A Guided Tour of Chapter 1: Markov Process and Markov Reward Process

Ashwin Rao

ICME, Stanford University

Intuition on the concepts of *Process* and *State*

- *Process:* time-sequenced random outcomes
- Random outcome eg: price of a derivative, portfolio value etc.
- *State:* Internal Representation S_t driving future evolution
- We are interested in $\mathbb{P}[S_{t+1}|S_t, S_{t-1}, \ldots, S_0]$
- Let us consider random walks of stock prices $X_t = S_t$

$$\mathbb{P}[X_{t+1} = X_t + 1] + \mathbb{P}[X_{t+1} = X_t - 1] = 1$$

- We consider 3 examples of such processes

Markov Property - Stock Price Random Walk Process

- Process is pulled towards level L with strength parameter α

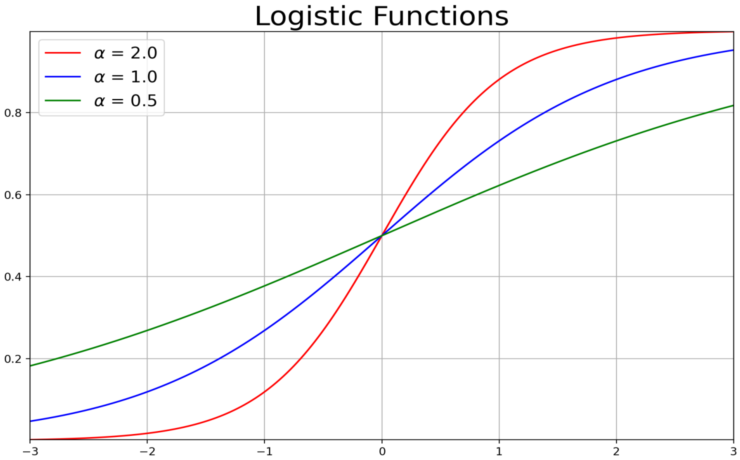
$$\mathbb{P}[X_{t+1} = X_t + 1] = \frac{1}{1 + e^{-\alpha_1 (L - X_t)}}$$

- Notice how the probability of next price depends only on current price
- "The future is independent of the past given the present"

$$\mathbb{P}[X_{t+1} | X_t, X_{t-1}, \ldots, X_0] = \mathbb{P}[X_{t+1} | X_t] \text{ for all } t \geq 0$$

- This makes the mathematics easier and the computation tractable
- We call this the *Markov Property* of States
- The state captures all relevant information from history
- Once the state is known, the history may be thrown away
- The state is a sufficient statistic of the future

Logistic Functions $f(x; \alpha) = \frac{1}{1+e^{-\alpha x}}$



Another Stock Price Random Walk Process

$$\mathbb{P}[X_{t+1} = X_t + 1] = \begin{cases} 0.5(1 - \alpha_2(X_t - X_{t-1})) & \text{if } t > 0 \\ 0.5 & \text{if } t = 0 \end{cases}$$

- Direction of $X_{t+1} - X_t$ is biased in the reverse direction of $X_t - X_{t-1}$
- Extent of the bias is controlled by "pull-strength" parameter α_2
- $S_t = X_t$ doesn't satisfy Markov Property, $S_t = (X_t, X_t - X_{t-1})$ does

$$\mathbb{P}[(X_{t+1}, X_{t+1} - X_t) | (X_t, X_t - X_{t-1}), (X_{t-1}, X_{t-1} - X_{t-2}), \ldots, (X_0, Null)]$$

$$= \mathbb{P}[(X_{t+1}, X_{t+1} - X_t) | (X_t, X_t - X_{t-1})]$$

- $S_t = (X_0, X_1, \ldots, X_t)$ or $S_t = (X_t, X_{t-1})$ also satisfy Markov Property
- But we seek the "simplest/minimal" representation for Markov State

