MARKOV CHAINS AND MARKOV DECISION PROCESSES

YUZHOU WANG

ABSTRACT. This paper provides a brief introduction to Markov chains and their fundamental properties. Then we use the previous knowledge to construct Markov Decision Processes (MDPs), which give a framework for modeling decision-making processes in optimization problems. Policy iteration and value iteration algorithms are introduced to solve for the optimal result of such models. We finish by introducing the hidden Markov models to solve the part-of-speech tagging problem.

CONTENTS

1. INTRODUCTION

Markov Decision Processes (MDPs) theory studies the sequential optimization of stochastic systems in discrete time. MDPs are built upon the Markov chain, a stochastic model describing a sequence of events where the probability of each event depends only on the previous event.

The Markov chain, named after Andrey A. Markov, was first studied by Markov himself in the early 20th century. One of his earliest examples of Markov chains was the distribution of vowels in Russian, where he studied Pushkin's novel *Eugene Onegin* to determine whether vowels and consonants could form a simple Markov chain (Grinstead & Snell 1997). Richard Bellman systematically developed the concept of dynamic programming in various papers and introduced the policy iteration algorithm, one of the key algorithms in MDPs (1955). Following Bellman's initial work, Ronald A. Howard originally modeled MDPs to solve a marketing problem in the Sears Company, and later started the systematic study of MDPs (1976). Later, Kenneth W. Church first implemented the hidden Markov model part-of-speech tagger, which will be discussed in the extended example in the last section of this paper (1989). Currently, MDPs are largely associated with reinforcement learning, which is one of three basic machine learning paradigms within artificial intelligence. Sutton & Barto employed the properties of MDPs in learning value functions in reinforcement learning (1998).

This paper is divided into three sections. In the first section, I build the foundations of Markov chains by introducing stochastic processes, the Markov property followed by a simple economic model forecasting the economic cycle, and the long-term behavior of Markov chains.

Next, I formally introduce MDPs and define 'policy' by using my previous example. I define the optimization problem and define the objective function using the Bellman equation and Bellman expectation equation. Then I expand the Bellman equation into a calculable form and outline two dynamic programming algorithms: policy iteration and value iteration.

Finally, I conclude the paper with an extended example of natural language processing: the part-of-speech tagging problem. After introducing linguistic terms and the hidden Markov model(HMM), I finalize the objective function for decoding parts of speech.

2. Markov Chain

Definition 2.1. A **stochastic process** is a collection of **random variables** X_n indexed by time variable n. The values of X_n are the **states** of the system. The states belong to a set called the **state space**.

In the case of a **discrete-time stochastic process** X_h, h takes values in a subset of non-negative integers. We will consider a special class of discrete-time stochastic processes characterized by its **Markov property**. Informally, having **Markov property** means that all possible future states of the process depend only on the current states but not the past ones. The term **Markov property** refers to the memoryless property of a stochastic process.

Definition 2.2. A discrete-time stochastic process X_t is called a **Markov chain** if:

$$\mathbf{P}\left\{X_n = i_n \mid X_0 = i_0, \ldots, X_{n-1} = i_{n-1}\right\} = \mathbb{P}\left\{X_n = i_n \mid X_{n-1} = i_{n-1}\right\} := p(i,j)$$

in which i_n belongs to the state space for all n.

A way to represent the entries of value in a Markov chain is through *transition matrix*.

Definition 2.3. The transition matrix $\mathbf{P}$ for the Markov chain is the $N \times N$ matrix whose (i,j) entry $\mathbf{P}_{ij}$ is $p(i,j)$. The matrix $\mathbf{P}$ is a stochastic matrix defined as follows,

$$0 \le \mathbf{P}_{ij} \le 1, \quad 1 \le i,j \le N$$

$$\sum_{j=1}^{N} \mathbf{P}_{ij} = 1, \quad 1 \le i \le N$$

Example 2.4. Consider /a simple economy prediction model where states correspond to inflation or deflation in given economic cycles. The states represent

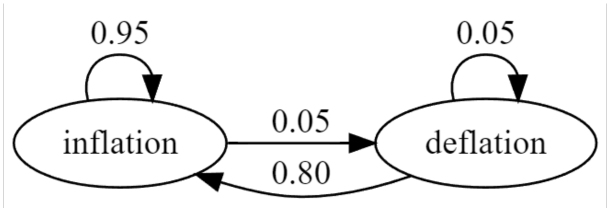

FIGURE 1. Economic Model

whether the hypothetical economy is currently undergoing inflation or deflation in a specific season.

