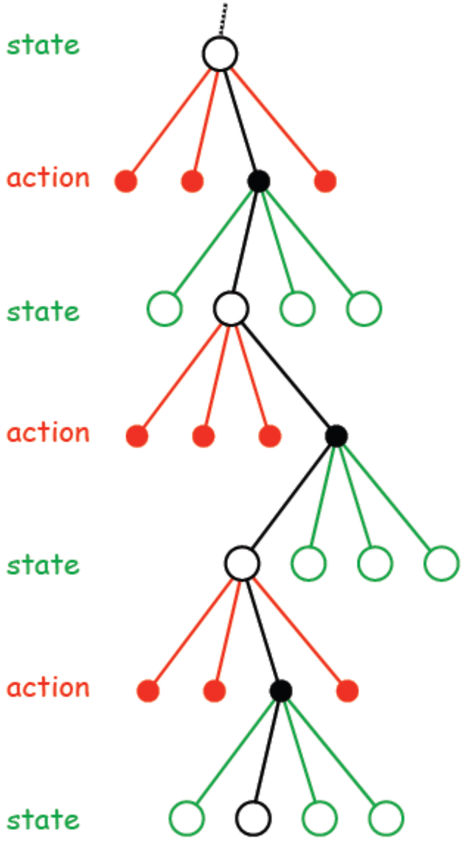
Reinforcement Learning

Temporal-Difference (TD) Learning

Subramanian Ramamoorthy
School of Informatics

31 January, 2017

Learning in MDPs



- You are learning from a long stream of experience:
  $s_0 a_0 r_0 s_1 a_1 r_1 \ldots s_k a_k r_k \ldots$

  … up to some terminal state
- Direct methods:
  Approximate value function (V/Q) straight away - without computing $\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$

Should you wait until episodes end or can you learn on-line?

Recap: Incremental Monte Carlo Algorithm

- Incremental sample-average procedure:

$$V(s) \leftarrow V(s) + \frac{1}{n(s)}[R - V(s)]$$

- Where $n(s)$ is number of first visits to state s
  - Note that we make one update, for each state, per episode
- One could pose this as a generic constant step-size algorithm:

$$V(s) \leftarrow V(s) + \alpha[R - V(s)]$$

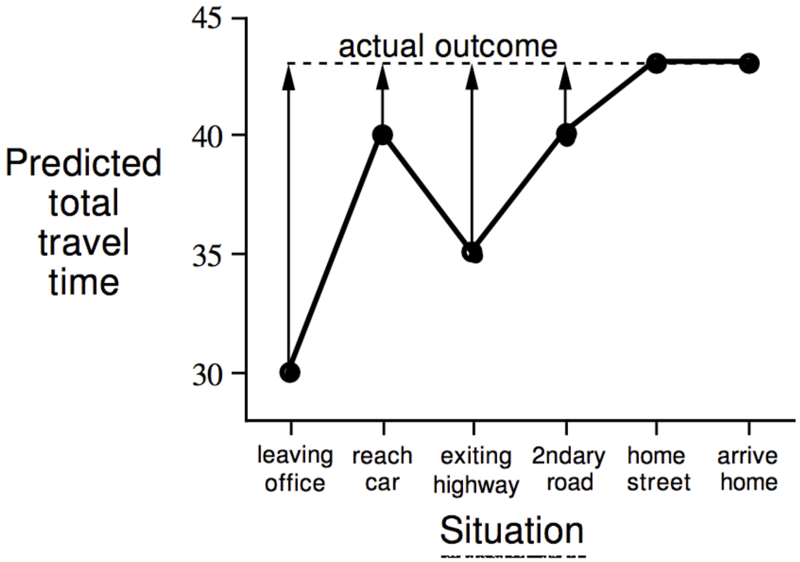
  - Useful in tracking non-stationary problems (task + environment)

Example: Driving Home

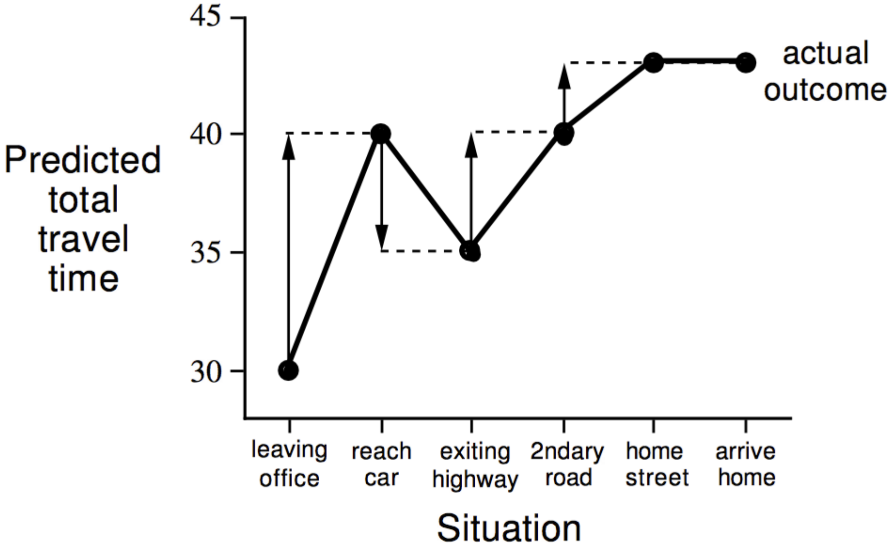
State	Elapsed Time (minutes)		Predicted Time to Go	Predicted Total Time
leaving office	0		30	30
reach car, raining	5	(5)	35	40
exit highway	20	(15)	15	35
behind truck	30	(10)	10	40
home street	40	(10)	3	43
arrive home	43	(3)	0	43

Driving Home

Changes recommended by Monte Carlo methods (α=1)