Yet Another Stock Price Random Walk Process

- Here, probability of next move depends on *all* past moves
- Depends on # past up-moves U_t relative to # past down-moves D_t

$$\mathbb{P}[X_{t+1} = X_t + 1] = \begin{cases} \frac{1}{1+(\frac{U_t+D_t}{D_t}-1)^{\alpha_3}} & \text{if } t > 0 \\ 0.5 & \text{if } t = 0 \end{cases}$$

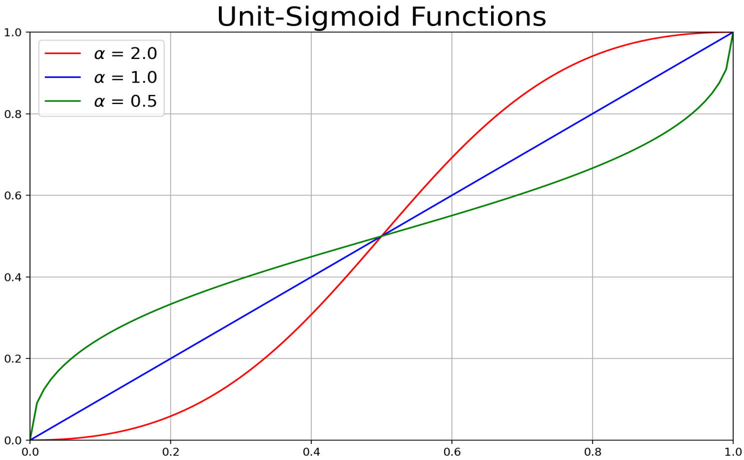
- Direction of $X_{t+1} - X_t$ biased in the reverse direction of history
- α_3 is a "pull-strength" parameter
- Most "compact" Markov State $S_t = (U_t, D_t)$

$$\mathbb{P}[(U_{t+1}, D_{t+1}) | (U_t, D_t), (U_{t-1}, D_{t-1}), \ldots, (U_0, D_0)]$$

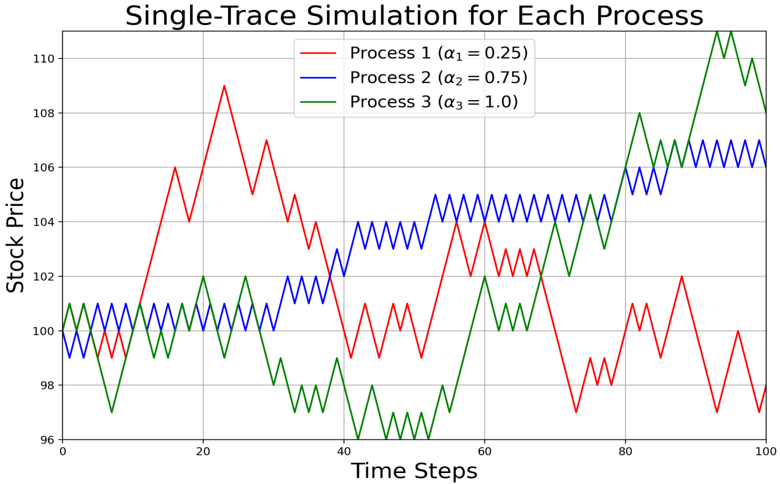
$$= \mathbb{P}[(U_{t+1}, D_{t+1}) | (U_t, D_t)]$$

