Table of contents

Preface

This notes are based in the course from Berstekas for the MIT see all lectures and other resources for complete the understanding.

Abstract

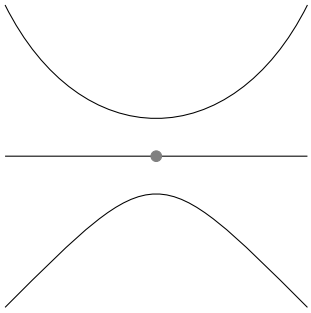
We present an introduction to the solution of multi-stage optimization problems. Starting from the dynamic programming algorithm, we consider theoretical and computational aspects, mainly of deterministic problems, and discuss how to generalize some of the results to Markovian decision processes.

The classic problem with which the essential ideas of dynamic programming can be introduced is the optimal route problem (shortest route considering distances, or cheapest route considering costs) (Bertsekas 2005; Brandimarte 2013). A simple version of this problem is shown schematically in

This display LaTeX formmula will be imagified:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

As well as this TikZ picture:



And this raw LaTeX block:

Ruta	Costo
$0 \rightarrow 1 \rightarrow 4 \rightarrow 7$	18
$0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 7$	22
$0 \rightarrow 1 \rightarrow 3 \rightarrow 6 \rightarrow 7$	18
$0 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7$	20
$\mathbf{0 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 7}$	16
$0 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 7$	23
$0 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7$	19
$0 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7$	21
$0 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7$	17
$0 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 7$	22
$0 \rightarrow 2 \rightarrow 5 \rightarrow 7$	18

In this problem, the optimal route to go from the node labeled 0 to the node labeled 7 is to be determined. The costs between each pair of nodes connected by an arrow are represented by a number next to it. For example, the cost to go from node 0 to node 2 is 6. The optimal route will be the one for which the sum of their costs is minimum. This optimal route can be obtained by an exhaustive enumeration; generating all possible routes from node 0 to node 7 and choosing the minimum cost route.

En la [ref:table] se muestran tales rutas y sus respectivos costos. Se observa que la trayectoria óptima es $ 0 →1 →3 →5 →7 $, con un costo de 16.

Outline

The textbook for chapter one is Bertsekas' book (Bertsekas 2005). Chapters 2 and 3 are adapted from Sutton's book (Ch. 3, Ch. 4, Sutton and Barto 2018). For application and broad connection with more machine learning applications, we refer to (Brunton and Kutz 2019). Also, we recommend a handbook of algorithms (Szepesvári 2022). For applications with implemented code, we follow the books (Bilgin 2020).

Abstract

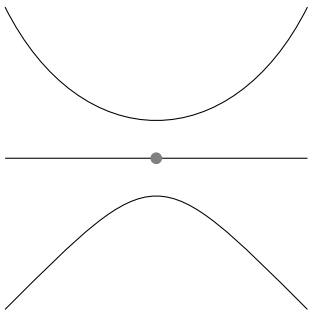
We present an introduction to the solution of multi-stage optimization problems. Starting from the dynamic programming algorithm, we consider theoretical and computational aspects, mainly of deterministic problems, and discuss how to generalize some of the results to Markovian decision processes.

The classic problem with which the essential ideas of dynamic programming can be introduced is the optimal route problem (shortest route considering distances, or cheapest route considering costs) (Bertsekas 2005; Brandimarte 2013). A simple version of this problem is shown schematically in

This display LaTeX formmula will be imagified:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

As well as this TikZ picture:



And this raw LaTeX block:

Ruta	Costo
$0 \to 1 \to 4 \to 7$	18
$0 \to 1 \to 3 \to 4 \to 7$	22
$0 \to 1 \to 3 \to 6 \to 7$	18
$0 \to 1 \to 3 \to 5 \to 6 \to 7$	20
$\mathbf{0 \to 1 \to 3 \to 5 \to 7}$	16
$0 \to 2 \to 3 \to 4 \to 7$	23

