02465: Introduction to reinforcement learning and control

Monte-carlo methods and TD learning

Tue Herlau

DTU Compute, Technical University of Denmark (DTU)

Lecture Schedule

Dynamical programming

1. The finite-horizon decision problem
   2 February
2. Dynamical Programming
   9 February
3. DP reformulations and introduction to Control
   16 February

Control

4. Discretization and PID control
   23 February
5. Direct methods and control by optimization
   1 March
6. Linear-quadratic problems in control
   8 March
7. Linearization and iterative LQR
   15 March

Reinforcement learning

8. Exploration and Bandits
   22 March
9. Policy and value iteration
   5 April
10. **Monte-carlo methods and TD learning**
    12 April
11. Model-Free Control with tabular and linear methods
    19 April
12. Eligibility traces and value-function approximations
    26 April
13. Q-learning and deep-Q learning
    3 May

Syllabus: `https://02465material.pages.compute.dtu.dk/02465public`
Help improve lecture by giving feedback on DTU learn

Reading material:

- [SB18, Chapter 5-5.4+5.10; 6-6.3]

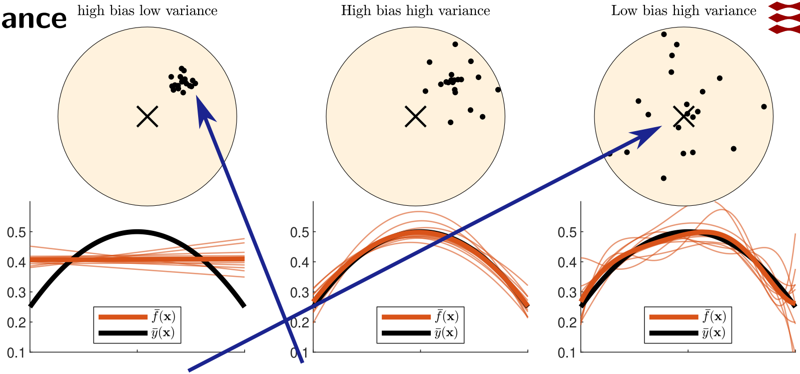
Learning Objectives

- Monte-Carlo rollouts to estimate the value function
- Monte-carlo rollouts for control
- Temporal difference learning

Housekeeping

- I am going to begin uploading the in-class examples (`irlc/lectures`)
  - Experiment! I have been hesitant because of varying *cough* code quality. I may remove these in case they cause confusion.
  - In-class demos are not exam material
- DTU server issues made the course website slow. I created a mirror (see DTU learn) and a copy of [SB18]
- DTU Course survey is online; remember to give your TAs feedback
  - Remember that concrete feedback is easier to act on
- This week the theoretical exercise is a bit longer because MC methods are less nice to implement (but try the TD(0) problem)

Bias/Variance



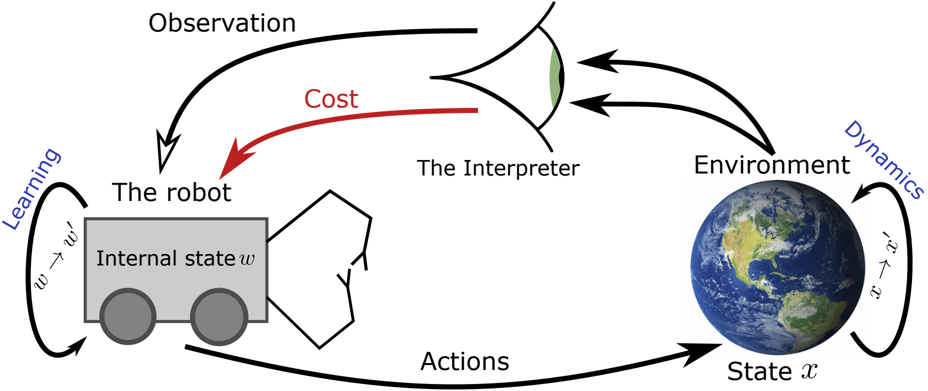
- An estimator can be **unbiased** and **biased**

$$\mathbb{E}[X] \approx \frac{1}{n}\sum_{i=1}^{n} x_i = \hat{\mu}_1, \quad \mathbb{E}[X] \approx \frac{1}{n+\sqrt{n}}\sum_{i=1}^{n} x_i = \hat{\mu}_2$$