- Note that X_t is not part of S_t since $X_t = X_0 + U_t - D_t$

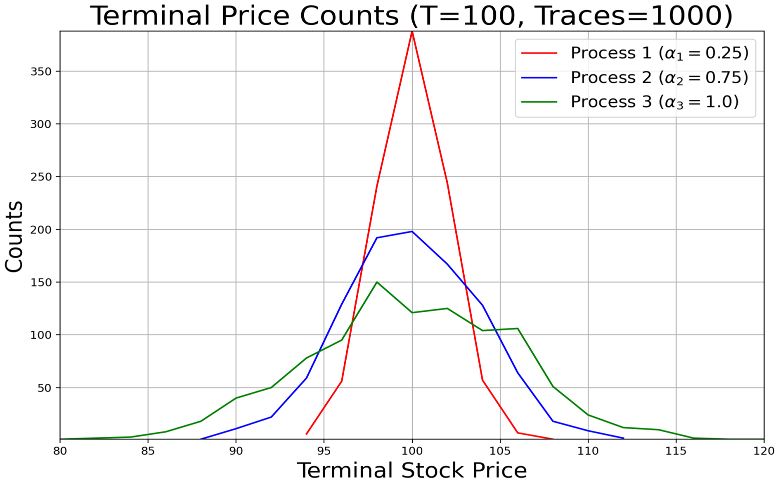
Unit-Sigmoid Curves $f(x;\alpha) = \frac{1}{1+(\frac{1}{x}-1)^{\alpha}}$



Single Sampling Traces for the 3 Processes



Terminal Probability Distributions for the 3 Processes



Definition for Discrete Time, Countable States

Definition

A *Markov Process* consists of:

- A countable set of states $\mathcal{S}$ (known as the State Space) and a set $\mathcal{T} \subseteq \mathcal{S}$ (known as the set of Terminal States)
- A time-indexed sequence of random states $S_t \in \mathcal{S}$ for time steps $t = 0, 1, 2, \ldots$ with each state transition satisfying the Markov Property: $\mathbb{P}[S_{t+1} | S_t, S_{t-1}, \ldots, S_0] = \mathbb{P}[S_{t+1} | S_t]$ for all $t \geq 0$.
- Termination: If an outcome for S_T (for some time step T) is a state in the set $\mathcal{T}$, then this sequence outcome terminates at time step T.

- The more commonly used term for *Markov Process* is *Markov Chain*
- We refer to $\mathbb{P}[S_{t+1} | S_t]$ as the transition probabilities for time t.
- Non-terminal states: $\mathcal{N} = \mathcal{S} - \mathcal{T}$
- Classical Finance results based on continuous-time Markov Processes

Some nuances of Markov Processes

- Stationary Markov Process: $\mathbb{P}[S_{t+1}|S_t]$ independent of t
- Stationary Markov Process specified with function $\mathcal{P} : \mathcal{N} \times \mathcal{S} \to [0,1]$

$$\mathcal{P}(s, s') = \mathbb{P}[S_{t+1} = s' | S_t = s] \text{ for all } s \in \mathcal{N}, s' \in \mathcal{S}$$

- $\mathcal{P}$ is the *Transition Probability Function* (source $s \to$ destination s')
- Convert non-Stationary to Stationary by augmenting *State* with time
- Default: *Discrete-Time, Countable-States Stationary Markov Process*
- Termination typically modeled with *Absorbing States* (we don't!)
- Separation between:
  - Specification of Transition Probability Function $\mathcal{P}$
  - Specification of Probability Distribution of Start States $\mu : \mathcal{S} \to [0,1]$
- Together ($\mathcal{P}$ and μ), we can produce *Sampling Traces*
- *Episodic* versus *Continuing* Sampling Traces

The @abstractclass MarkovProcess

```python
class MarkovProcess(ABC, Generic[S]):

    @abstractmethod
    def transition(self, state: S) -> \
            Optional[Distribution[S]]:
        pass

    def is_terminal(self, state: S) -> bool:
        return self.transition(state) is None

    def simulate(
        self,
        start_state_distribution: Distribution[S]
    ) -> Iterable[S]:
        state: S = start_state_distribution.sample()
        while True:
            yield state
```