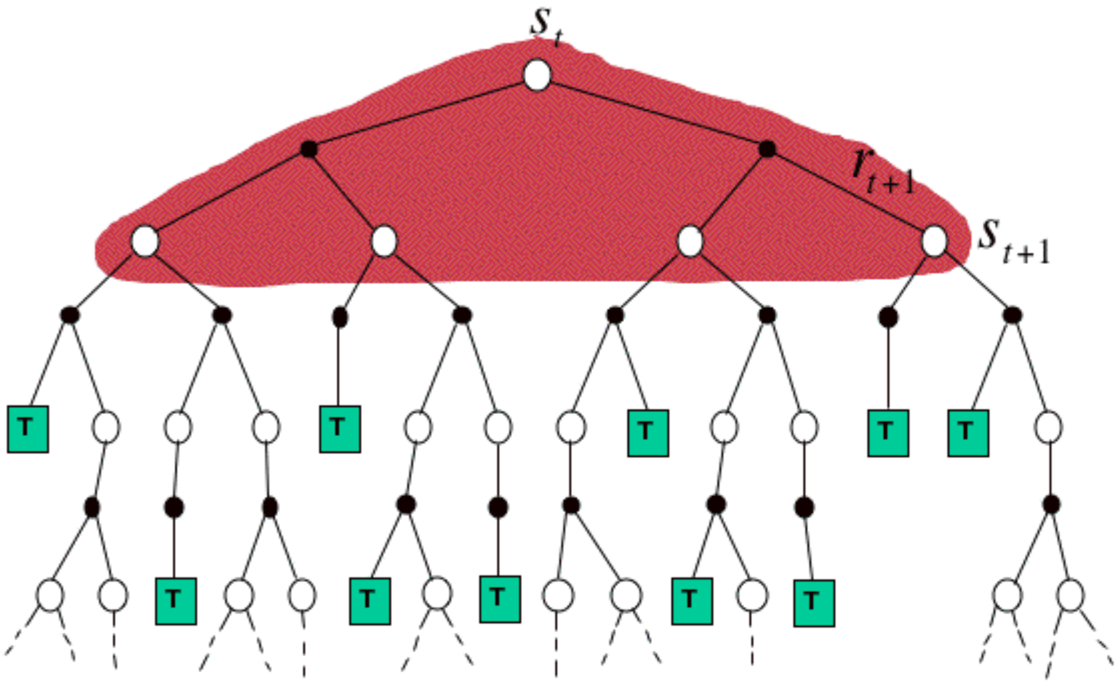


Changes recommended by TD methods (α=1)



What does DP Do?

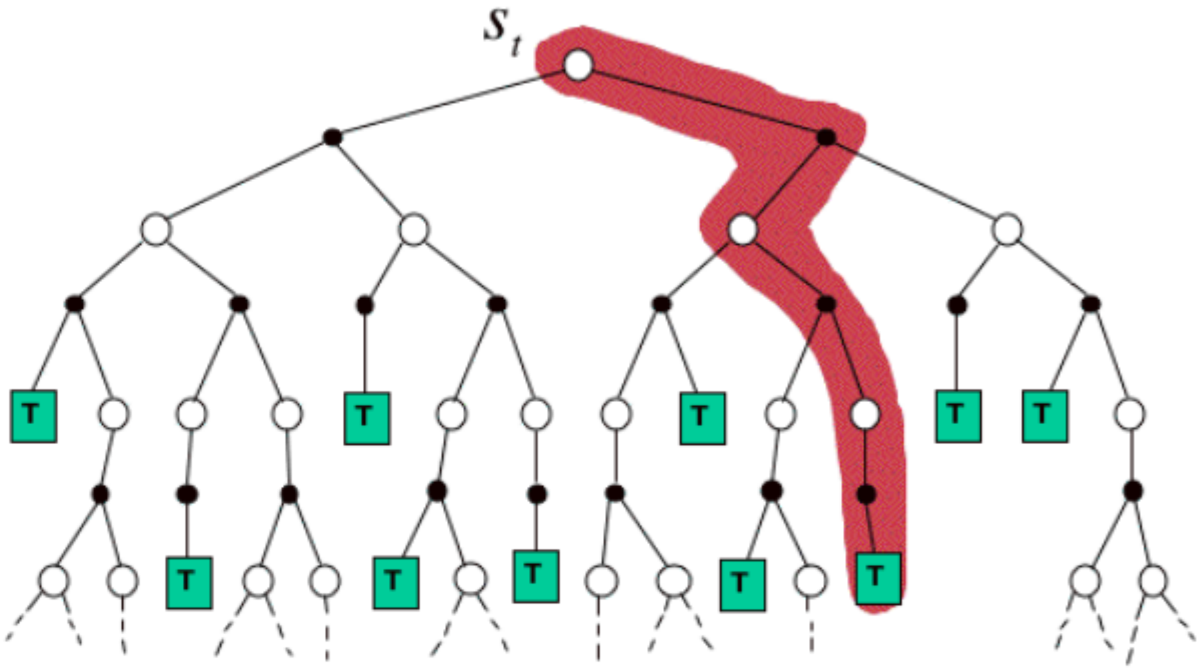
$$V(s_t) \leftarrow E_\pi\left\{r_{t+1} + \gamma\, V(s_t)\right\}$$



What does Simple MC Do?