In our model, inflation is followed by either another inflation with 95% probability or deflation with 5% probability. Deflation is followed by either inflation with 80% probability or another deflation with 20% probability. The transition matrix of our model is:

$$\mathbf{P} = \begin{bmatrix} 0.95 & 0.05 \\ 0.80 & 0.20 \end{bmatrix} \tag{2.5}$$

The purchasing power of the next economic cycle depends only on the current purchasing power. Therefore, the Markov property holds and our model is a Markov chain.

The behavior of a Markov chain after n steps is given by the transition matrix $\mathbf{P}^n$. The n-step transition probability for an (i,j) entry in $\mathbf{P}^n$ is denoted as $p_n(i,j)$.

Definition 2.6. A probability vector $\vec{n}$ is called an **invariant probability distribution for P** if,

$$\vec{n}\mathbf{P} = \vec{n}$$

In other words, such an invariant probability vector is a left eigenvector of **P** with eigenvalue 1.

Definition 2.7. In a Markov chain X_h with state space S, for some $i,j \in S$, i **leads to** j, denoted by $i \to j$, if there exists some $m \geq 0$ such that

$$p_m(i,j) > 0$$

If both $i \to j$, and $j \to i$, then i and j are said to **communicate** with each other, denoted by $i \leftrightarrow j$.

A subgroup of pairwise communicating states is called a **class**.

3. Markov Decision Processes(MDP)

3.1. **Introduction to Markov Decision Processes.** The basic theory of Markov chains leads us to Markov Decision Processes(MDP). They are optimization models for modeling decision-making in situations where outcomes are random.

Definition 3.1. A **Markov Decision Process (MDP)** is defined as a tuple $< \mathbb{S}, \mathbb{A}, p, r >$ that consists of:

(1) a state space $\mathbb{S}$;
(2) an action space $\mathbb{A}$;

(3) transition probability function $p : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$, where $p(s'|s,a) = p(s_{n+1} = s'|s_n = s, a_n = a)$ is the probability of the state $s' \in \mathbb{S}$ given the action $a \in \mathbb{A}$ in the state $s \in \mathbb{S}$;
(4) reward functions $r : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$, where $r(s,a)$ is the cost or reward of taking action a in the state s.

An MDP is called **discrete** if the state space and the action space are discrete. In the case of discrete MDPs, the transition probability function $p(s'|s,a)$ is denoted by $p_a(s,s')$.

Picking an action at a given state requires some mechanism, which is called a **policy**.

Definition 3.2. In the case of a discrete MDP, a **policy** is a sequence of mappings ϕ_t, time $t \in \mathbb{N}$, from each state $s \in \mathbb{S}$ to an action $a \in \mathbb{A}$. A **Markov policy** ϕ is defined by mappings $\phi_t \mathbb{S} \to \mathbb{A}$ such that $\phi_t(s) \in \mathbb{A}(s)$ for all $x \in \mathbb{S}$. If ϕ_t are independent of time t, then such a policy is called a **stationary Markov policy**. In other words, $\phi : \mathbb{S} \to \mathbb{A}$ such that $\phi(s) \in \mathbb{A}(s)$ for all $s \in \mathbb{S}$.

Example 3.3. The previous economic model in Example 1 can be expanded to illustrate the notion of policy.