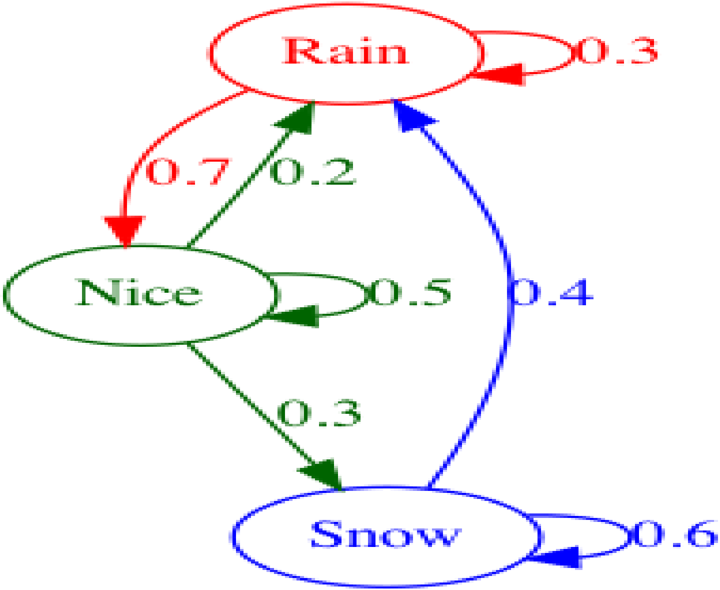
Finite Markov Process

- Finite State Space $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, $|\mathcal{N}| = m \leq n$
- We'd like a *sparse representation* for $\mathcal{P}$
- Conceptualize $\mathcal{P} : \mathcal{N} \times \mathcal{S} \rightarrow [0, 1]$ as $\mathcal{N} \rightarrow (\mathcal{S} \rightarrow [0, 1])$

```
Transition[S] = \
    Mapping[S, Optional[FiniteDistribution[S]]]
```

```
{
  "Rain": Categorical({"Rain": 0.3, "Nice": 0.7}),
  "Snow": Categorical({"Rain": 0.4, "Snow": 0.6}),
  "Nice": Categorical({
      "Rain": 0.2,
      "Snow": 0.3,
      "Nice": 0.5
  })
}
```

Weather Finite Markov Reward Process



class FiniteMarkovProcess

```
class FiniteMarkovProcess(MarkovProcess[S]):

    non_terminal_states: Sequence[S]
    transition_map: Transition[S]

    def __init__(self, transition_map: Transition[S]):
        self.non_terminal_states = [s for s, v in tran
                                    if v is not None]
        self.transition_map = transition_map

    def transition(self, state: S) -> \
            Optional[FiniteDistribution[S]]:
        return self.transition_map[state]

    def states(self) -> Iterable[S]:
        return self.transition_map.keys()
```

Order of Activity for Inventory Markov Process

$\alpha :=$ On-Hand Inventory, $\beta :=$ On-Order Inventory, $C :=$ Store Capacity

- Observe State S_t: (α, β) at 6pm store-closing
- Order Quantity $:= \max(C - (\alpha + \beta), 0)$
- Receive Inventory at 6am if you had ordered 36 hrs ago
- Open the store at 8am
- Experience random demand i with poisson probabilities:

$$\text{PMF } f(i) = \frac{e^{-\lambda}\lambda^i}{i!}, \text{ CMF } F(i) = \sum_{j=0}^{i} f(j)$$

- Inventory Sold is $\max(\alpha + \beta, i)$
- Close the store at 6pm
- Observe new state $S_{t+1} : (\max(\alpha + \beta - i, 0), \max(C - (\alpha + \beta), 0))$

