17. Dynamic Routing: MDP

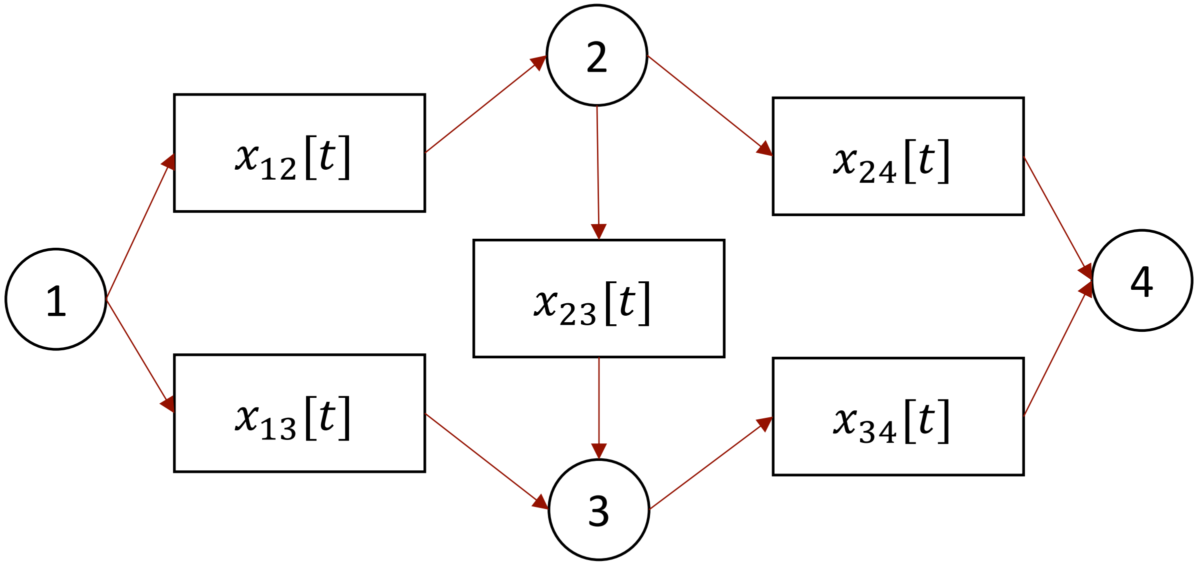
金力 Li Jin
li.jin@sjtu.edu.cn
上海交通大学密西根学院
Shanghai Jiao Tong University UM Joint Institute

Outline

- Modeling
  - Network dynamics
  - MDP formulation
- [Not required] Solution methods
  - Dynamic programming
  - Monte Carlo simulation
  - Temporal-difference

