Modeling Decision Trees with SilverDecisions

Edited by Przemysław Szufel • Michał Jakubczyk • Bogumił Kamiński

Modeling Decision Trees with SilverDecisions

SGH Publishing House

WARSAW 2022

Reviewers
Maciej Niewada
Bogdan Włodarczyk

Editor
Jakub Hamze

First Edition

ISBN 978-83-8030-552-6

SGH Publishing House
162 Niepodległości Ave., 02-554 Warsaw, Poland
www.wydawnictwo.sgh.waw.pl
e-mail: wydawnictwo@sgh.waw.pl

Cover design
Magdalena Limbach

DTP
Przemysław Szufel
Michał Jakubczyk
Bogumił Kamiński

Print and binding
volumina.pl Daniel Krzanowski
ul. Ks. Witolda 7–9
71–063 Szczecin
tel. 91 812 09 08
e-mail: druk@volumina.pl

Order 66/V/22

Contents

Chapter 1

Introduction

Bogumił Kamiński
Michał Jakubczyk
Przemysław Szufel

Making good decisions requires at least three elements: having identified the decision alternatives between which the choice is being made, knowing the consequences of these decision alternatives, and being able to compare the consequences.

These components are often ignored in real decisions and sometimes with good outcomes. A person might not consciously consider available strategies while trying to avoid a car accident but automatically makes an evasive maneuver. Similarly, someone may not think about the long-term consequences of risky behavior (e.g., smoking) and does not bear them due to luck. Finally, an employee may select a job based on an overall holistic assessment, rather than a point-by-point deliberation of the pros and cons, and be satisfied in the long run.

Nevertheless, we believe that a conscious deliberation of the above three elements helps. Trivially, you cannot make a better choice than the best alternative that is available and that you have identified. Likewise, one may fail to reject an alternative that may bring a very unfavorable outcome if they do not think about the possible consequences of their choices. We do not know which alternatives suits one's needs best if we do not know which combination of consequences they like most (e.g., a high quality is coming for a high price or a moderate quality available for a moderate price).

Obviously, in order to make good choices consistently, it is necessary to have domain-specific knowledge relevant to the field in which the decision is to be made. For example, physicians know the available medical options (e.g., drugs); they also read publications and learn about possible outcomes, positive or negative ones. Having gone through similar decisions in the past and having experienced the consequences of the past options (being satisfied or dissatisfied) help us to realize what we care for and what trade-offs we are willing to make.

Besides the domain-specific knowledge, organizing the decision-making process in a systematized way helps to better understand the available options and their consequences. For that purpose, *decision trees* are a particularly useful tool in our experience. The usefulness of decision trees comes from the fact that when a decision problem is sequential, i.e., when it entails several actions following one another (possibly intertwined with uncertain events that depend on the actions we take), then it may not be easy to intuitively grasp all the possible strategies that might be quite complex.

Let us illustrate this difficulty with an example. Imagine a patient who may have a disease that needs to be treated before it manifests with certainty. There are two diagnostic tests we may use, and each bears a risk of a false positive or a false negative signal (i.e., may suggest the presence of the disease in a truly healthy individual or may fail to detect a truly ill person). The decision to treat or not (to perhaps find out later the person was ill) the patient can be made based on any combination of tests. We ignore the parameters of the problem for simplicity, i.e., the probabilities.[1] The reader may be surprised to find out there are 74 strategies for how we may approach the patients' diagnosis and treatment.[2] Not all of them make sense (e.g., conducting both tests and then treating the patient regardless of the results), but otherwise, how can we make the best choice if we do not represent all the alternatives we have in a structured way?

