^k \to V^*$

value Iteration algorithm

Basic algorithm for finding solution of Bellman Equations iteratively.

1. initialize V_0 arbitrarily for each state, e.g to 0, set $n = 0$
2. Set $n = n + 1$.
3. Compute Bellman Backup, i.e. for each $s \in S$:
   3.1 $V_n(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_{n-1}(s')]$
4. GOTO 2.

Question: Does it converge? How fast? When do we stop?

MDP – convergence of value iteration

- value iteration converges
  - for finite-horizon MDPs: $|D|$ steps
  - for infinite-horizon: asymptotically
    - we can measure residual r and stop if it is small enough ($\mathrm{r} \leq \varepsilon(1-\gamma)/\gamma$)
    - $r = \max_{s \in S} |V_{i+1}(s) - V_i(s)|$
    - convergence depends on γ

VI with stopping criterion

1. initialize V_0 arbitrarily for each state, e.g to 0, set $n = 0$
2. Set $n = n + 1$.
3. Compute Bellman Backup, i.e. for each $s \in S$:
   3.1 $V_n(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_{n-1}(s')]$
   3.2 Calculate residual $Res = \max_{s \in S} |V_n(s) - V_{n-1}(s)|$
4. if $res > \epsilon$ GOTO 2. else TERMINATE

VI with stopping criterion

1. initialize V_0 arbitrarily for each state, e.g to 0, set $n = 0$
2. Set $n = n + 1$.
3. Compute Bellman Backup, i.e. for each $s \in S$:
   3.1 $V_n(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_{n-1}(s')]$
   3.2 Calculate residual $Res = \max_{s \in S} |V_n(s) - V_{n-1}(s)|$
4. if $res > \epsilon$ GOTO 2. else TERMINATE

Question: What is the policy?

VI with stopping criterion

1. initialize V_0 arbitrarily for each state, e.g to 0, set $n = 0$
2. Set $n = n + 1$.
3. Compute Bellman Backup, i.e. for each $s \in S$:
    3.1 $V_n(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_{n-1}(s')]$
    3.2 Calculate residual $Res = \max_{s \in S} |V_n(s) - V_{n-1}(s)|$
4. if $res > \epsilon$ GOTO 2. else TERMINATE

Question: What is the policy?

- *Greedy policy* π_n^V is the policy given as argmax of V_n.

MDP – extracting policy and policy iteration

- value iteration calculates only values
- the optimal policy can be extracted by using a greedy approach
  - $\pi^k(s) = \arg\max_{a \in A} \sum_{s' \in S} T^k(s,a,s') \left[R^k(s,a,s') + \gamma V^k(s')\right]$

- alternative algorithm – **policy iteration**
  - starts with an arbitrary policy
    - **policy evaluation:** recalculates value of states given the current policy π^k
    - **policy improvement:** calculates a new maximum expected utility policy π^{k+1}
  - until the strategy changes

MDP – VI/PI improvements

- value iteration is very simple
  - updates all states during each iteration
  - curse of dimensionality (huge state space)
  - **asynchronous VI**
    - select a single state to be updated in each iteration separately
    - each state must be updated infinitely often to guarantee convergence
    - lower memory requirements

- **Q: Can we use some heuristics to improve the convergence?**

Asynchronous VI

1. initialize V_0 arbitrarily for each state, e.g to 0
2. While $Res^V > \epsilon$, do:
   2.1 pick some state s
   2.2 Bellman backup $V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V(s')]$
   2.3 Update residual at s $Res^V(s) = |V_{\text{old}}(s) - V_{\text{new}}(s)|$

Asynchronous VI

1. initialize V_0 arbitrarily for each state, e.g to 0
2. While $Res^V > \epsilon$, do:
   2.1 pick some state s
   2.2 Bellman backup $V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')]$
   2.3 Update residual at s $Res^V(s) = |V_{\text{old}}(s) - V_{\text{new}}(s)|$

Question: Memory requirements compared to VI?

Asynchronous VI

1. initialize V_0 arbitrarily for each state, e.g to 0
2. While $Res^V > \epsilon$, do:
   2.1 pick some state s
   2.2 Bellman backup $V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V(s')]$
   2.3 Update residual at s $Res^V(s) = |V_{\text{old}}(s) - V_{\text{new}}(s)|$

Question: Memory requirements compared to VI?

Question: Convergence condition?

- Asymptotic as VI under condition that every state visited ∞ often.

Asynchronous VI

1. initialize V_0 arbitrarily for each state, e.g to 0
2. While $Res^V > \epsilon$, do:
   2.1 pick some state s
   2.2 Bellman backup $V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V(s')]$
   2.3 Update residual at s $Res^V(s) = |V_{\text{old}}(s) - V_{\text{new}}(s)|$

Question: Memory requirements compared to VI?

Question: Convergence condition?

- Asymptotic as VI under condition that every state visited ∞ often.

Question: How to pick s in 2.1?

- Simplest is *Gauss-Seidel VI*, that is run AVI over all states iteratively

MDP – VI/PI heuristics

- initial values can be assigned better
  - we can use a heuristic function instead of 0

- **Q: Can you think of any heuristic function?**
  - e.g., remember FFReplan/Robust FF?
  - we can use a single run of a planner on the determinized version

- **Q: What if the values V are initialized incorrectly?**

MDP – VI/PI with priority

- initialize V and a priority queue q
- select state s from the top of q and perform a Bellman backup
- add all possible predecessors of s to q
- repeat until convergence
  - priorities: changes in utility, position in the graph, …

- but, values are still updated regardless on the current values
- consider a typical probabilistic planning problem
  - finite-horizon MDP with some goal states

MDPs – Find and Revise

- we can further combine selective updates with heuristic search
  - starts with admissible $V(s) \geq V^*(s)$ for all states
  - select next state s' that is:
    - reachable from s_0 using current greedy policy π_V, and
    - residual $r(s') > \varepsilon$
  - update s'
  - repeat until such states exist

- many further improvements and algorithms …

MDPs – Real-Time Dynamic Programming

- updates the values only on the path from the starting state to the goal
- during one iteration updates one rollout/trial:
  - start with $s = s_0$
  - evaluate all actions using Bellman's Q-functions $Q(s, a)$
  - select action that maximizes current value: $\arg\max_{a \in A} Q(s, a)$
  - set $V(s) \leftarrow Q(s, a)$
  - get resulting state s'
  - if s' is not goal, then $s \leftarrow s'$ and go to step 2
- can be further improved with labeling (LRTDP) to identify solved states