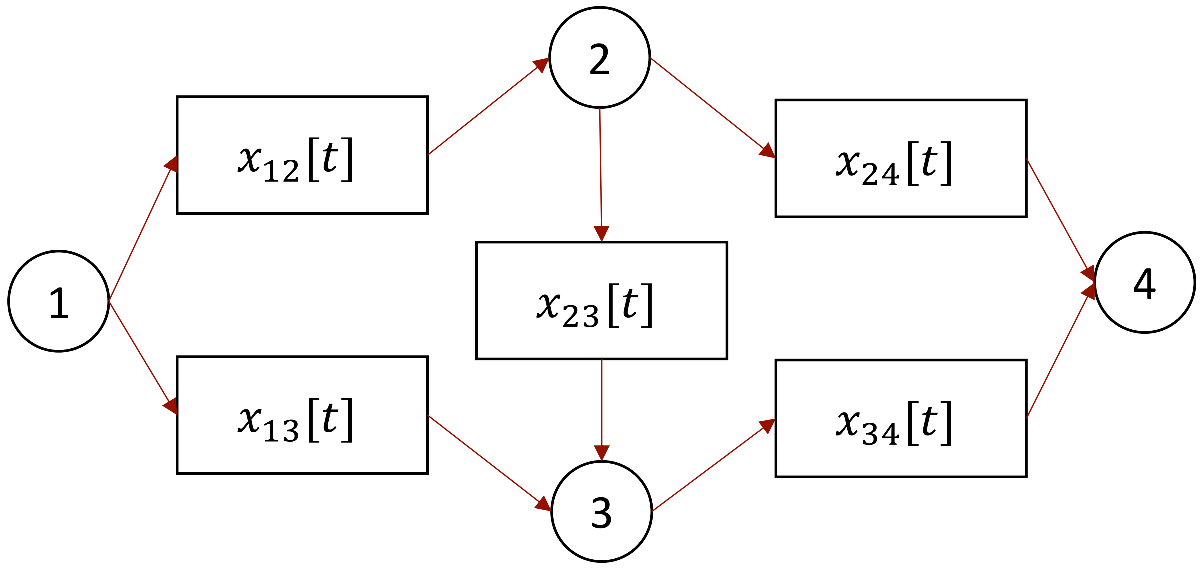
Setup

- Network $G(N, E)$
- Set of nodes $N = \{1,2,3,4\}$
- Set of links $E = \{12,13,23,24,34\}$
- Traffic state $x_{ij}[t], (i,j) \in E$
- State space $S = \{0,1,2\}^{|E|}$; $|S| = 3^5$



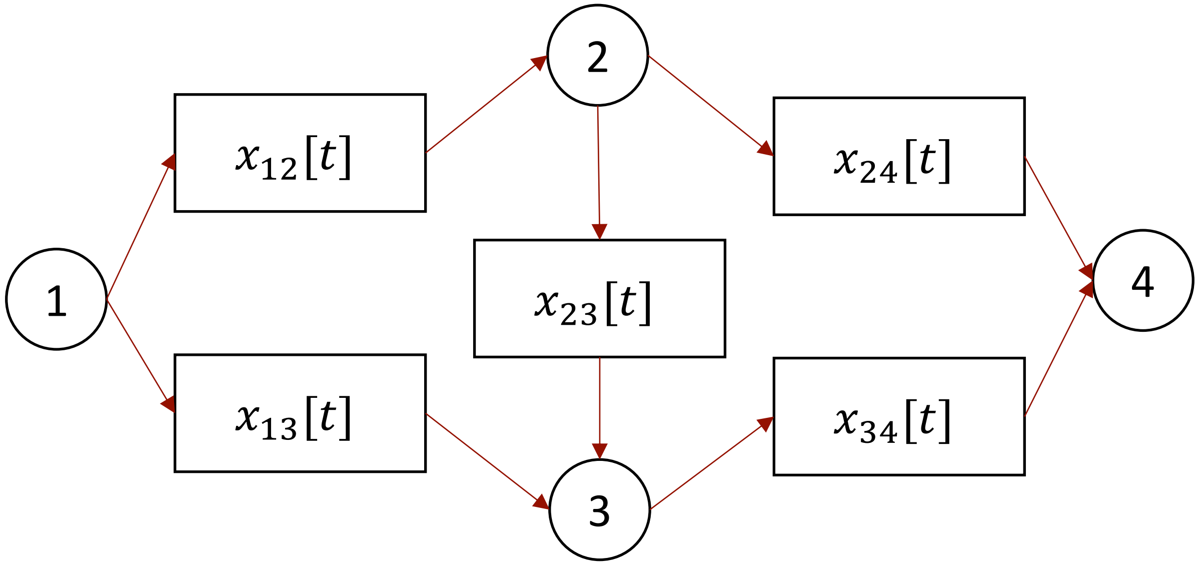
Arrival process

- At each time step, a vehicle arrives at node i with probability λ_i for $i = 1,2,3$.
- Probability of no arrivals = $1 - \lambda_i$.
- Every vehicle will go to node 4.
- If no available slot, vehicle is rejected.



