Markov Decision Processes and Monte Carlo Tree Search

Branislav Bošanský

PAH 2016/2017

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 $(1,2, \dots)$
  - transition function
    - $T: S \times A \times S \rightarrow [0,1]; \sum_{s' \in S} T(s, a, s') = 1$
  - reward function
    - $R: S \times A \times S \rightarrow \mathbb{R}$
    - typically bounded

MDP – policy

- history-dependent policy
  - $\pi: H \times A \to [0,1]; \sum_{a \in A} \pi(h,a) = 1$
- 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 : $\pi^{*,k}(s) = \operatorname{argmax}_\pi V_\pi^k(s)$
- for large (infinite) k we can approximate the value by dynamic programming
  - $V_\pi^0(s) = 0$
  - $V_\pi^k(s) = \sum_{s' \in S} T(s, a, s') \left[R(s, a, s') + \gamma V_\pi^{k-1}(s')\right] \qquad a = \pi(s)$

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**
  - $V^0(s) = 0 \quad \forall s \in S$
  - $V^k(s) = \max_{a \in A} \underbrace{\sum_{s' \in S} T(s,a,s')\left[R(s,a,s') + \gamma V^{k-1}(s')\right]}_{\text{Q-function } (Q(s,a))}$
  - Q-function ($Q(s,a)$)
- for $k \to \infty$ values converge to optimum $V^k \to V^*$

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

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?**

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

MDPs – Using Monte Carlo Methods

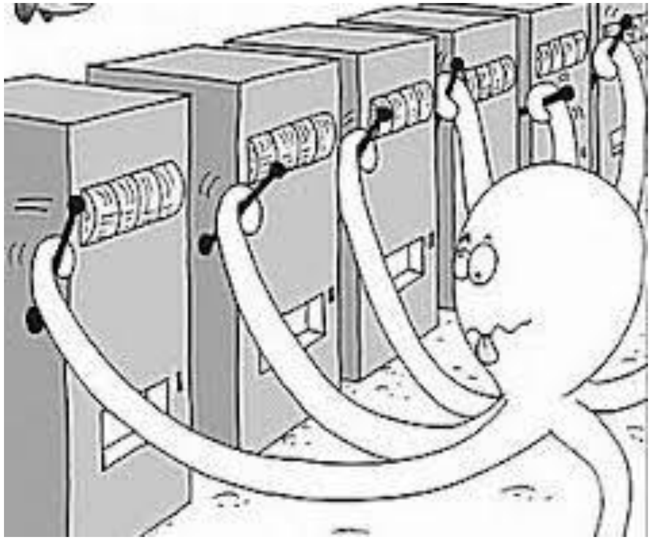
- Monte Carlo Simulation: a technique that can be used to solve a mathematical or statistical problem using repeated sampling to determine the properties of some phenomenon (or behavior)
- Monte-Carlo Planning: compute a good policy for an MDP by interacting with an MDP simulator
- when simulator of a planning domain is available or can be learned from data
  - even if not described as a MDP
  - queries has to be cheap (relatively)

MDPs – Using Monte Carlo Methods

- Monte Carlo sampling is a well known method for searching through large state space
- exploiting MC in sequential decision making has first been successfully designed in (Kocsis & Szepesvari, 2006)
- foundations in mathematical theory
  - Multi-Armed Bandit (MAB) Problem
  - Upper Confidence Bounds (UCB)
  - exploration/exploitation dilemma

MDPs – Using Monte Carlo Methods

- sequential decision problem (over a single state)
- $k \geq 2$ stochastic actions (arms a_i)
  - each parameterized with an unknown probability distribution ν_i
  - each with a stored expectation μ_i
  - if executed (pulled) rewarded at random from ν_i
- objective
  - get maximal reward after N pulls
  - minimize regret for pulling wrong arm(s)



MCTS - ϵ-greedy

- parameterized by $\boldsymbol{\epsilon}$
- flip a $\boldsymbol{\epsilon}$-biased coin
  - ($\boldsymbol{\epsilon}$): select arm a_i randomly with uniform probability and update μ_i
  - ($1-\boldsymbol{\epsilon}$): select estimated best arm a^* and update μ^*
- typically $\boldsymbol{\epsilon} \approx \mathbf{0.1}$ (but this can vary depending on circumstances)

MCTS – UCB1

Upper Confidence Bounds

- UCB1 arm selection:
  - select arm a_i maximizing UCB1 formula:

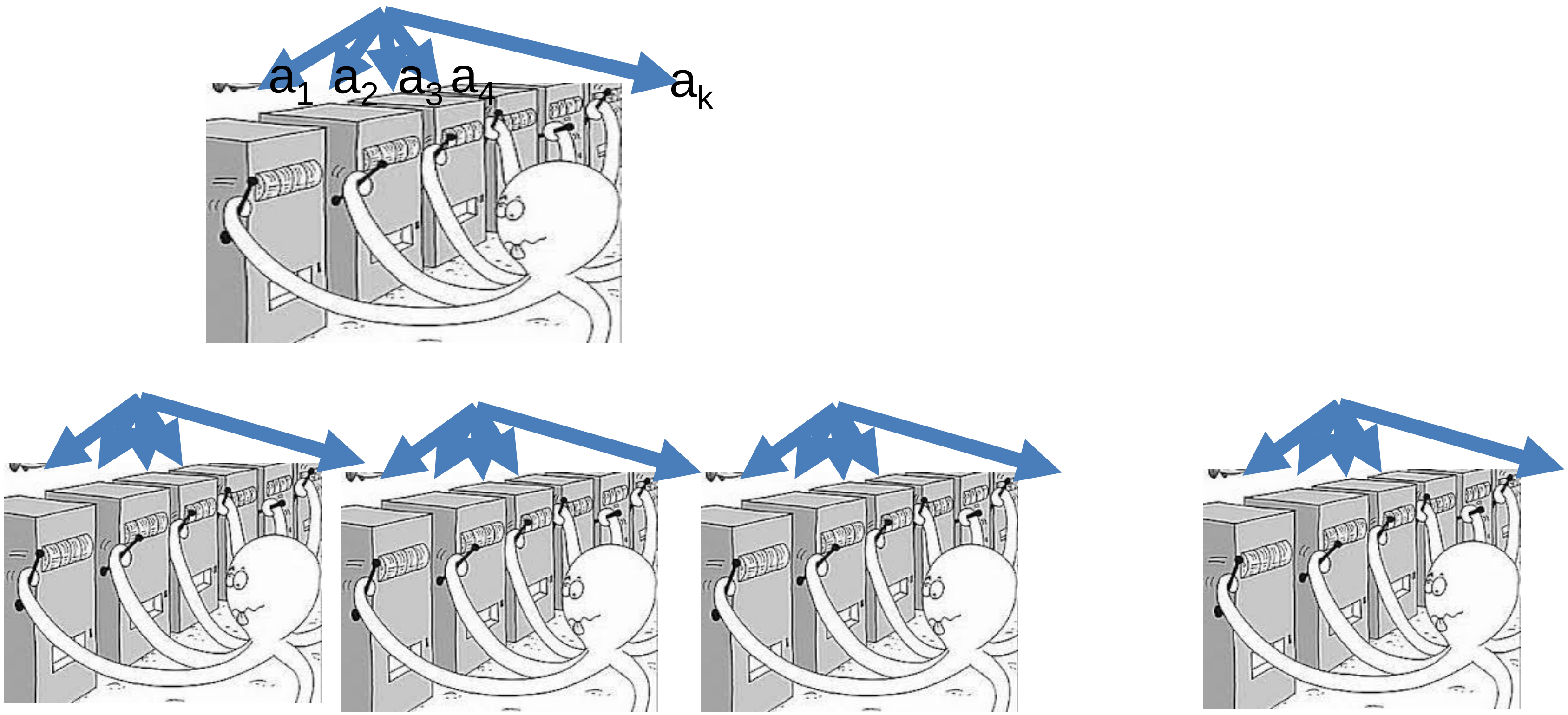
$$\mu_i + c\sqrt{\frac{\ln n}{n_i}}$$

  and update μ_i

  - n – times the state is visited; n_i – times the action is visited
  - μ_i – average reward from the previous plays
- exploration factor ensures to evaluate actions that are evaluated rarely