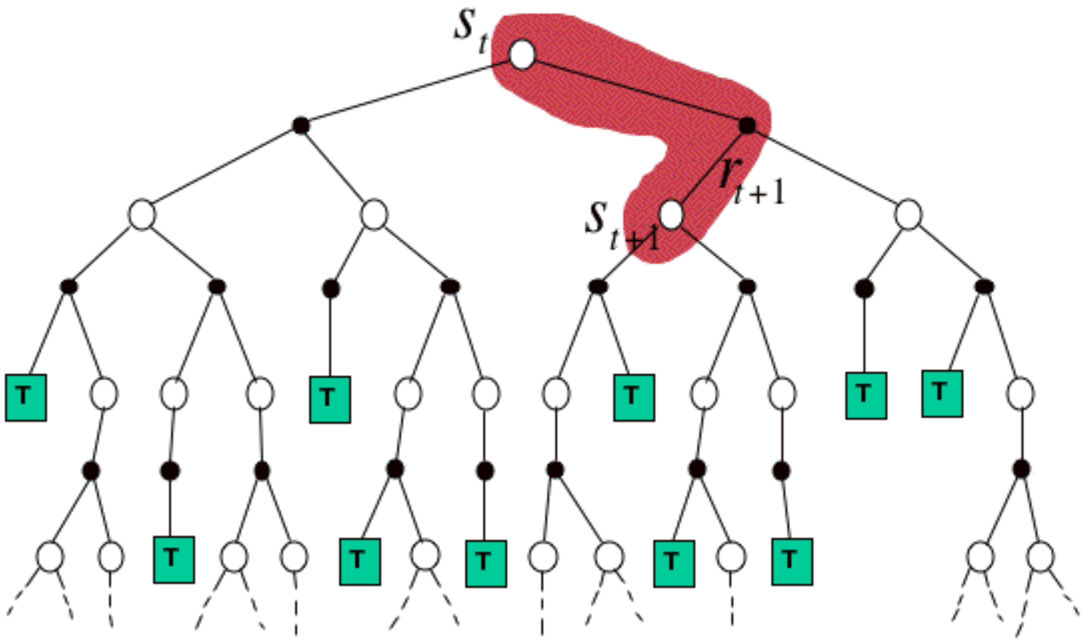
$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right]$$

where R_t is the actual return following state s_t.



Idea behind Temporal Difference Procedure

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$



Temporal Difference Prediction

Policy Evaluation is often referred to as the Prediction Problem: we are trying to predict how much return we'll get from being in state s and following policy π by learning the state-value function V^π. Compare:

Monte-Carlo update:

$$V(s_t) \rightarrow V(s_t) + \alpha[\underbrace{R_t}_{\text{Target: actual return from } s_t \text{ to end of episode}} - V(s_t)]$$

Simplest temporal difference update TD(0):

$$V(s_t) \rightarrow V(s_t) + \alpha[\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{Target: estimate of the return}} - V(s_t)]$$

Both have the same form

Temporal Difference Learning

- *Does not* require a model (i.e., transition and reward prob.) – learn directly from experience
- Update estimate of $V(s)$ soon after visiting the state s

$$\Delta V_t(s_t) = \alpha[r_{t+1} + \gamma V_{t+1}(s_{t+1}) - V_t(s_t)]$$

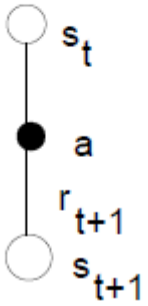
$V_{t+1}(s_t) - V_t(s_t)$

Actual 1-step reward

Discounted estimate of future reward

Initial estimate of future reward

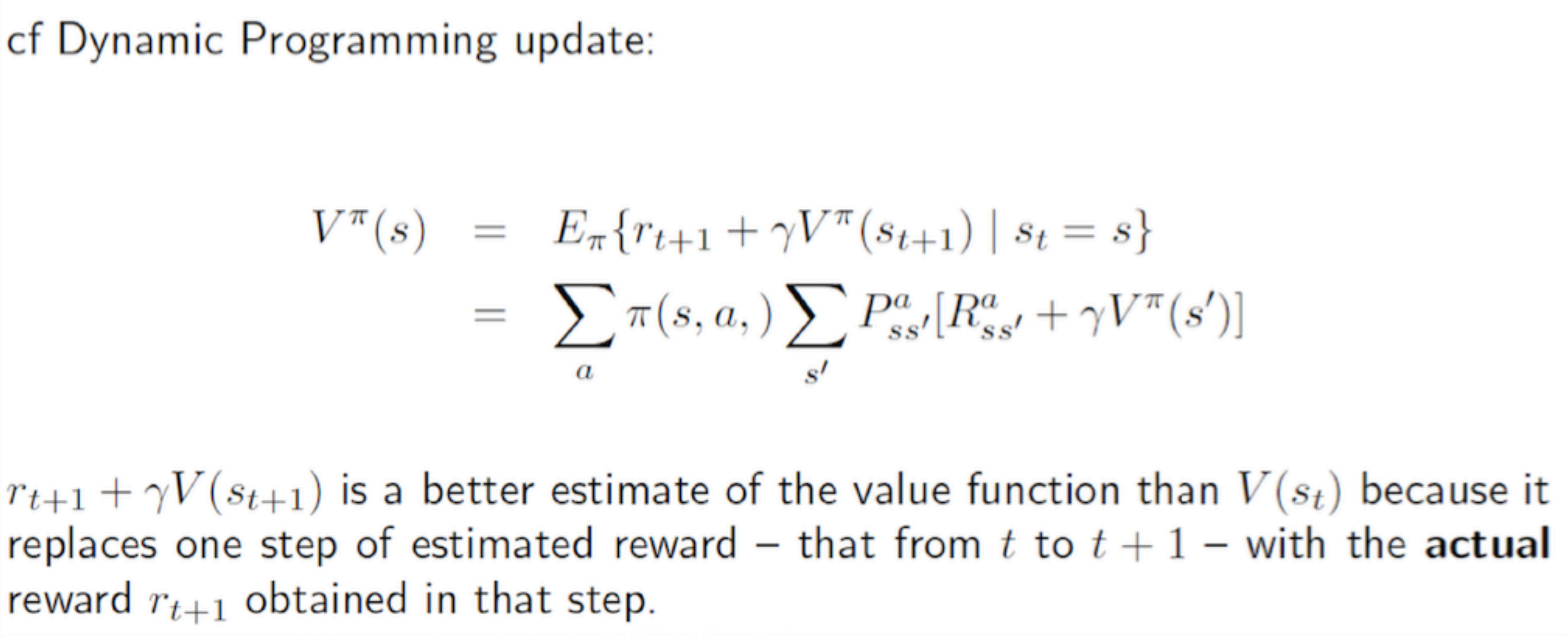
This is better than .... this



Backup diagram

TD(0) Update

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$



cf Dynamic Programming update:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\} \\ &= \sum_{a} \pi(s,a,) \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{\pi}(s')] \end{aligned}$$

$r_{t+1} + \gamma V(s_{t+1})$ is a better estimate of the value function than $V(s_t)$ because it replaces one step of estimated reward – that from t to $t+1$ – with the **actual** reward r_{t+1} obtained in that step.