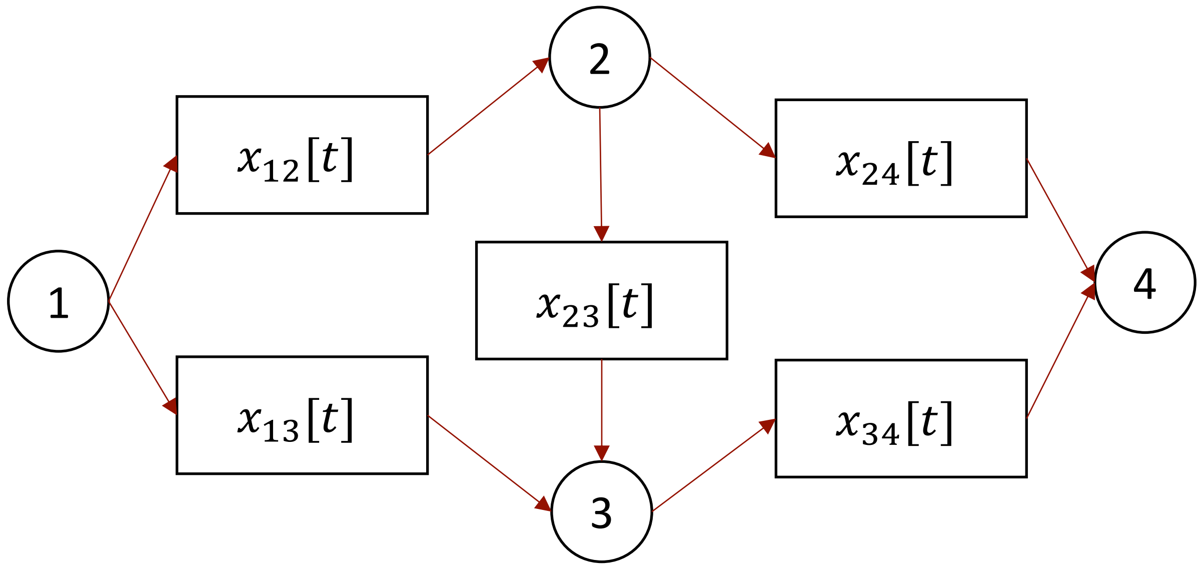
Departure process

- Every node can process no more than 1 vehicle per time step. (“Intersection”)
- Hence, a node has to select the upstream traffic to accommodate.
- A node also has to select the downstream link for the vehicle going through the node.



Action & action space

- Let $a_i[t]$ be the action for node i at time t.
- Node 1: $a_1[t] \in \{12,13\}$
- Node 2: $a_2[t] \in \{23,24\}$
- Node 3: $a_3[t] \in \{13,23\}$
- Node 4: $a_4[t] \in \{24,34\}$

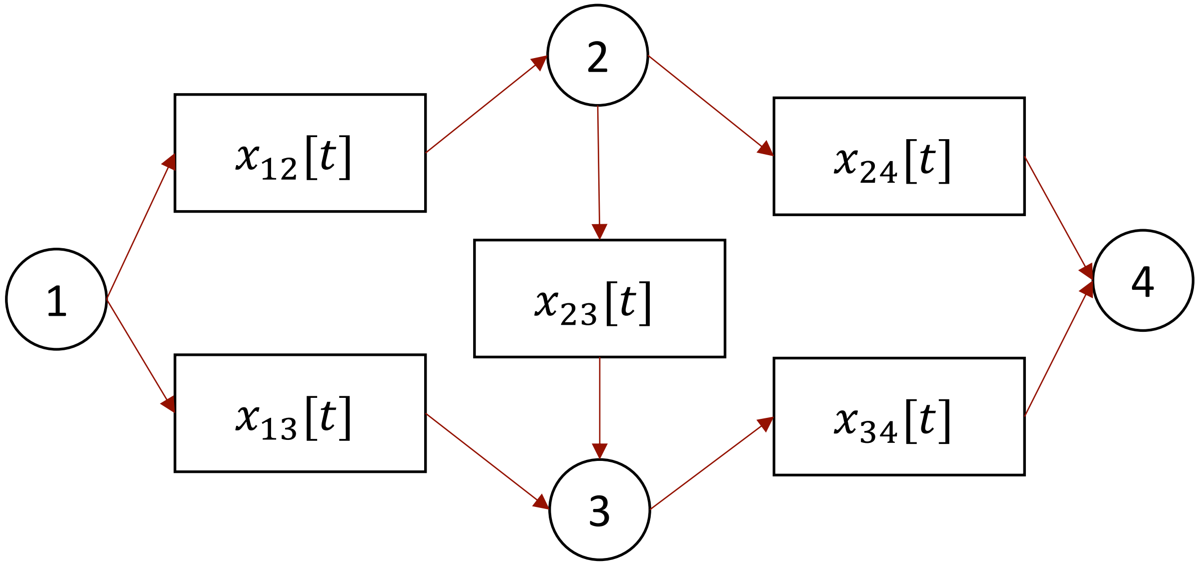


Transition/dynamics

Given $x[t]$, $a[t]$, what is $x[t+1]$?