Structuring the decision problem can help avoid subjectivity in decision-making and make it easier to focus on the quantitative aspects of decision outcomes. In Chapter 5, we have presented examples in the domain of pandemic management, agriculture, sales, services, and transportation. Moreover, decision trees can also be used to present, suggest and discuss decision scenarios. For example, this approach is used by law firms to clearly discuss the decision situation and possible choices, probabilities, and outcomes of their clients. This approach makes it easier to explain and recommend

[1] We assume that the same type of test performed twice necessarily gives the identical result

[2] Assuming the two tests cannot be performed simultaneously. We should also subtract 16 strategies that are equivalent, e.g., always performing two tests and then making a decision based on a combination of results, then the ordering of the tests does not matter.

decisions made in a lawsuit. Decision trees are a way to model the sequential decision problems we consider in the book in a graphical way, such as graphs (a mathematical concept, more information is given in the book in Chapter 3). Decision trees make it explicit what the possible paths of events may occur for a given strategy and with what probability; hence, they allow measuring the consequences of available alternatives. The decision trees have been successfully used for more than half a century [15, 25]. Beyond the typical business context, the decision trees are also used in health technology assessment [5, 24, 29], and this method of problem-solving forms a part of the standard teaching curriculum and guidelines [10, 12].

The trees representing actual problems (like the one above) may grow huge, and solving them (to be exact, solving the problem represented by a given tree) may require many calculations. Therefore, dedicated software is useful both in the construction and in the solving phase. We believe that the Silver Decisions software (`SilverDecisions`, henceforth) possesses all the features that help the user with the three difficult steps of decision-making, as described above. The basic functionalities of `SilverDecisions` are presented in Section 2.1; for example, `SilverDecisions` allows building a tree via a graphical interface; copy-paste makes it easier to represent more complex problems; variables can be defined and used to increase flexibility. The outcomes of each policy can be presented graphically (e.g., league table in multiple criteria problems discussed in Chapter 4). The user can also inspect the pessimistic or optimistic outcome for each strategy by considering various decision-making rules. Moreover, we designed `SilverDecisions` software in a way that makes it programmable by using variables and functions in the process of calculating the results. Those outcomes can be later presented in various labels within a decision tree. Finally, `SilverDecisions` enhances inspecting the impact of how the pay-offs are evaluated; for example, in multiple criteria problems, the user may define a range of values measuring the trade-offs between individual criteria to understand the impact of own preferences on the recommended behavior.

`SilverDecisions` can also help to ask questions beyond the original problem: how the parameters must change to require re-evaluation (sensitivity analysis, see Chapter 3) or how worthy it is to pursue additional information (Chapter 4).

An interested reader can find an additional discussion of the mathematical theory behind the analysis of the sensitivity of decision trees in [18].

Modern data science and quantitative methods offer several other tools and approaches for supporting a decision process. This includes simulation modeling (building digital twin models of the phenomena interesting to the decision-maker and experimenting with such models – for example, see [23]), mathematical programming optimization (models that are defined as a set of variables, constraints, and a goal function are solved by tools known as optimization solvers – for example, see [38]), machine learning (models used for classification, prediction, and segmentation of the data on the observed phenomena). Additionally, those various approaches in real-world scenarios are often combined – for example, an output from a forecast can become input for a simulation model, and the output from the simulation could become an input for a decision tree. Interested reader can also see [6, 9] for a wider context of sequential decisions problems.

The present book is organized into the following chapters:

- Chapter 2 presents a comprehensive introduction to building decision trees with `SilverDecisions`.
- Chapter 3 discusses the mathematical foundations of analyzing decision trees using `SilverDecisions`.
- Chapter 4 presents more advanced scenarios of working with decision trees, including *real option valuation*, *multiple criteria analysis*, and including *time factor* in the analysis of decision trees.
- Chapter 5 presents several case studies that `SilverDecisions` users might want to follow to understand how the software can be used in practice.

The `SilverDecisions` software can be used for free and without installation by using a web browser. You can find it at http://silverdecisions.pl/.

Book guide and conventions

In this book, we use different formatting to signal actions that the user can take

⌨	represents a keyboard action
🖱	represents a mouse action
'Label'	represents a text visible in `SilverDecisions` application
`text`	represents a text entered by a user to `SilverDecisions`
`code`	represents JavaScript code for programming decision trees (advanced software functionality discussed in section 4)

Acknowledgments

We express our gratitude to the members of our `SilverDecisions` team: Michał Wasiluk (developer) and Marcin Czupryna, Timothy Harrell, and Anna Szczurek (testing and documentation).

The initial `SilverDecisions` development process has received funding from the EU Horizon 2020 research and innovation program under grant agreement No 645860 as a part of the ROUTE-TO-PA project http://routetopa.eu/ to support decision processes visualization within the Social Platform for Open Data (http://spod.routetopa.eu/).