TD(0) Algorithm for Learning V^{π}

```
• Initialise $V(s)$ arbitrarily; $\pi$ is the policy to be evaluated; choose learning rate $\alpha$ and discount factor $\gamma$
• Repeat for each episode
      Pick a start state $s$
      Repeat for each step in episode
          Get action $a$ given by policy $\pi$ for state $s$
          Take action $a$, observe reward $r$ and next state $s'$
          $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
          $s \leftarrow s'$
      until $s$ is terminal
```

From S+B Fig. 6.1

Why TD Learning?

- Don't need a model of the environment
- On-line and incremental – updates each step – so can be fast

  don't need to wait till the end of the episode so need less memory, computation

  subsequent updates take immediate advantage of updated values

  cf. Monte Carlo – waits till end of episode, episodes may be long or tasks continuing, some MC must ignore episodes with exploratory steps
- Updates are based on actual experience (r_{t+1})
- Converges to $V^\pi(s)$ – but must decrease step size α as learning continues

Why?

Bootstrapping, Sampling

TD **bootstraps**: it updates its estimates of V based on other estimates of V

DP also bootstraps

MC does not bootstrap: estimates of complete returns are made at the end of the episode

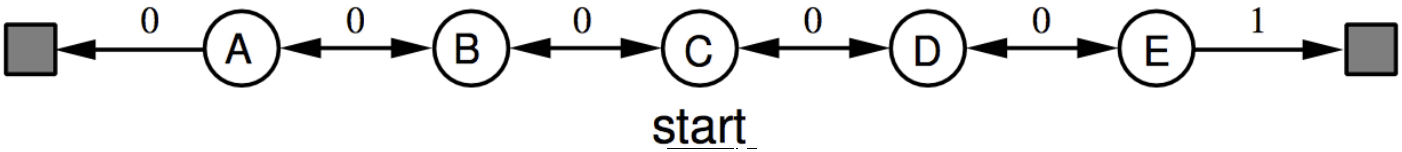
TD **samples**: its updates are based on one path through the state space

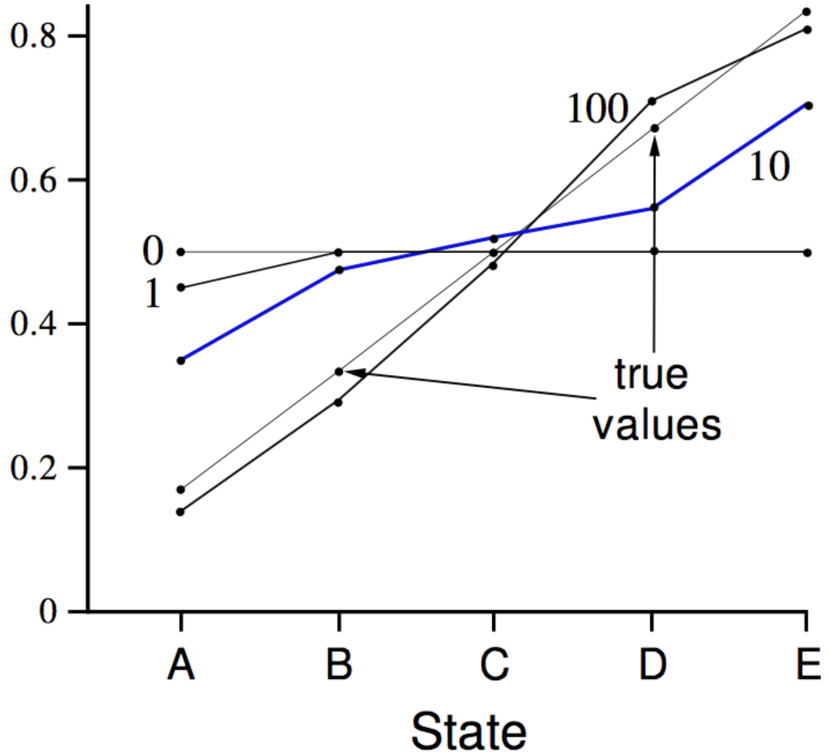
MC also samples

DP does not sample: its updates are based on all actions and all states that can be reached from the updating state