- A biased estimator is **asymptotically consistent** if it is unbiased as $n \to \infty$:

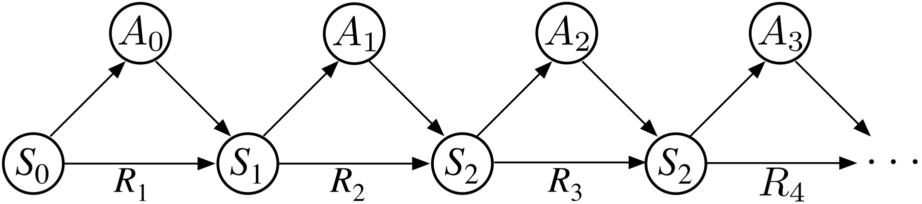
$$\hat{\mu}_2 = \frac{1}{n+\sqrt{n}}\sum_{i=1}^{n} x_i = \frac{n}{n+\sqrt{n}}\frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{1+\frac{1}{\sqrt{n}}}\hat{\mu}_1$$

Monte-Carlo estimation and control



- **Model free**; requires no knowledge of MDP
- Uses simplest possible idea: State value = mean return
- **Limitation**: Can only be used on episodic MDPs

From last time



Value and action-value function

The **state-value function** $v_\pi(s)$ is the expected return starting in s and assuming actions are selected using π:

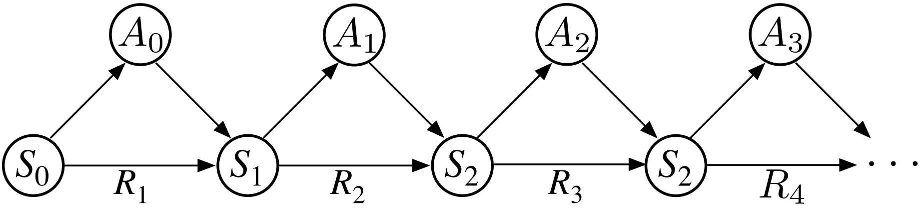
$$v_\pi(s) = \mathbb{E}_\pi\left[G_t | S_t = s\right], \quad A_t \sim \pi(\cdot|S_t)$$

The **action-value function** $q_\pi(s, a)$ is the expected return starting in s, taking action a, and then follow π:

$$q_\pi(s, a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right]$$

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

Monte Carlo evaluation: Idea



- Recall return defined by

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

- Each rollout by a policy π, starting in s, is an estimate of

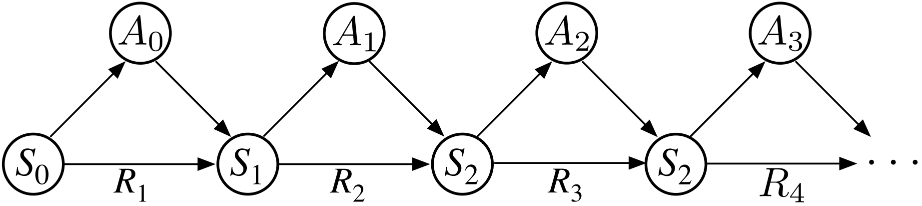
$$v_\pi(s) = \mathbb{E}_\pi\left[G_t | S_t = s\right], \quad A_t \sim \pi(\cdot|S_t)$$

- Each rollout of π, starting in s and taking action a, is an estimate

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right]$$

`unf_policy_evaluation_gridworld.py` `mc_value_first_one_state_.py`, `mc_value_first_one_state_b.py`

Every-Visit Monte-Carlo value estimation



Simulate an episode of experience $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, r_T$ using π

- **First** step t we visit a state s **Every** step t we visit a state s
- Increment number of times s visited $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value estimate is $V(s) = \frac{S(s)}{N(s)}$