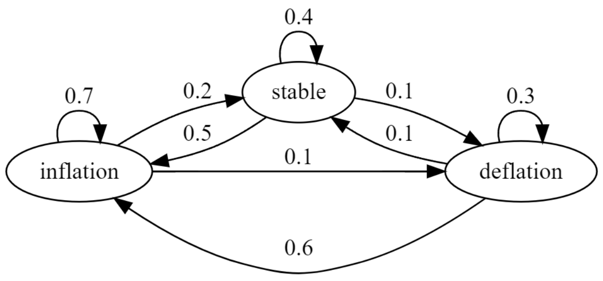

FIGURE 2. Expanded Economic Model

In this model shown by Figure 2, when the current state is ‘deflation,’ a policy could be the action of moving from deflation to stable, from deflation to inflation, or from deflation back to deflation. Any policy is considered a **Markov** policy because all of such policies belong to the action set. It is also **stationary** because for all times $t > 0$, the policies stay the same.

3.2. **Optimization.** From now on, assume all MDPs are finite state and action MDPs.

Definition 3.4. The standard form of an **optimization** problem is

$$\text{minimize } f(\vec{x})$$
$$\text{subject to } \vec{x} \in \Omega$$

The function $f : \mathbb{R}^n \to \mathbb{R}$ is called the **objective function**. The **decision variables** $\vec{x}$ is an n-vector of independent variables $\vec{x} \in \mathbb{R}^n$.

The optimization problem is a decision problem that involves searching for the best vector $\vec{x}$ of the decision variables over all possible vectors in Ω.

To start the optimization steps, it is important to define the value functions meaningfully. To do it, Richard E. Bellman introduced the concept of **Bellman equations** (1957).

Definition 3.5. The **Bellman equation** for the value function $v(s), s \in \mathbb{S}$, states that

$$v(s) = E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s] \tag{3.6}$$

here γ denotes the **discounted factor,** and R_{t+1} denotes the reward at time $t+1$.

The Bellman equation decomposes the value function into two parts: immediate reward R_{t+1} and the discounted value of successor states. In other words, the value function equals the expectation of reward one receives when leaving the state plus the value of the state that moves to. Similarly, we can construct a Bellman equation subject to some policy p.

Definition 3.7. The **Bellman expectation equation** for the value function $v_p(s), s \in \mathbb{S}$ states that

$$v_p(s) = E_p[R_{t+1} + \gamma v_p(S_{t+1}) | S_t = s] \tag{3.8}$$

here, γ denotes the **discounted factor** and R_{t+1} denotes the reward at time $t+1$. We can express value functions in terms of state-action pairs. For any $a \in \mathbb{A}$

$$v_p(s, a) = E_p[R_{t+1} + \gamma v_p(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \tag{3.9}$$

We can use this result to find the state-value function and the state-action value function for a given MDP with a given policy p.

Note that, for a given MDP, there could exist many different state-action value functions subject to different policies.

Definition 3.10. Let $v(i, p)$ be the utility function under policy p, and let state $i \in S$ be the initial state. The **value vector** v of the utility function is $v_*(i) := \sup_p v(i, p), i \in \mathbb{S}$.

If a policy p satisfies $v(i, p) = v_*(i), \forall i \in S$, it is called an **optimal policy**. Formally, we write:

$$\pi_*(a|s) = \begin{cases} 1, & \text{if } a = \operatorname*{argmax}_{a \in \mathbb{A}} v_*(s, a), \\ 0, & \text{otherwise} \end{cases} \tag{3.11}$$

This binary function helps to determine an optimal policy from a state.

We can expand equation (3.8) into,

$$
\begin{aligned}
v_p(s) &= E_p[R_{t+1} + \gamma v_p(S_{t+1})|S_t = s] \\
&= E_p[R_{t+1}|S_t = s] + \gamma E_p[v_p(S_{t+1})|S_t = s] \\
&= \sum_{r\in\mathcal{R}} rp(r|s) + \gamma \sum_{r\in\mathcal{R}} \sum_{s'\in\mathbb{S}} \sum_{a\in\mathbb{A}} v_p(s')p(s'|s, r|a)\pi(a|s) \\
&= \sum_{r\in\mathcal{R}} \sum_{s'\in\mathbb{S}} \sum_{a\in\mathbb{A}} r\pi(a|s)p(s'|s, r|a) + \gamma \sum_{r\in\mathcal{R}} \sum_{s'\in\mathbb{S}} \sum_{a\in\mathbb{A}} v_p(s')p(s'|s, r|a)\pi(a|s) \\
&= \sum_{a\in\mathbb{A}} \pi(a|s) \sum_{r\in\mathcal{R}} \sum_{s'\in\mathbb{S}} p(s'|s, r|a)[r + \gamma v_p(s')] \\
&= \sum_{a\in\mathbb{A}} \pi(a|s)(r(a, s) + \gamma \sum_{s'\in\mathbb{S}} p(s'|s, a)v_p(s'))
\end{aligned}
\tag{3.12}
$$