Ruta	Costo
$0 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7$	19
$0 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7$	21
$0 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7$	17
$0 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 7$	22
$0 \rightarrow 2 \rightarrow 5 \rightarrow 7$	18

In this problem, the optimal route to go from the node labeled 0 to the node labeled 7 is to be determined. The costs between each pair of nodes connected by an arrow are represented by a number next to it. For example, the cost to go from node 0 to node 2 is 6. The optimal route will be the one for which the sum of their costs is minimum. This optimal route can be obtained by an exhaustive enumeration; generating all possible routes from node 0 to node 7 and choosing the minimum cost route.

En la [ref:table] se muestran tales rutas y sus respectivos costos. Se observa que la trayectoria óptima es $ 0 →1 →3 →5 →7 $, con un costo de 16.

1 Introduction

Reinforcement Learning is part of a decades-long trend within artificial intelligence and machine learning toward greater integration with statistics, optimization, and other mathematical subjects. For example, the ability of some reinforcement learning methods to learn with parameterized approximators addresses the classical "curse of dimensionality" in operations research and control theory. More distinctively, reinforcement learning has also interacted strongly with psychology and neuroscience, with substantial benefits going both ways. Of all the forms of machine learning, reinforcement learning is the closest to the kind of learning that humans and other animals do, and many of the core algorithms of reinforcement learning were originally inspired by biological learning systems.

1.1 Trial-Error

According to Richard S. Sutton and Andrew G. Barto Sutton and Barto (2018)–the first authors to use the term–Reinforced Learning Reinforcement learning is about what to do, that is, how to map situations to action so that we optimize a reward. **The learner must discover which action yield the best reward by trying them.** In the most general sense, action may not only affect immediate reward but also the next situation and, through that, all subsequent rewards.

1.1.1 Sensation action and goal

At the same time, Reinforcement Learning encloses a problem, a class of solution methods, and the field that studies this problem and its solutions. Its formalism is based on the theory of controlled dynamical systems, with a strong focus on the optimal control of partially known Markov decision processes. Then, the core idea consists of capturing the essence of the problem when an agent learns through experience and interaction to reach a goal. This agent can sense the state of its environment to some extent and must be able to take action that affects the state. The agent also must have a goal or goals related to the state of the environment.

MDPs are designed to incorporate three essential elements: sensation, action, and goal. Therefore, any approach suitable for solving such problems should be considered a potential method for Reinforcement Learning.

1.2 Exploration-exploration dilemma and uncertainty

To obtain the best reward, the agent must prefer actions used in the past and perceived as effective to produce a reward. However, to discover such actions, the agent must try actions never used before. So, there is a delicate trade-off between exploiting and exploring. The agent has to exploit its knowledge to produce a reward but simultaneously has to explore to improve its reward in the future. Here, our dilemma is that neither exploration nor exploitation can be pursued exclusively without failing the task.

Another essential aspect of reinforcement learning is that it specifically deals with the entire process of a goal-directed agent interacting with an uncertain environment. This aspect differs from many approaches that only focus on subproblems rather than considering how they might contribute to the bigger picture. For example, many machine learning researchers have studied supervised Learning without specifying how such an ability would ultimately be helpful. Other researchers have developed planning theories with general goals without considering planning's role in real-time decision-making or whether the predictive models necessary for planning are well suited. Although these approaches have produced valuable results, their focus on isolated subproblems leads to significant limitations.

Reinforcement learning takes the opposite approach, beginning with a fully interactive, goal-seeking agent. In reinforcement learning, the agent has explicit goals, can sense aspects of their environment, and can choose actions to influence its environment.

1.3 Examples

- A master chess player makes a move.
- An adaptive controller adjusts parameters of a petroleum refinery's operation in real time.
- A gazelle calf struggles to its feet minutes after being born.
- A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station.
- Phil prepares his breakfast

1.4 The (possible) 4 elements of Reinforcement Learning

Given an agent, we identify four main element in a reinforcement learning model:

> a **policy**, a **reward**, a value function and (optionally) a model of the **environment**.