Value estimate converge to $v_\pi(s)$

- **Every-visit is biased but consistent (non-trivial)**

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    Unless S_t appears in $S_0, S_1, \dots, S_{t-1}$:

      Append G to $Returns(S_t)$

      $V(S_t) \leftarrow \text{average}(Returns(S_t))$

`lecture_09_mc_value_first.py`, `mc_value_every_one_state.py`, `lecture_09_mc_value_every.py`

Quiz: A two-state gridworld

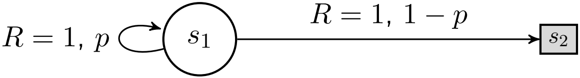


Figure: A simple MRP with one non-terminal state s_1 and one terminal state s_2. With probability p the process stay in s_1 and with probability $1 - p$ it jumps to s_2, and in each jump it gets a reward of $R_t = 1$.

Assume that $\gamma = 1$ and we evaluate the agent for the episode:

- s_1, s_1, s_1, s_2 (accumulated reward 3)

What is the estimated return using (1) first visit and (2) every-visit Monte-Carlo?

a. First-visit: $V^{\text{first}}(s_1) = 3$, every-visit: $V^{\text{every}}(s_1) = 2$

b. First-visit: $V^{\text{first}}(s_1) = 3$, every-visit: $V^{\text{every}}(s_1) = 3$

c. First-visit: $V^{\text{first}}(s_1) = 1$, every-visit: $V^{\text{every}}(s_1) = 2$

d. First-visit: $V^{\text{first}}(s_1) = 1$, every-visit: $V^{\text{every}}(s_1) = 3$

Incremental mean

Recall from the bandit-lecture that:

$$
\begin{aligned}
\mu_n &= \frac{1}{n}\sum_{i=1}^{n} x_i \\
&= \frac{1}{n}\left(x_n + \frac{n-1}{n-1}\sum_{i=1}^{n-1} x_i\right) \\
&= \frac{1}{n}x_n + \mu_{n-1} - \frac{1}{n}\mu_{n-1} \\
&= \mu_{n-1} + \frac{1}{n}\left(x_n - \mu_{n-1}\right)
\end{aligned}
$$

Incremental updates

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

- **No α: Update $N(s) \leftarrow N(s) + 1$, $S(s) \leftarrow S(s) + G$ and estimate $V(s) = \frac{S(s)}{N(s)}$**
- **With α: $V(s) \leftarrow V(s) + \alpha(G - V(s))$**

TD(0) value-function estimation

Bellman equation

- Recursive decomposition of value function

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) | S_t = s\right]$$

- **Observation:** By the MC principle $R_{t+1} + \gamma v_\pi\left(S_{t+1}\right)$ is an estimate of $v_\pi(s)$
- The estimate of v involves v. This is known as **bootstrapping**.
  - TD(0) uses **bootstrapping**
  - Monte-Carlo does not use **bootstrapping**

TD(0)

- MC learning: G_t estimate of $v_\pi(s)$; update:

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

- TD learning: $R_{t+1} + \gamma v_\pi(S_{t+1})$ estimate of $v_\pi(s)$; update:

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

Tabular TD(0) for estimating v_π

Input: the policy π 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 π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

`lecture_09_td_keyboard.py`

Comparisons

- TD can learn **online**
  - TD can learn after each step
  - MC must wait until the end of episode to learn
- TD can learn **without** knowing the final outcome
  - TD can learn from incomplete sequences
  - MC requires complete sequences
- TD works in **non-episodic** environments
  - TD work in non-terminating environments
  - MC only works in episodic environments

Bias variance tradeof

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$ is an **unbiased** estimate of $v_\pi(S_t)$
- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is an **unbiased** estimate of $v_\pi(S_t)$
- Actual TD target $R_{t+1} + \gamma V(S_{t+1})$ is a **biased** estimate of of $v_\pi(S_t)$
- TD target has lower variance than the return-target G_t:
  - Return is a sum over rewards involving many steps
  - TD target only depend on one action, transition, reward triplet