This form of the Bellman expectation equation could be easily calculated.

Proving the existence and then computing the optimal policies are the goals of Markov decision theory. The value function can be recognized as the objective function in such models. This objective function has two properties: **optimal substructure** and **overlapping sub-problems**, which are the criteria of **dynamic programming** problems.

Definition 3.13. A problem is said to have **optimal substructure** if an optimal solution can be built from optimal solutions of the sub-problems.
A problem is said to have **overlapping sub-problems** if the problem can be divided into smaller sub-problems that can be revisited over and over again.

The Bellman equation states that the value of a state is equal to the function of immediate reward and the discounted value of successor states. This equation calculates the optimal solution from earlier states. The value functions ensure that we only need to compute earlier states once.

3.3. Policy Iteration.

One method to find a solution to this equation is **policy iteration**. Algorithm 1 evaluates a policy p using Bellman Expectation Equation and greedily (making the locally optimal choice) calculates the value function to find the optimal policy. In the execution of the algorithm, the result is continuously improved by repeatedly evaluating the policy.

Theorem 3.14. *The policy iteration algorithm* ***converges*** *to an optimal solution (Bellman, 1955).*

By the convergence property, we can construct an algorithm based on dynamic programming. An overview of Algorithm 1 is shown below.

Algorithm 1 Policy Iteration

```
Require: states $\mathbb{S}$, actions $\mathbb{A}$
Ensure: Convergence of $v(s), s \in S$
  $V(s) \leftarrow$ random values, $s \in \mathbb{S}$
  $\delta \leftarrow$ threshold for determining convergence
  policies $\pi(s) \leftarrow$ random values
  while optimal $= 0$ do
      while the maximum difference between two iterations $< \delta$ do
          for all $s \in \mathbb{S}$ do
              for all $s' \in \mathbb{S}$ do
                  for all $r' \in \mathcal{R}$ do
                      $V(s) \leftarrow V(s) + p(s'|s, r|a)[r + \gamma V(s')]$
                  end for
              end for
              $V(s) \leftarrow \max v(s, a)$
          end for
      end while
      optimal $\leftarrow 1$
      for all $s \in \mathbb{S}$ do
          $\pi(s) \leftarrow \underset{a \in \mathbb{A}}{\operatorname{argmax}} \sum_{s' \in \mathbb{S}, r' \in \mathcal{R}} p(s'|s, r|a)[r + \gamma V(s')]$
          if the previous action $= \pi(s)$ then
              optimal $\leftarrow 0$
          end if
      end for
      if optimal $= 1$ then
          return
      end if
  end while
```

In some cases, early stopping conditions, including fixing the maximum number of iterations or setting a relatively larger threshold, could be implemented to accelerate the convergence of the algorithm.

3.4. **Value Iteration.** Another algorithm to find the solution is **value iteration**. Instead of evaluating policies, we compute the optimal value function and then determine the policy from the final optimal value function.