- # of states: $3^5 = 243$
- # of actions: $2^4 = 16$
- # of state-action combinations:

$$243 \times 16 = ??$$



Transition matrix

- When we code the transition dynamics, we can use a big matrix to store the dynamics information.
- Consider a given policy, i.e., a function

$$\pi: \{0,1,2\}^5 \rightarrow \{12,13\}\times\{23,24\}\times\{13,23\}\times\{24,34\}$$

or $\pi: x \mapsto a$

- Matrix size: $P \in [0,1]^{273\times 273}$
- Note that $a = \pi(x)$
- Interpretation of elements:

$$p(x'|x,a) = P_{x,x'}.$$

Reward

- Two types of rewards/costs $r[t]$:

1. Rejecting a vehicle leads to a reward of $-c < 0$.
2. At time t, we collect a reward of $-\|x[t]\|_1$.

- In summary, we collect more rewards if

1. we let in more vehicles,
2. we make vehicles arrive at the destination as soon as possible

- Cumulative reward ("return") with discount

$$G[t] = \sum_{\tau=t}^{\infty} \gamma^{\tau} r[\tau], \qquad \gamma \in (0,1].$$

Outline

- Modeling
  - Network dynamics
  - MDP formulation
- [Not required] Solution methods
  - Dynamic programming
  - Monte Carlo simulation
  - Temporal-difference

Solution of MDPs

- For practical problems, analytical solution is in general very hard.
- Instead, we consider a class of approaches to decision making in incomplete-information and stochastic settings.
- Core concepts:
  - Dynamic programming: perfect prior knowledge of system dynamics (transition probabilities)
  - Monte Carlo Methods: learning expected rewards and returns through averaging experiences
  - Temporal-difference learning: combination of the above two
- Not required for quiz 2; maybe useful for project

Dynamic programming

- The term **dynamic programming (DP)** refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as an MDP.
- Classical DP algorithms are of limited utility in RL both because of their assumption of a perfect model and because of their great computational expense.
- But they are still important theoretically.
- In fact, all RL methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment.

Classical DP algorithm

- The key idea of DP, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies.
- Step 1: Policy evaluation (prediction).
  - Compute the state value function for π.
- Step 2: Policy improvement.
  - Use Bellman optimality equation to improve π to π'.
- Step 3: Policy iteration
  - Iteratively evaluate and improve the policy.
- We only consider finite MDPs.

Policy Evaluation (Prediction)

- Value function:

$$v_\pi(s) = \mathrm{E}\left[\sum_{\tau=t}^{\infty} \gamma^\tau r[\tau] \middle| x[t] = s\right], \qquad s \in S.$$

- **Objective**: for a given policy π, compute the state-value function v_π.
- We begin with an initial guess $\{v_0(s); s \in S\}$.
- Then, we iterate to obtain $\{v_1(s); s \in S\}$ via

$$v_1(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big(r + \gamma v_0(s')\big),$$

$$\forall s \in S.$$

Policy Evaluation (Prediction)

- Such an iteration is called a "sweep", since we update $v_0(s)$ to $v_1(s)$ for all s.
- We can keep doing this via

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big(r + \gamma v_k(s')\big), s \in \mathcal{S}.$$

$$\begin{tikzcd} v_0 \arrow[r] & v_1 \arrow[r] & \cdots \arrow[r] & v_k \arrow[r] & v_{k+1} \arrow[r] & \cdots \arrow[r] & v_\pi \end{tikzcd}$$

a "sweep"

- Iteration terminates when the increment is small, i.e. when $\max_s \|v_{k+1}(s) - v_k(s)\| \leq \theta$, where θ is a threshold.
- This is called a full policy-evaluation backup.

Policy Evaluation (Prediction)

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

```
Input $\pi$, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
    $\Delta \leftarrow 0$
    Loop for each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$
```

Policy Evaluation (Prediction)

- For dynamic routing problem, you need to store v_k as a 3^5-dimensional vector.
- For the kth iteration, you need to update al the 3^5 elements of the vector to obtain v_{k+1}.

$$v_{k+1}(s) = \sum_{s'} p(s'|s,a)\big(r(s,a) + \gamma v_k(s')\big).$$

- Keep iterating until v_k converges...