Examples: see e.g. random walk example S+B sect. 6.2

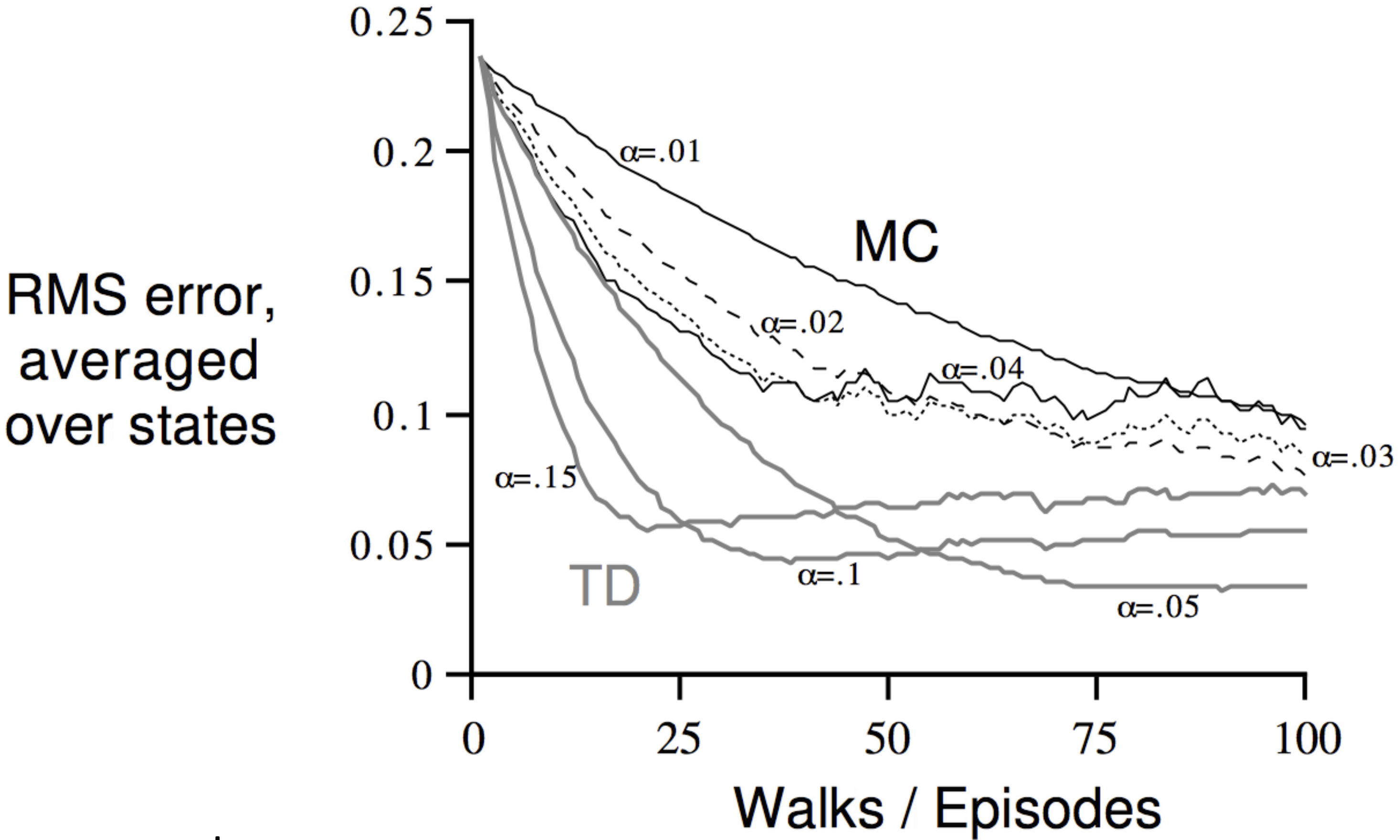
Random Walk Example





Values learned by TD(0) after various numbers of episodes

TD and MC on the Random Walk



Data averaged over
100 sequences of episodes

Understanding TD vs. MC

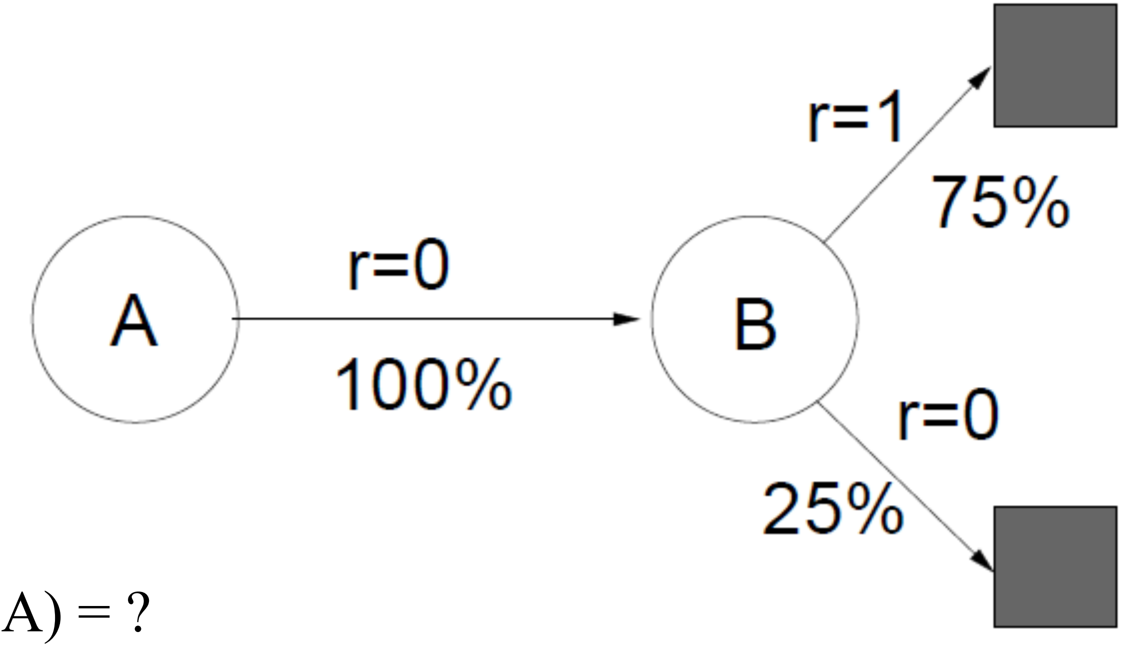
S+B Example 6.4:

- You observe 8 episodes of a process:

  A,0,B,0 B,1 B,1 B,1 B,1 B,1 B,1 B,0
- Interpretation:
  - First episode starts in state A, transitions to B getting a reward of 0, and terminates with a reward of 0
  - Second episode starts in state B, terminates with a reward of 1, etc.

Question: What are good estimates for $V(A)$ and $V(B)$?