Policy A policy is as a set of actions that guide the agent to respond according to its perception of the environment. It's like a set of instructions that tell the agent what to do when it encounters a certain situation. In general, policies may be stochastic, specifying probabilities for each action.

Reward The reward signal thus defines what are the good and bad events for the agent. In a biological system, we might think of rewards as analogous to the experiences of pleasure or pain. They are the immediate and defining features of the problem faced by the agent. The reward signal is the primary basis for altering the policy; if an action selected by the policy is followed by low reward, then the policy may be changed to select some other action in that situation in the future. In general, reward signals may be stochastic functions of the state of the environment and the actions taken.

Value function Whereas the reward signal indicates what is good in the immediate sense, a value function specifies what is good in the long run. In simple terms, the value of a state represents the total reward an agent can anticipate to receive in the future, beginning from that state. While rewards reflect the immediate appeal of environmental states, values signify the long-term appeal of states, considering the potential future states and the rewards they offer. For example, a state might consistently yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Alternatively, the opposite could also be true. Rewards can be compared to pleasure (when high) and pain (when low). At the same time, values represent a more precise and long-term assessment of how satisfied or dissatisfied we are with the state of our environment. In a sense, rewards are primary, whereas values, as predictions of rewards, are secondary. Without rewards, there could be no values, and the only purpose of estimating values is to achieve more rewards.

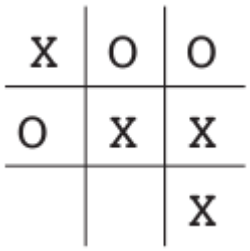
Action choices are made based on value judgments. We seek actions that bring about states of highest value, not highest reward, because these actions obtain the greatest amount of reward for us over the long run. In fact, the most important component of almost all reinforcement learning algorithms we consider is a method for efficiently estimating values.

Environment model The environment model is something that mimics the behavior of the environment or, more generally, that allows inferences to be made about how the environment will behave. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for planning. This means making decisions by considering potential future situations before they occur. For our purposes, the environment can be represented as a dynamic system through an ordinary differential equation or a discrete finite difference equation.

1.5 A toy RL-exmaple: Tic-Tac-Toe

To illustrate the general idea of reinforcement learning and contrast it with other ap- proaches, we next consider a single example in more detail.

Consider the familiar child's game of tic-tac-toe.



Although the tic-tac-toe game is a simple problem, it cannot be satisfactorily solved using classical techniques.

For instance, the classical "*minimax*" solution from game theory is not applicable here because it assumes the opponent's specific way of playing. A *minimax* player would never reach a game state from which it could lose. Even if, in reality, it always won from that state due to incorrect play by the opponent. The classical optimization methods for sequential decision problems, like dynamic programming, can find the best solution for any opponent. However, these methods need a detailed description of the opponent as input, including the probabilities of the opponent's moves in each board state.

Alternatively, this information can be estimated through experience, such as playing numerous games against the opponent. The best approach to this problem is to first learn a model of the opponent's behavior with a certain level of confidence, and then use dynamic programming to calculate an optimal solution based on the approximate opponent model.

Sutton and Barto (see pp. 9-12 Sutton and Barto 2018) propose the following way to approach tic tac toe with Reinforcement Learning:

Setup:

- First we would set up a table of numbers (or labels), one for each possible state of the game.
- Each number will be the latest estimate of the probability of our winning from that state.
- We treat this estimate as the state's value, and the whole table is the learned value function.

- State A has higher value than state B, or is considered 'better' than state B, if the current estimate of the probability of our winning from A is higher than it is from B.
- If we always play Xs, then for all states with three Xs in a row the probability of winning is 1, because we have already won.
- Similarly, for all states with three Os in a row, or that are filled up, the correct probability is 0–we cannot win from them.
- We set the initial values of all the other states to 0.5, representing a guess that we have a 50% chance of winning.