Policy Improvement

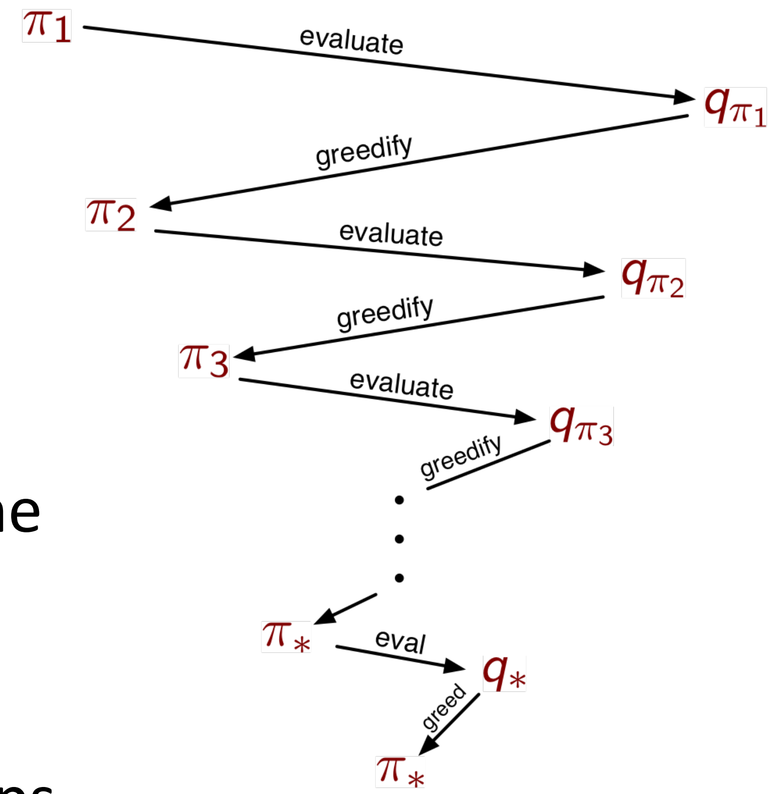
- **Objective**: Given a policy π and its evaluation $v_\pi(s)$ (and $q_\pi(s,a)$), find a policy π' that is better than π.
- This is done by greedifying with respect to $v_\pi(s)$:

$$\pi'(s) = \arg\max_a q_\pi(s,a), \qquad \forall s.$$

- Then, we have improved π to π'.
- What if the policy is unchanged by this? Then the policy must be optimal!
- The above approach, as well as the policy improvement theorem, can be extended to stochastic policies easily.

Policy Iteration

- π_k evaluates to a unique value function (soon we will see how to learn it).
- π_k can be greedified to produce a better policy π_{k+1}.
- That in turn evaluates to a value function which can in turn be greedified.
- Each policy is strictly better than the previous, until eventually both are optimal.
- There are no local optima, and π_k converges in a finite number of steps, usually very few.



Efficiency of DP

- To find an optimal policy is polynomial in the number of states…
- BUT, the number of states is often astronomical, e.g., often growing exponentially with the number of state variables (what Bellman called “the curse of dimensionality”).
- In practice, classical DP can be applied to problems with a few millions of states.
- Asynchronous DP can be applied to larger problems, and is appropriate for parallel computation.
- It is surprisingly easy to come up with MDPs for which DP methods are not practical.

Monte Carlo methods

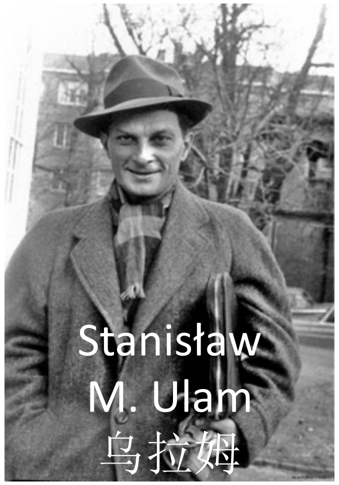
In a general context:

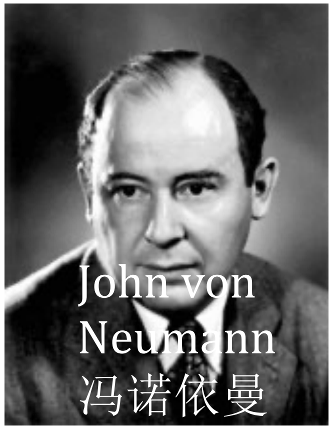
- **Monte Carlo methods**, or Monte Carlo experiments, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results.
- The underlying concept is to use randomness to solve problems that might be deterministic in principle.