Inventory Markov Process States and Transitions

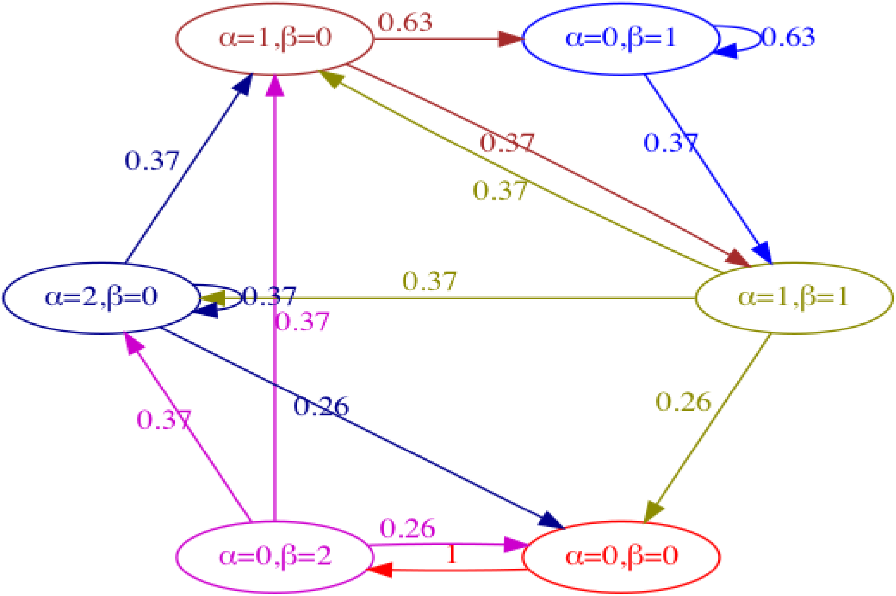
$$\mathcal{S} := \{(\alpha, \beta) : 0 \leq \alpha + \beta \leq C\}$$

If $S_t := (\alpha, \beta), S_{t+1} := (\alpha + \beta - i, C - (\alpha + \beta))$ for $i = 0, 1, \ldots, \alpha + \beta$

$$\mathcal{P}((\alpha, \beta), (\alpha + \beta - i, C - (\alpha + \beta))) = f(i) \text{ for } 0 \leq i \leq \alpha + \beta - 1$$

$$\mathcal{P}((\alpha, \beta), (0, C - (\alpha + \beta))) = \sum_{j=\alpha+\beta}^{\infty} f(j) = 1 - F(\alpha + \beta - 1)$$

Inventory Markov Process



Stationary Distribution of a Markov Process

Definition

The *Stationary Distribution* of a (Stationary) Markov Process with state space $\mathcal{S} = \mathcal{N}$ and transition probability function $\mathcal{P} : \mathcal{N} \times \mathcal{N} \rightarrow [0, 1]$ is a probability distribution function $\pi : \mathcal{N} \rightarrow [0, 1]$ such that:

$$\pi(s) = \sum_{s' \in \mathcal{N}} \pi(s) \cdot \mathcal{P}(s', s) \text{ for all } s \in \mathcal{N}$$

For Stationary Process with finite states $\mathcal{S} = \{s_1, s_2, \ldots, s_n\} = \mathcal{N}$,

$$\pi(s_j) = \sum_{i=1}^{n} \pi(s_i) \cdot \mathcal{P}(s_i, s_j) \text{ for all } j = 1, 2, \ldots n$$

Turning $\mathcal{P}$ into a matrix, we get: $\boldsymbol{\pi}^T = \boldsymbol{\pi}^T \cdot \boldsymbol{\mathcal{P}}$
$\boldsymbol{\mathcal{P}}^T \cdot \boldsymbol{\pi} = \boldsymbol{\pi} \Rightarrow \boldsymbol{\pi}$ is an eigenvector of $\boldsymbol{\mathcal{P}}^T$ with eigenvalue of 1

MRP Definition for Discrete Time, Countable States

Definition

A *Markov Reward Process (MRP)* is a Markov Process, along with a time-indexed sequence of *Reward* random variables $R_t \in \mathbb{R}$ for time steps $t = 1, 2, \ldots$, satisfying the Markov Property (including Rewards): $\mathbb{P}[(R_{t+1}, S_{t+1}) | S_t, S_{t-1}, \ldots, S_0] = \mathbb{P}[(R_{t+1}, S_{t+1}) | S_t]$ for all $t \geq 0$.