Training:

We then play many games against the opponent.

To select our moves we examine the states that would result from each of our possible moves (one for each blank space on the board) and look up their current values in the table. Most of the time we move greedily,selecting the move that leads to the state with greatest value, that is, with the highest estimated probability of winning.

Occasionally, however, we select randomly from among the other moves instead. These are called exploratory moves because they cause us to experience states that we might otherwise never see.

A sequence of moves made and considered during a game can be diagrammed as the following figure:

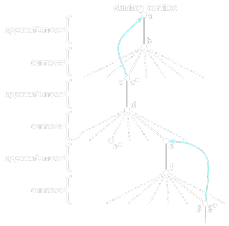


Figure 1.1: Figure taken from (Sutton and Barto 2018). A sequence of tic-tac-toe moves. Solid black lines represents moves taken during a play. Dashed lines represent plausible but not taken moves. The $*$ symbol indicates the move currently estimated to be the best. Thus the move e denotes an *exploratory* move

2 Finite Markov Decision Processes (MDPs)

That is what ChatGPT would answer to a 5-year-old kid. Alright, let's imagine you have a little robot friend named Robo. Robo likes to explore and do different things, but Robo doesn't always know what to do next. A Markov Decision Process (MDP) is like giving Robo a set of rules to help it decide what to do next based on where it is and what it knows.

Imagine Robo is in a room full of toys. Each toy is like a different choice Robo can make, like playing with blocks or reading a book. But Robo can't see the whole room at once, so it has to decide what to do based on what it can see and remember.

In an MDP, Robo learns from its past experiences. If it finds that playing with blocks usually makes it happy, it's more likely to choose that again next time. But if it tries reading a book and doesn't like it, it might choose something else next time.

So, a Markov Decision Process helps Robo make decisions by learning from what it's done before and what it can see around it, kind of like how you learn from playing with different toys and remembering which ones you like best.

2.1 The Agent–Environment Interface

:::{#exm-roboClean}

A mobile robot has the job of collecting empty soda cans in an office environment. It has sensors for detecting cans, and an arm and gripper that can pick them up and place them in an onboard bin; it runs on a rechargeable battery. The robot's control system has components for interpreting sensory information, for navigating, and for controlling the arm and gripper. High-level decisions about how to search for cans are made by a reinforcement learning agent based on the current charge level of the battery. To make a simple example, we assume that only two charge levels can be distinguished, comprising a small state set $\mathcal{S} = \{\texttt{high}, \texttt{low}\}$. In each state, the agent can decide whether to

1. actively **search** for a can for a certain period of time,
2. remain stationary and **wait** for someone to bring it a can, or
3. head back to its home base to **recharge** its battery.

When the energy level is **high**, recharging would always be foolis h, so we do not include it in the action set for this state. The action sets are then $\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$ and $\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$.

The rewards are zero most of the time, but become positive when the robot secures an empty can, or large and negative if the battery runs all the way down. The best way to find cans is to actively search for them, but this runs down the robot's battery, whereas waiting does not. Whenever the robot is searching, the possibility exists that its battery will become depleted. In this case the robot must shut down and wait to be rescued (producing a low reward).

If the energy level is **high**, then a period of active search can always be completed without risk of depleting the battery. A period of searching that begins with a **high** energy level leaves the energy level high **with** probability α and reduces it to low with probability $1 - \alpha$. On the other hand, a period of searching undertaken when the energy level is low leaves it low with probability β and depletes the battery with probability $1 - \beta$. In the latter case, the robot must be rescued, and the battery is then recharged back to **high**. Each can collected by the robot counts as a unit reward, whereas a reward of -3 results whenever the robot has to be rescued. Let $r_{\texttt{search}}$ and $r_{\texttt{wait}}$, with $r_{\texttt{search}} > r_{\texttt{wait}}$, denote the expected numbers of cans the robot will collect (and hence the expected reward) while searching and while waiting respectively. Finally, suppose that no cans can be collected during a run home for recharging, and that no cans can be collected on a step in which the battery is depleted. This system is then a finite MDP, and we can write down the transition probabilities and the expected rewards, with dynamics as indicated in the table on the left:

Figure 2.1

Exercise 2.1. Give a table analogous to that in (p.53, Ex. 3.3, Sutton and Barto 2018), but for $p(s_0, r|s, a)$. It should have columns for s, a, s_0 , r, and $p(s_0, r|s, a)$, and a row for every 4-tuple for which $p(s_0, r|s, a) > 0$.

2.2 Goals and Rewards

2.3 Returns and Episodes

2.4 Unified Notation for Episodic and Continuing Tasks

2.5 Policies and Value Functions

2.6 Optimal Policies and Optimal Value Functions

2.7 Optimality and Approximation

2.8 Summary

3 Dynamic Programming

An explanation from ChatGPT. Alright, imagine you have a big puzzle to solve, but it's too big for you to finish in one go. So, you decide to break it into smaller puzzles, and you solve each of these small puzzles one by one. But, here's the clever part: as you solve these small puzzles, you remember the solutions. That way, if you come across the same small puzzle again, you don't have to solve it all over again. You already know the answer! Dynamic programming is like solving a big puzzle by breaking it into smaller ones and remembering the solutions to the smaller ones to make solving the big puzzle easier and faster.

3.1 Policy Evaluation (Prediction)

3.2 Policy Improvement

3.3 Policy Iteration

3.4 Value Iteration

3.5 Asynchronous Dynamic Programming

3.6 Generalized Policy Iteration

3.7 Efficiency of Dynamic Programming

3.8 Summary

4 Applications

4.1 Recycling Robot

4.2 A robot with randomly moves in a grid world.

5 Project proposal

References

Bertsekas, Dimitri P. 2005. *Dynamic Programming and Optimal Control. Vol. I*. Third. Athena Scientific, Belmont, MA.

Bilgin, E. 2020. *Mastering Reinforcement Learning with Python: Build Next-Generation, Self-Learning Models Using Reinforcement Learning Techniques and Best Practices*. Packt Publishing. https://books.google.com.mx/books?id=s0MQEAAAQBAJ.

Brandimarte, Paolo. 2013. *Numerical Methods in Finance and Economics: A MATLAB-Based Introduction*. 2nd ed. Hoboken, New Jersey: John Wiley & Sons.

Brunton, Steven L., and J. Nathan Kutz. 2019. *Data-Driven Science and Engineering*. Cambridge University Press, Cambridge. https://doi.org/10.1017/9781108380690.

Sutton, Richard S., and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. Second. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA.

Szepesvári, Csaba. 2022. *Algorithms for Reinforcement Learning*. Vol. 9. Synthesis Lectures on Artificial Intelligence and Machine Learning. Springer, Cham. https://doi.org/10.1007/978-3-031-01551-9.

List of Home Works and due dates

Homework 001 due date: september 16, 2024-12:00:00

1. Read (Sec 1.1, pp 1-2 Sutton and Barto 2018) and answer the following.

Explain why Reinforcement Learning differs for supervised and unsupervised learning.