Monte Carlo Casino, Monaco

Monte Carlo methods

- Developed by Stanisław Ulam during the US’s nuclear weapon project
- Recognized by von Neumann
- Being secret, the work of von Neumann and Ulam required a code name.
- A colleague of von Neumann and Ulam, Nicholas Metropolis, suggested using the name Monte Carlo, which refers to the Monte Carlo Casino in Monaco where Ulam's uncle would borrow money from relatives to gamble.





Monte Carlo methods

- **Monte Carlo (MC) method in reinforcement learning**
- MC methods learn directly from episodes of experiences.
- MC is model-free: no knowledge of MDP transitions $p(s'|s,a)$ or rewards r required.
- MC learns from complete episodes: no bootstrapping.
- MC uses the simplest possible idea: value = mean return.
- Caveat: can only apply MC to episodic MDPs: All episodes must terminate.
- Requires strong computation & storage power.

Monte-Carlo Policy Evaluation

- Goal: learn v_π from episodes of experience under policy π
$$S_1, A_1, R_2, \dots, S_k \sim \pi.$$
- Recall that the return is the total discounted reward
$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T.$$
- Recall that the value function is the expected return
$$v_\pi(s) = \mathrm{E}[G_t | S_t = s].$$
- MC method uses empirical mean return instead of expected return.
- Recall SLLN: empirical mean → expectation a.s.
$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} X_k = \bar{X}. \text{ a.s.}$$

First-Visit Monte-Carlo Policy Evaluation

- Objective: evaluate (i.e., compute the value function of) state s
- We begin our MC sampling upon the first time-step t that state s is visited in an episode.
- Increment counter $N(s) \leftarrow N(s) + 1$.
- Increment total return (at the end of an episode) $S(s) \leftarrow S(s) + G_t$.
- G_t is estimated until the end of episode.
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- BY SLLN, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$ a.s.

Monte-Carlo for dynamic routing

- Let’s go back to dynamic routing.
- Consider a given routing policy $\pi: S \rightarrow A$.
- We first write codes that simulates evolution of the network under the given policy.
- For each state $s \in S$, we set s as the initial condition and simulate the network for sufficiently many time steps.
- This gives us a sample of return $\sum_{\tau=t}^{\infty} \gamma^{-\tau} r[\tau]$.
- Then, we approximate $v_\pi(s)$ with $\sum_{\tau=t}^{\infty} \gamma^{-\tau} r[\tau]$.
- You can also take average over multiple samples.

TD learning

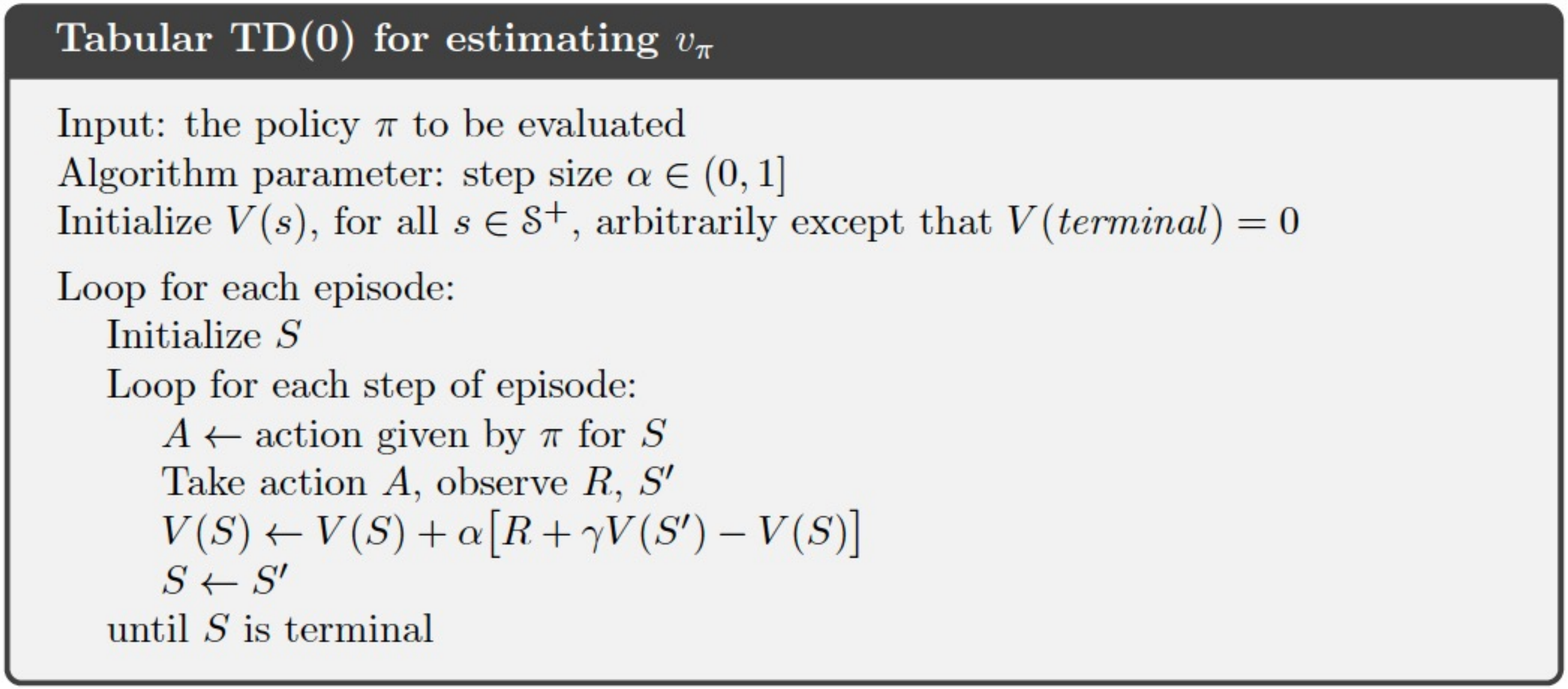
If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning.