The development of the currently available version of the software (version 1.2) has been financed by the Polish National Agency for Academic Exchange.

The book is financed by the Polish National Agency for Academic Exchange.

Chapter 2

Fundamentals

Przemysław Szufel

The goal of this chapter is to provide a quick overview of the key functionalities of `SilverDecisions` as well as a discussion of more advanced features related to support for programming decision trees.

We start with an introductory tutorial showing how to implement a simple decision tree. This includes operations such as adding and removing nodes and basic decision tree settings (Section 2.1). Subsequently, we move to an overview of options available in the application menus (Section 2.2). This includes a thorough overview of tree layout settings, visualization options, and optimization modes (however, without multi-criteria decision modeling, which will be discussed later in the book in Section 4.2). Finally, in Section 2.3 we present decision tree programmability options offered in the software. We show how to define variables at various decision tree scopes along with custom functions that can be used subsequently across the decision tree.

2.1 Building your first tree

We will follow the well-established tradition for software books by starting with a simple yet usable working example.

2.1.1 Software requirements

`SilverDecisions` is accessible through a web page http://silverdecisions.pl/. The application can be launched inside the web browser by pressing the `Run` button on the project's homepage.

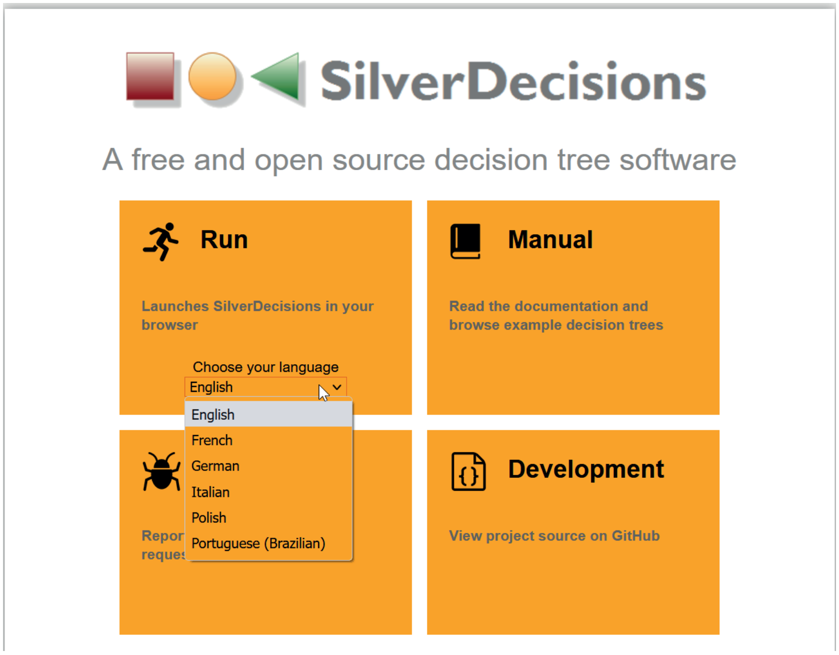

SilverDecisions
A free and open source decision tree software
Run
Launches SilverDecisions in your browser
Choose your language
English
English
French
German
Italian
Polish
Portuguese (Brazilian)
Manual
Read the documentation and browse example decision trees
Repor
reques
Development
View project source on GitHub

Figure 2.1: On the welcome screen, you can select the language version of the software before running it.

The software can be run[1] under Windows, Linux, and OS X. The recommended web browser is Google Chrome - a browser that runs across all major operating systems and platforms. In this book, for all illustrations, we use `SilverDecisions` version 1.2 being run on Google Chrome version 93 on Windows 10 operating system - this is the reference execution environment for `SilverDecisions`. The full list of currently supported web browsers is on the project's website. Note that software is not fully supported in non-desktop environments (e.g., mobile phones or tablets).

If you find a bug in our software, you are welcome to submit a new issue on the web page https://github.com/SilverDecisions/SilverDecisions/issues. Please note that you might be using a newer software version than the one described in this book – the up-to-date documentation is available at https://github.com/SilverDecisions/SilverDecisions/wiki/Documentation.