S+B Example 6.4: Underlying Markov Process



$V(A) = ?$

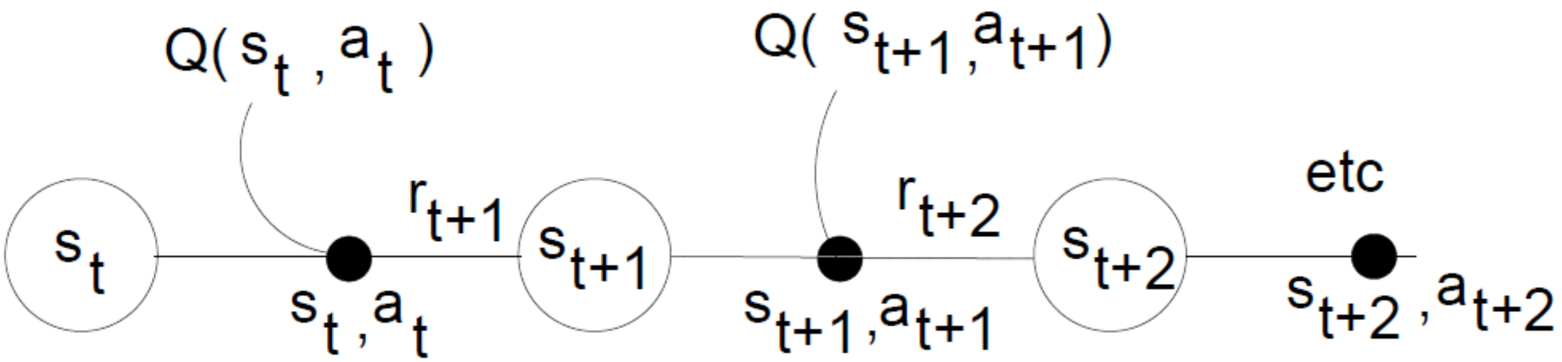
TD and MC Estimated

- Batch Monte Carlo (update after all episodes done) gets V(A) = 0.
  - This best matches the training data
  - It minimises the mean-square error on the training set
- Consider sequentiality: A to B to terminating state; V(A) = 0.75.
  - This is what TD(0) gets
  - Expect that this will produce better estimate of future data even though MC gives the best estimate on the present data
  - Is correct for the maximum-likelihood estimate of the model of the Markov process that generates the data, i.e. the best-fit Markov model based on the observed transitions
  - Assume this model is correct; estimate the value function – “certainty-equivalence estimate”

TD(0) tends to converge faster because it moves towards a *better* estimate.

TD for *Control*: Learning Q-Values

Learn action values $Q^{\pi}(s, a)$ for the policy π



SARSA update rule:

$$\Delta Q_t(s_t, a_t) = \alpha[r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]$$

TD for Control: Learning Q-Values

- Choose a behaviour policy π and estimate the Q-values (Q^π) using the SARSA update rule. Change π towards greediness wrt Q^π.
- Use ϵ-greedy or ϵ-soft policies.
- Converges with probability 1 to optimal policy and Q-values if visit all state-action pairs infinitely many times and policy converges to greedy policy, e.g. by arranging for ϵ to tend towards 0.

Remember: learning optimal Q-values is useful since it tells us immediately which is(are) the optimal action(s) – they have the highest Q-value