Bias variance tradeof continued

- (first-visit) MC has high variance, no bias
  - Good convergence properties
  - (..even with function approximators)
  - Not very sensitive to initial value of V
  - Simple to use/understand (a bit annoying to implement)
- TD has low variance, some bias
  - Usually more efficient to learn than MC
  - Asymptotically consistent
    - (but not always with function approximators)
  - More sensitive to initial value (bootstrapping)

MC vs. TD

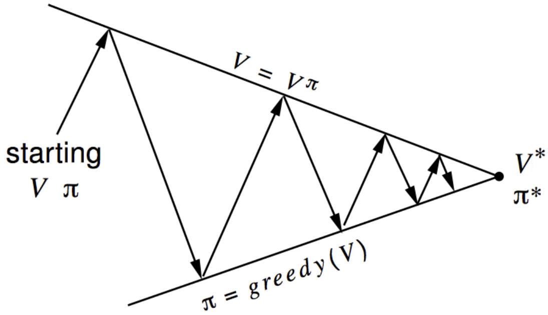
- TD exploits Markov property
  - Usually more efficient in Markov environments
- MC does not exploit Markov property:
  - Usually more efficient in non-Markovian environments

Control: On vs. off policy learning

Starting from initial policy π, goal is to determine the optimal policy π^*

- On policy learning
  - Learn (and improve) π using samples from π
- Off-policy learning
  - Learn (and improve) π using samples from some other policy π'
- Examples: π' could be (old) experience from π or an epsilon-greedy version of π

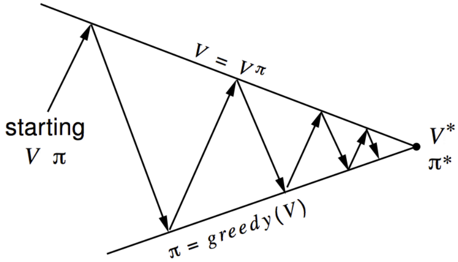
How to turn value-function iteration to controller



- Given initial policy π
- Compute v_π using policy evaluation
- Let π' be greedy policy vrt. v_π
- Repeat until $v_\pi = v_{\pi'}$

`unf_policy_improvement_gridworld.py`

Two problems



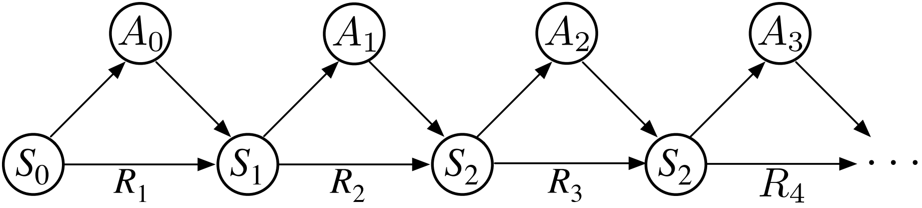
- **Problem:** We need a model to do policy improvement

$$\pi'(s) = \arg\max_a \mathbb{E}[R + \gamma V(S')|s, a]$$

  - **Solution:** Estimate/save $q_\pi(s, a)$ instead of $v_\pi(s)$: $\pi'(s) = \arg\max_a Q(s, a)$
- **Problem:** Acting greedily means all $q(s, a)$-values are not estimated by MC
  - **Solution:** Be ε-greedy in π

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\text{argmax}} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

First-Visit Monte-Carlo value estimation



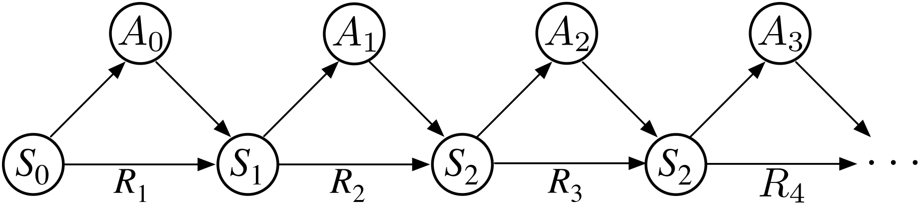
Simulate an episode of experience $s_0, a_0, r_1, s_1, a_1, r_2, \dots, r_T$ using π