Algorithm 2 Value Iteration

```
Require: states $\mathbb{S}$, actions $\mathbb{A}$
Ensure: a policy $\pi$ such that $\pi$ is the optimal policy
  $V(s) \leftarrow$ random values, $s \in \mathbb{S}$
  $\delta \leftarrow$ threshold for determining convergence
  $V(start) \leftarrow 0$
  while the maximum difference between two iterations $< \delta$ do
      for all $s \in \mathbb{S}$ do
          $V(s) \leftarrow \max_{a \in \mathbb{A}} \sum_{s' \in \mathbb{S}} p(s'|s, r|a)[r + \gamma V(s')]$
      end for
  end while
  return $\pi, \pi(s) = \operatorname{argmax}_{a \in \mathbb{A}} \sum_{s' \in \mathbb{S}, r' \in \mathcal{R}} p(s'|s, r|a)[r + \gamma V(s')]$
```

Example 3.15. In the setting of Example 1, let the reward for being stable be 10, the penalty for deflation be −20, and the penalty of being inflation be −5. Set the values of each state at the first step as follows:

$$\tilde{\mathbf{v}}_{\mathbf{0}} = \begin{bmatrix} -5 & 10 & -20 \end{bmatrix} \tag{3.16}$$

Iterating and setting the discount factor to $\gamma = 0.9$, we get the following values:

$$\tilde{\mathbf{v}}_{\mathbf{1}} = \begin{bmatrix} -8.5 & 9.5 & -28 \end{bmatrix} \tag{3.17}$$

We apply our algorithm until termination.

4. Extended Example: Part-of-Speech Tagging

4.1. **Introduction.** Parts of speech are vital clues to sentence structure and meaning in analyzing languages. For example, since verbs in English are often preceded by adverbs and nouns, it tells us about possible neighboring words.

Definition 4.1. The task of **part-of-speech tagging** assigns a part-of-speech to each word in a text. The input is a sequence $x_1, x_2, \cdots, x_n$ of words, and the output is a sequence $y_1, y_2, \cdots, y_n$ of labels.

Example 4.2. This is a result of part-of-speech tagging in a famous sentence by Bertrand Russell (Russell 1905).

(1)

The	present	King	of	France
Determiner(DT)	Adjective(JJ)	Noun(NN)	Preposition(IN)	Noun(NN)

is	bald.
Verb(VB)	Adjective(JJ)

This is a grammatically correct sentence, so we can clearly tag the corresponding part of speech to each word. This process illustrates the property of each word in the sentence, and it paves the way for **syntactic parsing**, which analyzes syntactic relationships (e.g. constituency relationship within a verb phrase).

4.2. **Hidden Markov Model.**

Definition 4.3. A **hidden Markov model(HMM)** is defined as a tuple $<\mathbb{S}, \Sigma, p, e>$consists of:

(1) a state space $\mathbb{S}$;
(2) an **alphabet** Σ consists of **symbols** $\Sigma = \sigma_1, \sigma_2, \cdots, \sigma_k$. In the case of part-of-speech tagging in English, the symbols would be the 8 parts of speech and the alphabet would be the set of parts of speech.
(3) transition probability $p : \mathbb{S} \to \mathbb{R}$, where $p(s'|s) = p(q_{t+1} = s', q_t = s)$ is the probability of transitioning from the state $s \in \mathbb{S}$ to the state $s' \in \mathbb{S}$ at time t.
(4) emission probability function $e : \mathbb{S} \times \Sigma \to \mathbb{R}$, where $e(s, j)$ is the probability of generating symbol $\sigma_j \in \Sigma$ in state $s \in \mathbb{S}$;

The **hidden Markov Model** is based on the Markov chains and the Markov Decision Processes discussed previously. However, events in HMM are hidden. In part-of-speech tagging, the parts of speech of words are hidden events, since they must be inferred from the word sequence.

The transition probabilities represent the probability of a tag occurring given the previous tag in a **text corpus**, a language resource consisting of a large set of texts. The formula for calculating the transition probability is

$$p(q_{t+1} = s'|q_t = s) = \frac{C(q_{t-1}, q_t)}{C(q_t)} \tag{4.4}$$

where $C(\sigma_i, \sigma_j)$ is the probability of tag $\sigma_i \in \Sigma$ occurring before tag $\sigma_j \in \Sigma$, and $C(\sigma_i)$ is the probability of occurring tag $\sigma_i \in \Sigma$ in the corpus.