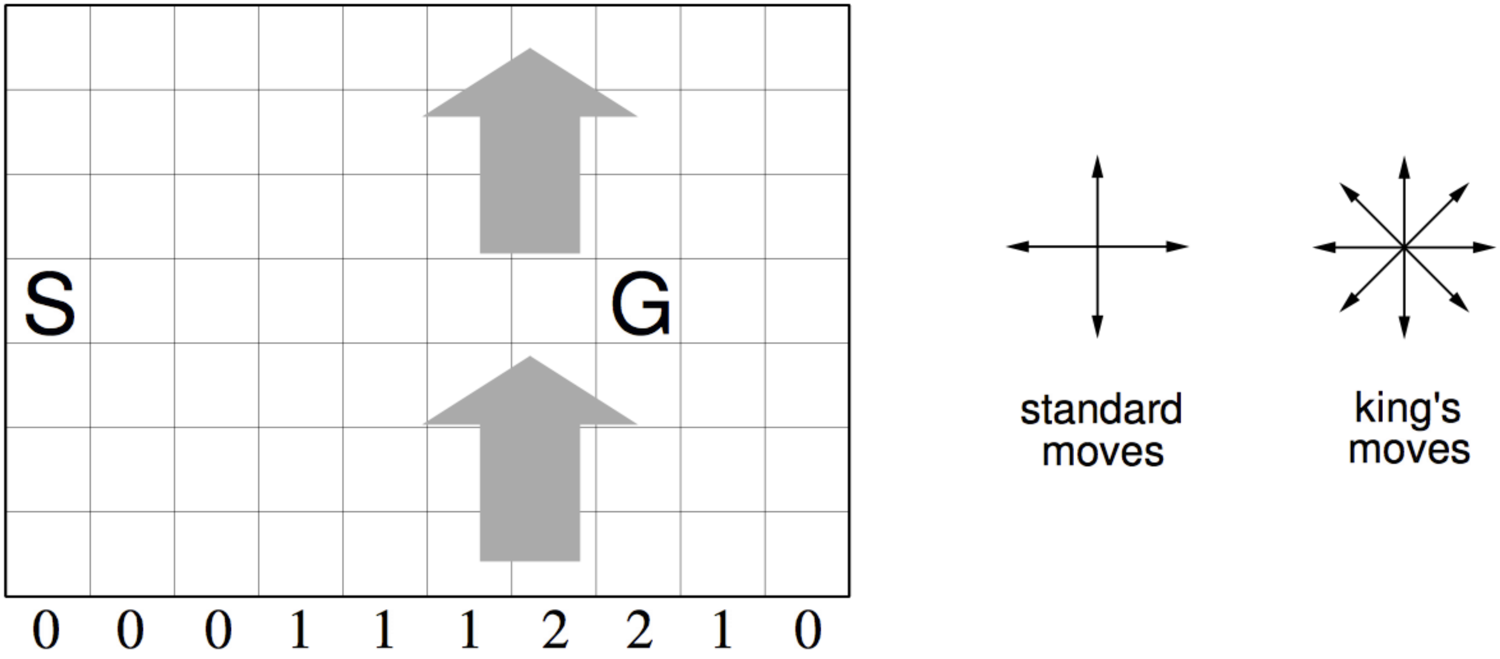
Algorithm: SARSA

- Initialise $Q(s, a)$
- Repeat `many times`
  - Pick s, a
  - Repeat `each step to goal`
    - Do a, observe r, s'
    - Choose a' based on $Q(s', a')$ ϵ`-greedy`
    - $Q(s, a) = Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    - $s = s'$, $a = a'$
  - Until s terminal (where $Q(s', a') = 0$)

Use with policy iteration, i.e. change policy each time to be greedy wrt current estimate of Q

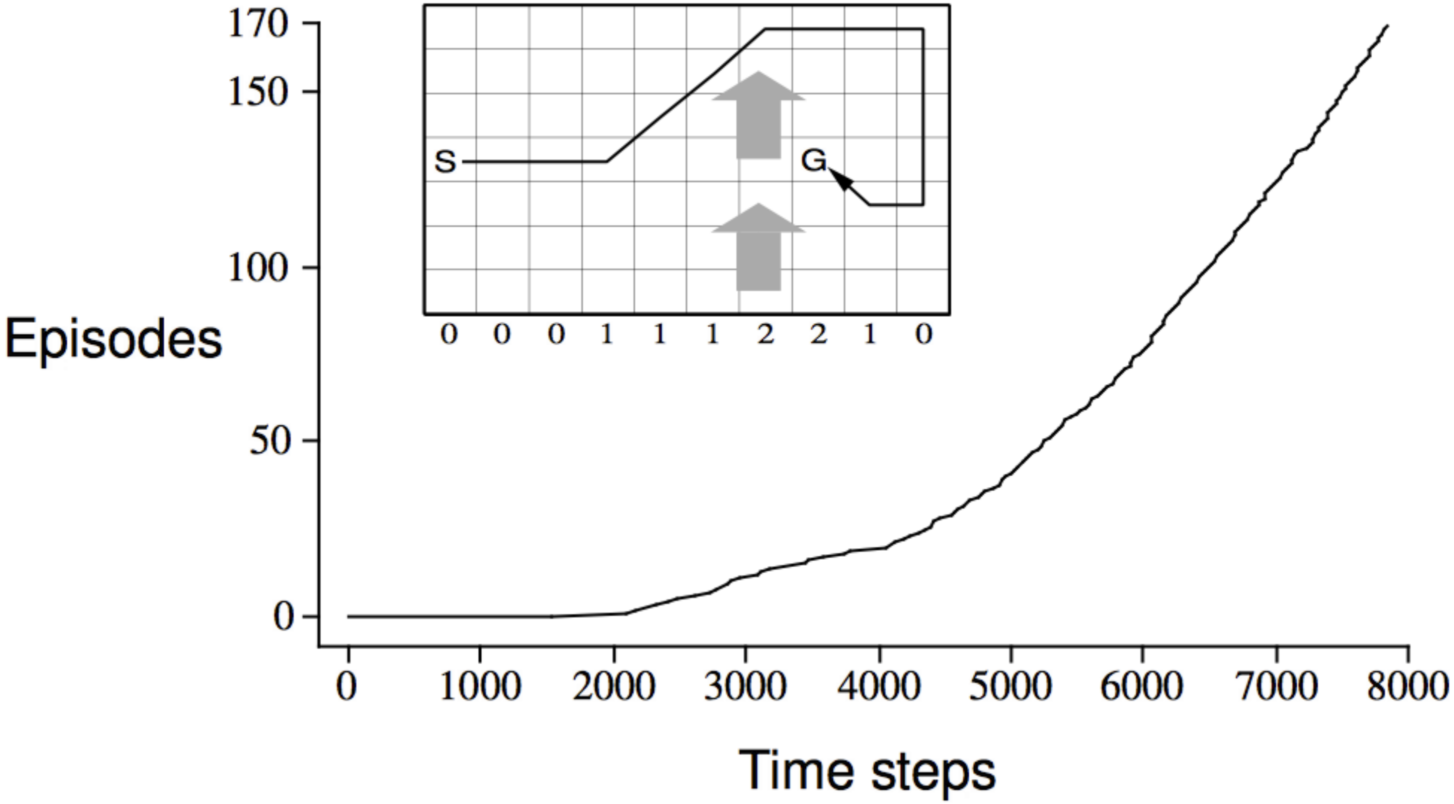
Example: windy gridworld, S+B sect. 6.4

Windy Gridworld



undiscounted, episodic, reward = –1 until goal

Results of Sarsa on the Windy Gridworld



Q-Learning

SARSA is an example of **on-policy** learning. Why?

Q-LEARNING is an example of **off-policy** learning

Update rule:

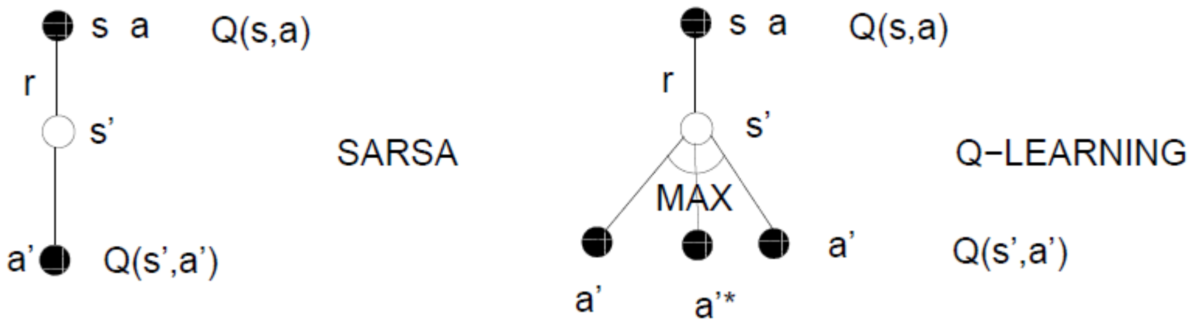
$$\Delta Q_t(s_t, a_t) \quad = \quad \alpha[r_{t+1} + \boxed{\gamma \max_a Q_t(s_{t+1}, a)} - Q_t(s_t, a_t)]$$