- TD methods learn directly from episodes of experience
- TD is model-free: no knowledge of MDP transitions/rewards
- TD learns from incomplete episodes by bootstrapping
- TD updates a guess towards a guess

Temporal-Difference Learning

- Consider the prediction problem: estimate (or learn) v_π online from experience under policy π.
- In MC, we have
$$V^{new}(S_t) \leftarrow V^{old}(S_t) + \alpha\left(G_t - V^{old}(S_t)\right),$$
where G_t is from experiment.
- In TD, we consider replace G_t with $R_{t+1} + V^{old}(S_{t+1})$:
$$V^{new}(S_t) \leftarrow V^{old}(S_t) + \alpha\left(R_{t+1} + V^{old}(S_{t+1}) - V^{old}(S_t)\right).$$
- In case of discounted problems, we have
$$V^{new}(S_t) \leftarrow V^{old}(S_t) + \alpha\left(R_{t+1} + \gamma V^{old}(S_{t+1}) - V^{old}(S_t)\right).$$

TD prediction (policy evaluation)



Tabular TD(0) for estimating v_π

```
Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Because TD bases its update in part on an existing estimate, we say that it is a **bootstrapping** method, like DP.

Dynamic routing

- Start with some initial guess $V_0(s)$ for all s.
- At each time t, update $V(x[t])$ as follows:

$$V^{new}(x[t]) \leftarrow V^{old}(x[t]) + \alpha\left(r(x[t],a[t]) + \gamma V^{old}(x[t+1]) - V^{old}(x[t])\right).$$

- Note that such updates happen at every time step.
- After you simulate $x[t]$ for sufficiently long time, the values for all states should be updated.
- Finally, you can improve the policy as we did in DP.

For vehicle control (speed tracking)

- State: $s[t]$ = speed.
- Action: $a[t]$ = engine torque.
- Dynamics: $s[t+1] = f(s[t], a[t])\delta + w[t]$.
- δ = step size, $w[t]$ = noise.
- To fit the MDP framework, you can discretize the values for $s[t], a[t], w[t]$.
- For example, $s[t] \in \{0,1,2, \dots, 10\}; a[t] \in \{0,1,2\}; w[t] \in \{-1,0,1\}$.
- You can further assume a probability distribution for $w[t]$.
- Then, you can write the transition probability $p(s'|s,a)$.

Setup for learning-based CACC

- First, set up a vehicle model for simulation.
- For example, you can use the model you obtained in MP1.
- Then, pretend that you do not know the model.
- Discretize the state and action space and obtain the MDP formulation.
- Apply TD method to obtain an optimal control policy.