The emission probabilities represent the probability that a given tag will be associated with a certain word. The formula for calculating the emission probability is

$$p(w_i|q_t = s) = \frac{p(t_i|w_i)}{C(q_t)} \tag{4.5}$$

where $C(\sigma_i)$ is the probability of occurring tag $\sigma_i \in \Sigma$ in the corpus, and w_i is the i-th word in the text.

Example 4.6. Assume determiners(DT) occur 50000 times, of which it is followed by a singular noun(NN) 45000 times in a specific corpus, the transition probability will be

$$p(DT|NN) = \frac{C(NN, DT)}{C(DT)} = \frac{45000}{50000} = 0.9 \tag{4.7}$$

Of all 50000 occurrences of determiner in the corpus, it is associated with *a* 35000 times. The emission probability will be

$$p(a|DT) = \frac{C(DT, a)}{C(DT)} = \frac{35000}{50000} = 0.7 \tag{4.8}$$

4.3. **Part-of-Speech Decoding.** The goal of part-of-speech decoding is to choose the tag $t_1 \cdots t_n$ that is most possible given n words in a text $w_1 \cdots w_n$:

$$\begin{aligned} \hat{t}_{1:n} &= \underset{t_1 \ldots t_n}{\operatorname{argmax}} p\left(t_1 \ldots t_n \mid w_1 \ldots w_n\right) \\ &= \underset{t_1 \ldots t_n}{\operatorname{argmax}} \frac{p\left(w_1 \ldots w_n \mid t_1 \ldots t_n\right) p\left(t_1 \ldots t_n\right)}{p\left(w_1 \ldots w_n\right)} \\ &= \underset{t_1 \ldots t_n}{\operatorname{argmax}} p\left(w_1 \ldots w_n \mid t_1 \ldots t_n\right) p\left(t_1 \ldots t_n\right) \end{aligned} \tag{4.9}$$

By the Markov assumption,

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} \prod_{i=1}^{n} p\left(w_i \mid t_i\right) p\left(t_i \mid t_{i-1}\right) \tag{4.10}$$

The parts on the right hand side of (4.9) are emission probability and transition probability defined on 4.2. We can employ policy iteration or value iteration discussed above to solve this decoding problem. Moreover, **Viterbi algorithm**, a dynamic programming algorithm, can solve this problem in $O(N^2T)$ time, which is significantly faster than the previous iteration-based algorithms (Viterbi 1967).

Acknowledgments

I would like to sincerely thank my mentor, Polina Baron, for the dedicated guidance on the selection of research subject, generous help throughout the process, and the final formation of the paper. Many thanks to Professor J. Peter May for organizing this REU program, and Professor Daniil Rudenko for such wonderful lectures in the apprentice program.

References

[1] Bellman, R. (1955). Functional equations in the theory of dynamic programming. V. Positivity and quasi-linearity. *Proceedings of the National Academy of Sciences, 41*(10), 743–746. https://doi.org/10.1073/pnas.41.10.743

[2] Bellman, R. (1957). *Dynamic Programming.* Princenton University Press.

[3] Church, K. W. (1989). A stochastic parts program and noun phrase parser for unrestricted text. *ICASSP.*

[4] Feinberg, E. A., & Kallenberg, L. (2002). Finite State and Action MDPs. In *Handbook of Markov Decision Processes: Methods and Applications* (pp. 3–71). essay, Kluwer Academic Publishers.

[5] Grinstead, C., & Snell, L. (1997). *Introduction to Probability.* American Mathematical Soc. pp. 464–466. ISBN 978-0-8218-0749-1.

[6] Howard, R. A. (1978). Comments on the origin and application of Markov Decision Processes. *Dynamic Programming and Its Applications*, 201–205. https://doi.org/10.1016/b978-0-12-568150-6.50015-5

[7] Russell, B. (1905). On denoting. *Mind, 14*(4): 479–493. doi:10.1093/mind/XIV.4.479.

[8] Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning.* Cambridge, MA: MIT Press.

[9] Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory, 13*(2), 260–269. https://doi.org/10.1109/tit.1967.1054010