MCTS – from UCB1 to UCT

- UCB1 applied on trees – UCT



MCTS – from UCB1 to UCT

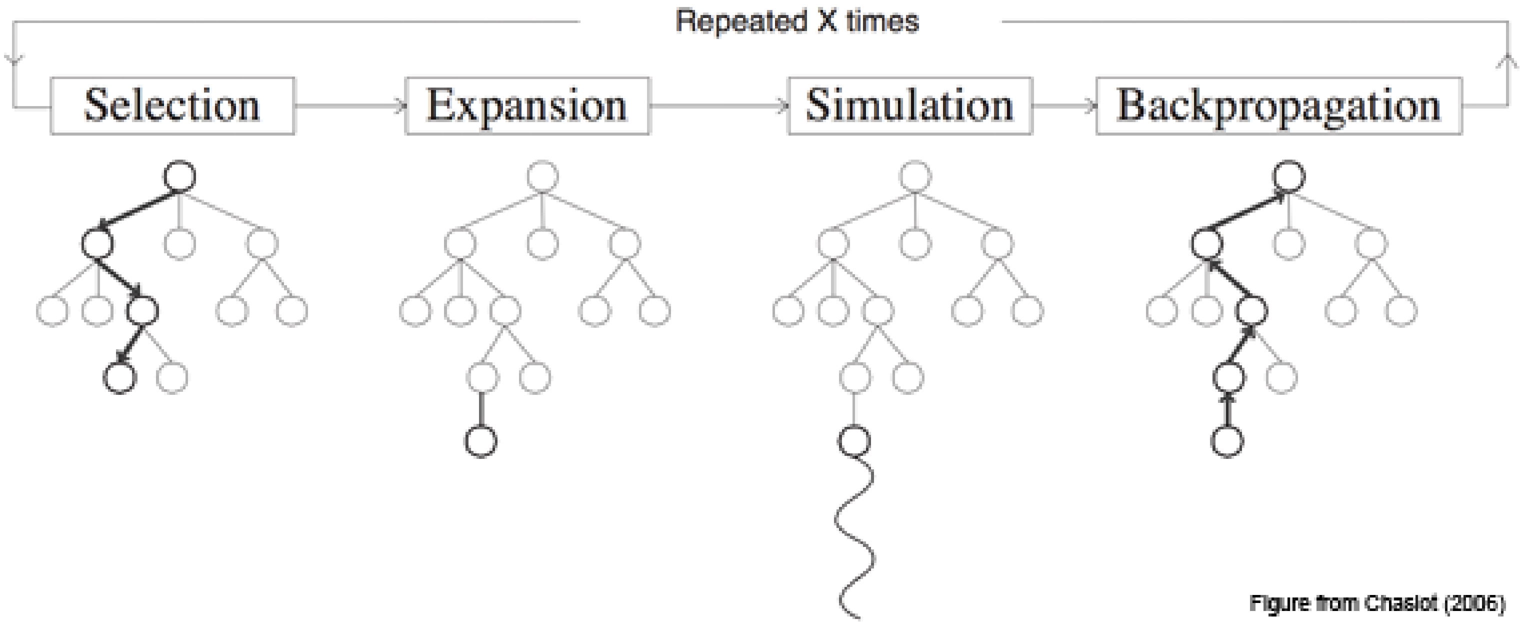


Figure from Chaslot (2006)

MCTS – UCT

Upper Confidence Bounds on Trees

- selection (UCB1)
  - for each action a_i applicable in s UCB selects the one that maximizes

$$c\sqrt{\frac{\ln n}{n_i}} + \sum_{s' \in S} T(s, a_i, s')[R(s, a_i, s') + \gamma V(s')]$$

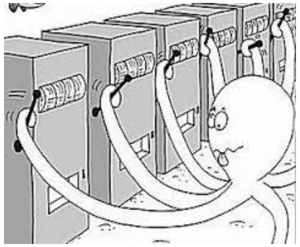
  - n – times the state is visited; n_i – times the action is visited
  - $V(s)$ – average reward from the previous iterations
  - c - exploration constant (linear to expected utility)
- exploration factor ensures to evaluate actions that are evaluated rarely

MCTS – UCT

Upper Confidence Bounds on Trees

- expansion (MCTS)
  - in a selection node where not all actions were yet sampled, expand (uniformly) randomly one of the new nodes
- simulation (MCTS)
  - (uniformly) randomly select actions in decision nodes
  - using the simulator based on the probabilities in the MDP simulate world behavior in the chance nodes MDP
- backup (MCTS)
  - updating μ_i^s for all search tree nodes along the trial based on the rewards (incl. the simulation)

MDPs – Using Monte Carlo Methods



- learning-while-acting
  - reward for each action
  - cumulative regret (exploration/exploitation dilemma)
  - algorithms: ϵ-greedy, UCB1
  - used in: Monte Carlo Tree Search, UCB1 applied to trees (UCT)
- online planning/learning-while-planning
  - reward only for final decision (N "free action tries" by simulator)
  - simple regret
  - algorithms: uniform sampling, ϵ-greedy, Sequential Halving
  - used in: Trial-based Heuristic Tree Search (THTS)