2. See the first Steve Brunton's youtube video about Reinforcement Learning. Then accordingly to its presentation explain what is the meaning of the following expression:

$$V_\pi(s) = \mathbb{E}\left(\sum_t \gamma^t r_t | s_0 = s\right).$$

3. Form (Sec. 1.7 Sutton and Barto 2018) obtain a time line pear year from 1950 to 2012. Use the following format see https://kim.quarto.pub/milestones–bar-timelines/

```
library(bibtex)
## Activate the Core Packages
biblio <- bibtex::read.bib("../references.bib")
library(tidyverse) ## Brings in a core of useful functions
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```
library(gt)        ## Tables
## Specific packages
library(milestones)
## Initialize defaults
## Initialize defaults
column <- lolli_styles()

data <- read_csv(col_names=TRUE, show_col_types=FALSE, file='rl_time_line.csv')

## Sort the table by date
data <- data |>
  arrange(date)

## Build a table
gt(data) |>
  #cols_hide(columns = event) |>
  tab_style(cell_text(v_align = "top"),
            locations = cells_body(columns = date)) |>
  tab_source_note(source_note = "Source: Wikipedia")
```

date	event	Reference
1957	The term “optimal control” appears	MR0090477 Bellman, Richard Dynamic programming. P
NA	MDPs	MR0091859 Bellman, Richard A Markovian decision pro

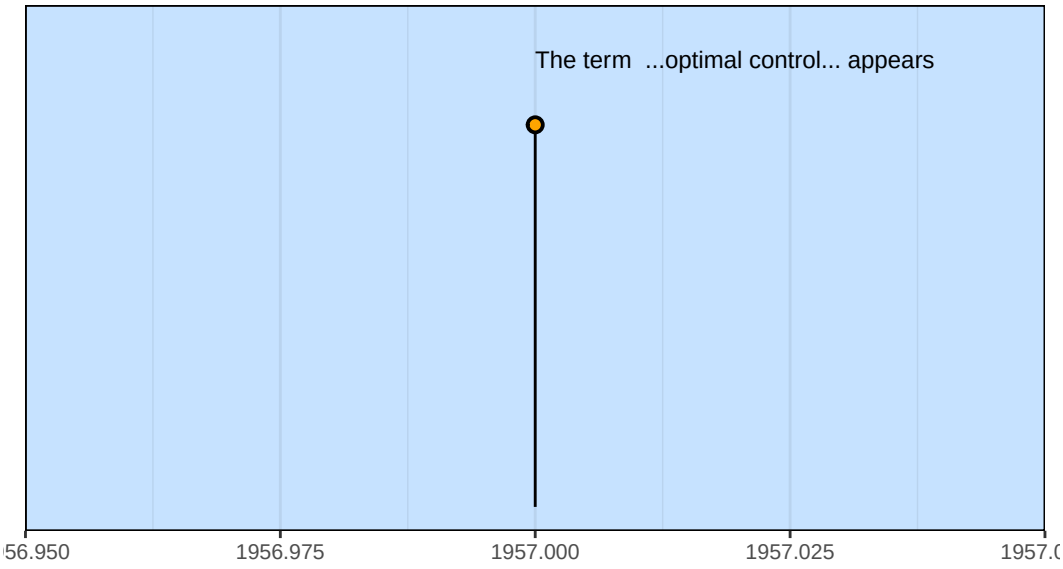
Source: Wikipedia

```
## Adjust some defaults
column$color <- "orange"
column$size  <- 15
column$source_info <- "Source: Wikipedia"

## Milestones timeline
milestones(datatable = data, styles = column)
```

```
Warning: Removed 1 row containing missing values or values outside the scale range
(`geom_text()`).
```

```
Warning: Removed 1 row containing missing values or values outside the scale range
(`geom_segment()`).
```

```
Warning: Removed 1 row containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_segment()`).

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on 'The term "optimal control" appears' in 'mbcsToSbcs': dot
substituted for <e2>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on 'The term "optimal control" appears' in 'mbcsToSbcs': dot
substituted for <80>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on 'The term "optimal control" appears' in 'mbcsToSbcs': dot
substituted for <9c>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on 'The term "optimal control" appears' in 'mbcsToSbcs': dot
substituted for <e2>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on 'The term "optimal control" appears' in 'mbcsToSbcs': dot
substituted for <80>

Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on 'The term "optimal control" appears' in 'mbcsToSbcs': dot
substituted for <9d>
```



Source: Wikipedia