$$S_0, R_1, S_1, R_2, S_2, \ldots, S_{T-1}, R_T, S_T$$

- By default, assume stationary: $\mathbb{P}[(R_{t+1}, S_{t+1}) | S_t]$ independent of t
- Stationary MRP specified with function $\mathcal{P}_R : \mathcal{N} \times \mathbb{R} \times \mathcal{S} \rightarrow [0, 1]$

$$\mathcal{P}_R(s, r, s') = \mathbb{P}[(R_{t+1} = r, S_{t+1} = s') | S_t = s]$$

- $\mathcal{P}_R$ known as the *Transition Probability Function*

@abstractclass MarkovRewardProcess

```
class MarkovRewardProcess(MarkovProcess[S]):
    @abstractmethod
    def transition_reward(self, state: S)\
            -> Optional[Distribution[Tuple[S, float]]]
        pass

    def transition(self, state: S) -> Optional[Distrib
        distribution = self.transition_reward(state)
        if distribution is None:
            return None

        def next_state(distribution=distribution):
            next_s, _ = distribution.sample()
            return next_s

        return SampledDistribution(next_state)
```

MRP Reward Functions

- The reward transition function $\mathcal{R}_T : \mathcal{N} \times \mathcal{S} \rightarrow \mathbb{R}$ is defined as:

$$\mathcal{R}_T(s, s') = \mathbb{E}[R_{t+1} | S_{t+1} = s', S_t = s]$$

$$= \sum_{r \in \mathbb{R}} \frac{\mathcal{P}_R(s, r, s')}{\mathcal{P}(s, s')} \cdot r = \sum_{r \in \mathbb{R}} \frac{\mathcal{P}_R(s, r, s')}{\sum_{r \in \mathbb{R}} \mathcal{P}_R(s, r, s')} \cdot r$$

- The reward function $\mathcal{R} : \mathcal{N} \rightarrow \mathbb{R}$ is defined as:

$$\mathcal{R}(s) = \mathbb{E}[R_{t+1} | S_t = s]$$

$$= \sum_{s' \in \mathcal{S}} \mathcal{P}(s, s') \cdot \mathcal{R}_T(s, s') = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathbb{R}} \mathcal{P}_R(s, r, s') \cdot r$$

Inventory MRP

- Embellish Inventory Process with Holding Cost and Stockout Cost
- Holding cost of h for each unit that remains overnight.
- Think of this as "interest on inventory", also includes upkeep cost
- Stockout cost of p for each unit of "missed demand"
- For each customer demand you could not satisfy with store inventory
- Think of this as lost revenue plus customer disappointment ($p \gg h$)

Order of Activity for Inventory MRP

$\alpha :=$ On-Hand Inventory, $\beta :=$ On-Order Inventory, $C :=$ Store Capacity

- Observe State S_t: (α, β) at 6pm store-closing
- Order Quantity $:= \max(C - (\alpha + \beta), 0)$
- Record any overnight holding cost $(= h \cdot \alpha)$
- Receive Inventory at 6am if you had ordered 36 hours ago
- Open the store at 8am
- Experience random demand i with poisson probabilities:

$$\text{PMF } f(i) = \frac{e^{-\lambda}\lambda^i}{i!}, \text{ CMF } F(i) = \sum_{j=0}^{i} f(j)$$

- Inventory Sold is $\max(\alpha + \beta, i)$
- Record any stockout cost due $(= p \cdot \max(i - (\alpha + \beta), 0))$
- Close the store at 6pm
- Register reward R_{t+1} as negative sum of holding and stockout costs
- Observe new state $S_{t+1} : (\max(\alpha + \beta - i, 0), \max(C - (\alpha + \beta), 0))$

Finite Markov Reward Process

- Finite State Space $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, $|\mathcal{N}| = m \leq n$
- We'd like a *sparse representation* for $\mathcal{P}_R$
- Conceptualize $\mathcal{P}_R : \mathcal{N} \times \mathbb{R} \times \mathcal{S} \rightarrow [0,1]$ as $\mathcal{N} \rightarrow (\mathcal{S} \times \mathbb{R} \rightarrow [0,1])$

```
StateReward = FiniteDistribution[Tuple[S, float]]
RewardTransition = Mapping[S, Optional[StateReward[S]]]
```