Always update using *maximum* Q value available from next state: then $Q \Rightarrow Q*$, optimal action-value function

Algorithm: Q-Learning

```
• Initialise $Q(s, a)$
• Repeat        many times
    – Pick $s$        start state
    – Repeat        each step to goal
        ∗ Choose $a$ based on $Q(s, a)$        $\epsilon$-greedy
        ∗ Do $a$, observe $r$, $s'$
        ∗ $Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        ∗ $s = s'$
    – Until $s$ terminal
```

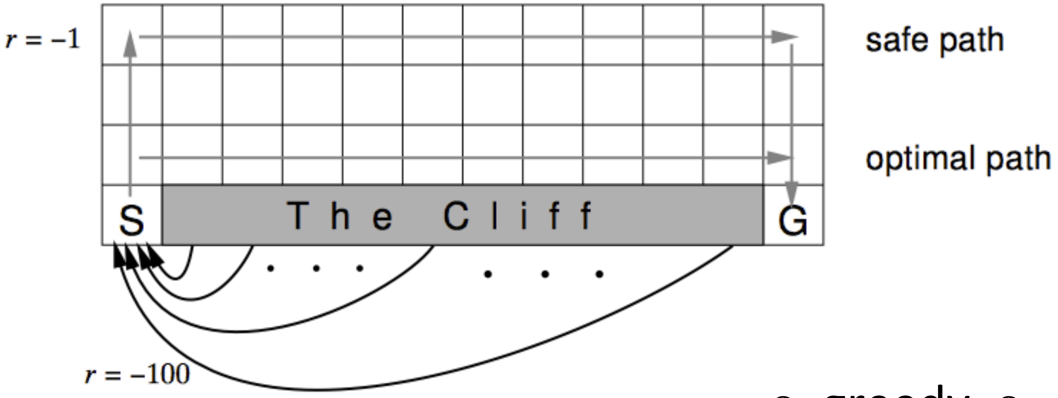
Backup Diagrams: SARSA and Q-Learning



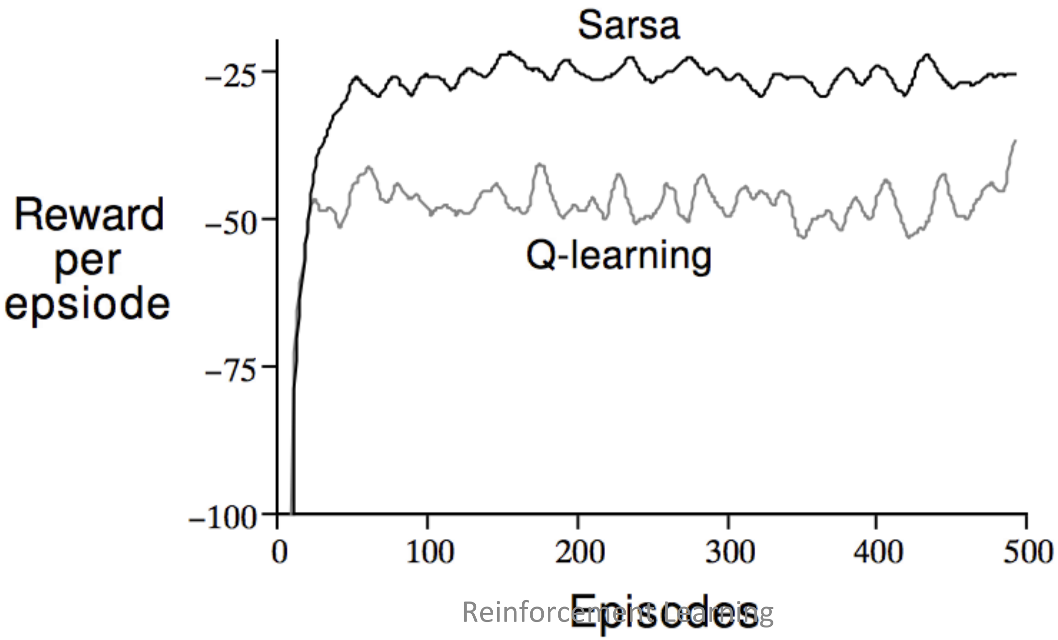
SARSA backs up using the action a' actually chosen by the behaviour policy.

Q-LEARNING backs up using the Q-value of the action a'^* that is the *best* next action, i.e. the one with the highest Q value, $Q(s', a'^*)$. The action actually chosen by the behaviour policy *and followed* is not necessarily a'^*

Cliffwalking



ε−greedy, $\varepsilon = 0.1$



Q-Learning vs. SARSA

QL: $Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ off-policy

SARSA: $Q(s, a) = Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$ on-policy

In the cliff-walking task:

QL: learns optimal policy along edge

SARSA: learns a safe non-optimal policy away from edge

ϵ-greedy algorithm

For $\epsilon \neq 0$ **SARSA** performs better online. Why?

For $\epsilon \to 0$ gradually, both converge to optimal.

Summary

- Idea of Temporal Difference Prediction
- 1-step tabular model-free TD method
- Can extend to the GPI approach:
  - On-policy: SARSA
  - Off-policy: Q-learning
- TD methods bootstrap and sample, combining benefits of DP and MC methods