[1] Please note that the `SilverDecisions` may run on tablets and mobile devices, but the functionality is limited to read-only - tree editing might be not available.

2.1.2 First steps

In this tutorial, we will create a simple decision tree representing a decision problem of an investor who is deciding to locate her money in one of two startups. Our investor has only a budget for one investment, and her goal is to maximize the expected profit (in other words, our investor is risk-neutral).

To start creating a tree, simply click the right mouse button[2] anywhere in the white plotting canvas. The context menu will pop up. Let us choose the *'Add Decision Node'* option from the context menu, and the first node will be added to the canvas – see Figure 2.2.

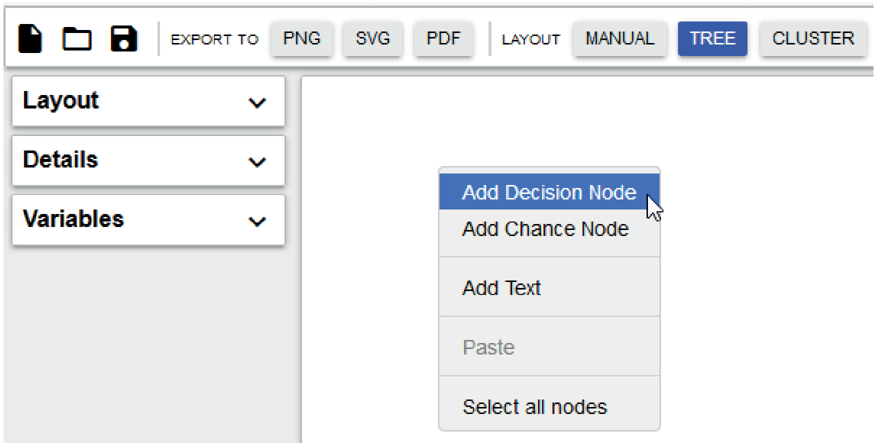


Figure 2.2: In order to start, 🖱 right-click the white canvas to add a node to the tree.

Observe that there are red exclamation marks – *'!!'* (Figure 2.3) marks at the top-right side of the decision node. This means that the tree is not proper (it is not allowed to have a decision node without child nodes). This can be achieved either by 🖱 left-clicking on the decision node again and pressing ⌨ Ctrl-Alt-C or 🖱 right-clicking on the decision node and selecting the *'Add Chance Node'* option from the context menu. In this tutorial, we add two chance nodes – see Figure 2.3.

We add two-terminal node to each chance node. As the result, a tree that can be seen in Figure 2.5, has been constructed.

[2] The context menu may also be opened by 🖱 double-clicking anywhere on the canvas or on a node – in this way, the software can be used on devices without the right mouse button. Additionally, on mobile devices, a long press can be used instead of 🖱 right-button. For the rest of the book, we will assume that a computer mouse is being used.

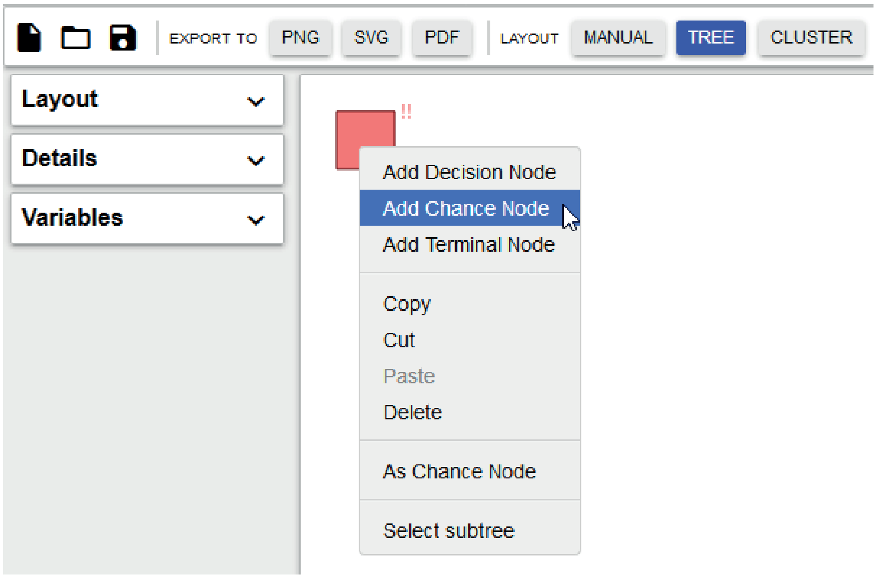


Figure 2.3: Once the decision node is created, 🖰 right-click on the decision node to add a chance node to the tree.