- **First** step t we visit a state s
- Increment number of times s visited $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value estimate is $V(s) = \frac{S(s)}{N(s)}$

Value estimate converge to $v_\pi(s) = \mathbb{E}[G_t | S_t = s]$

`lecture_10_mc_action_value_first_one_state.py`

First-Visit Monte-Carlo action-value estimation



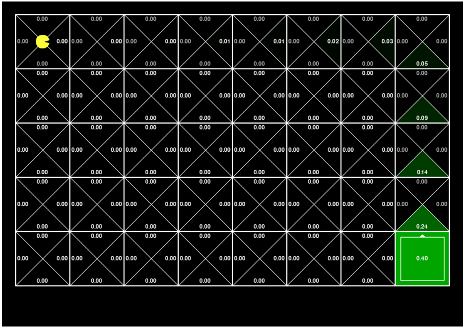
Simulate an episode of experience $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, r_T$ using π

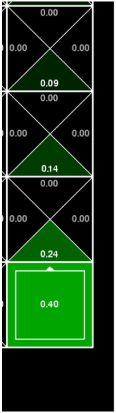
- **First** step t we visit a pair (s, a)
- Increment number of times s visited $N(s, a) \leftarrow N(s, a) + 1$
- Increment total return $S(s, a) \leftarrow S(s, a) + G_t$
- Action-value estimate is $Q(s, a) = \frac{S(s,a)}{N(s,a)}$

Action-value estimate converge to $q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$

`lecture_10_mc_q_estimation.py` (first-visit)

Quiz: Control and incremental updates





A first-visit Monte-Carlo agent (with incremental updates) is trained for one episode (terminal reward of $+1$). What was the discount factor γ?

a. $\gamma = 0.5$

b. $\gamma = 0.4$

c. $\gamma = 0.6$

d. $\gamma = 0.3$

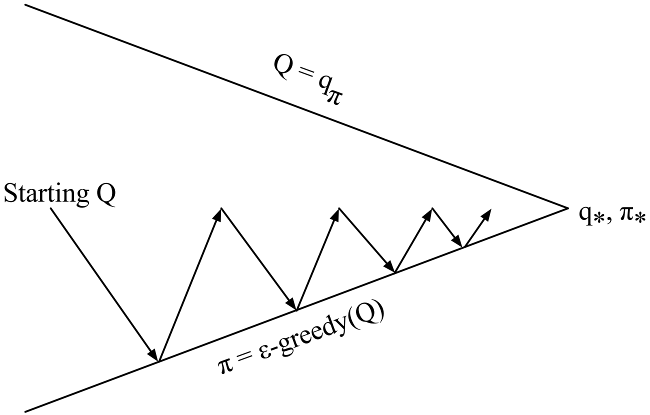
e. Don't know.

Convergence result

Policy improvement, ε-greedy version

For any ε-greedy policy π, the ε-greedy policy π' with respect to q_π is an improvement: $v_{\pi'}(s) \geq v_\pi(s)$.

Monte-Carlo control



Repeat for every episode

- **Policy evaluation**: Monte-Carlo policy evaluation to approximate $q_\pi \approx Q$
- **Policy improvement**: ε-greedy policy improvement on Q

Implementation

- To implement this, store Q-values in `self.Q[s,a]` in the `TabularAgent` class
- Note we already have implemented the epsilon-greedy exploration method

MC control

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \dots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

`lecture_10_mc_control.py` $(\alpha = \frac{1}{10}, \varepsilon = 0.15, \gamma = 0.9)$.

Greedy in the limit with infinite exploration

Greedy in limit of infinite exploration (GLIE)

GLIE means that

- All state-action pairs are explored infinitely often

$$\lim_{k \to \infty} N_k(s, a) = \infty$$

- The exploration rate ε decays to zero

$$\lim_{k \to \infty} \pi_k(a = a^* | s) = 1, \quad a^* = \arg\max_a Q_k(s, a')$$

- One way to ensure GLIE is letting $\varepsilon_k = \frac{1}{k}$
- Assuming GLIE then MC control will converge to the optimal policy.

Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning: An Introduction.
The MIT Press, second edition, 2018.
(Freely available online).