Return as "Accumulated Discounted Rewards"

- Define the *Return* G_t from state S_t as:

$$G_t = \sum_{i=t+1}^{\infty} \gamma^{i-t-1} \cdot R_i = R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \ldots$$

- $\gamma \in [0, 1]$ is the discount factor. Why discount?
  - Mathematically convenient to discount rewards
  - Avoids infinite returns in cyclic Markov Processes
  - Uncertainty about the future may not be fully represented
  - If reward is financial, discounting due to interest rates
  - Animal/human behavior prefers immediate reward
- If all sequences terminate (Episodic Processes), we can set $\gamma = 1$

Value Function of MRP

- Identify states with high "expected accumulated discounted rewards"
- *Value Function* $V : \mathcal{N} \rightarrow \mathbb{R}$ defined as:

$$V(s) = \mathbb{E}[G_t | S_t = s] \text{ for all } s \in \mathcal{N}, \text{ for all } t = 0, 1, 2, \ldots$$

- Bellman Equation for MRP (based on recursion $G_t = R_{t+1} + \gamma \cdot G_{t+1}$):

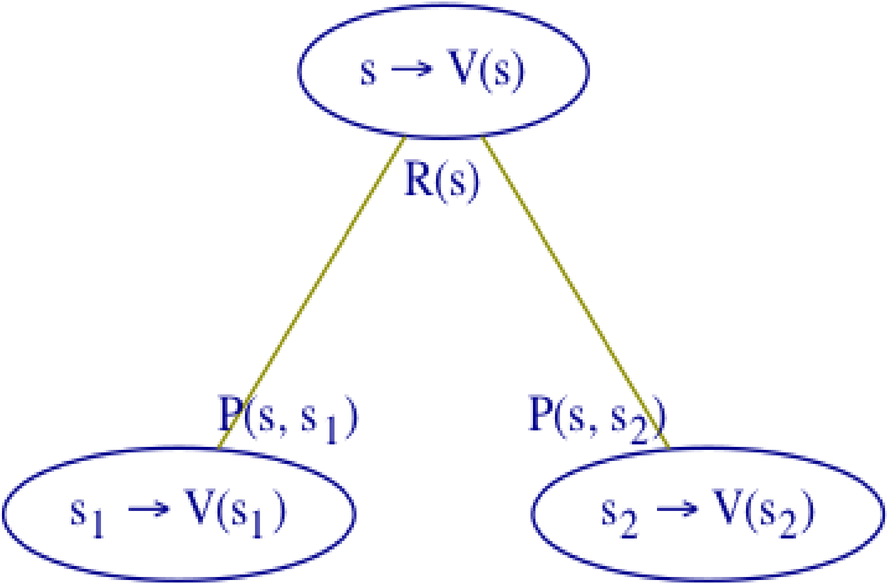
$$V(s) = \mathcal{R}(s) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}(s, s') \cdot V(s') \text{ for all } s \in \mathcal{N}$$

- In Vector form:

$$\boldsymbol{v} = \boldsymbol{\mathcal{R}} + \gamma \boldsymbol{\mathcal{P}} \cdot \boldsymbol{v}$$
$$\Rightarrow \boldsymbol{v} = (\boldsymbol{I}_m - \gamma \boldsymbol{\mathcal{P}})^{-1} \cdot \boldsymbol{\mathcal{R}}$$

  where $\boldsymbol{I}_m$ is $m \times m$ identity matrix
- If m is large, we need Dynamic Programming (or Approx. DP or RL)

Visualization of MRP Bellman Equation



Key Takeaways from this Chapter

- **Markov Property**: Enables us to reason effectively & compute efficiently in practical systems involving sequential uncertainty
- **Bellman Equation**: Recursive Expression of the Value Function - this equation (and its MDP version) is the core idea within all Dynamic Programming and Reinforcement Learning algorithms.