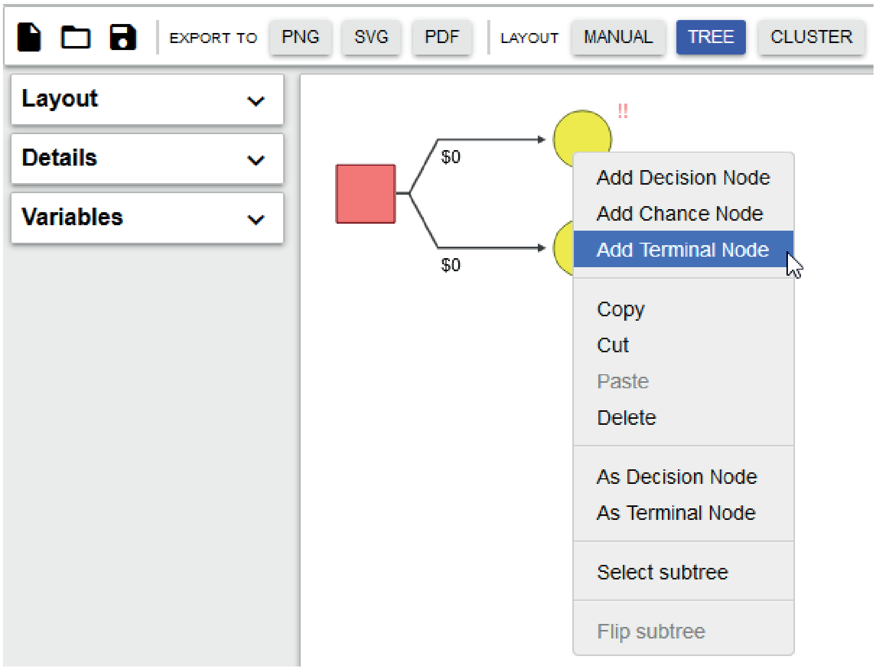


Figure 2.4: Once the chance node is created, 🖰 right-click on the chance node to add a terminal node to the tree.

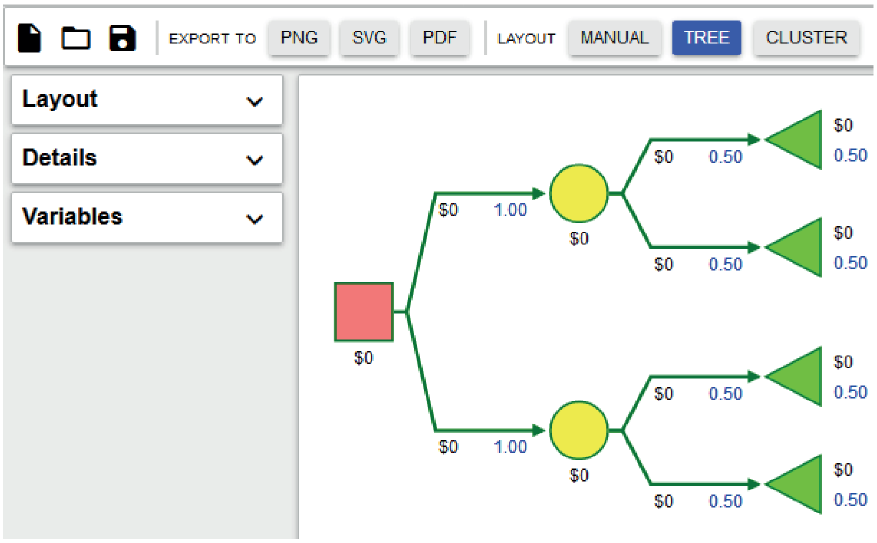


Figure 2.5: The final example tree contains one decision node, two chance nodes, and four-terminal nodes.

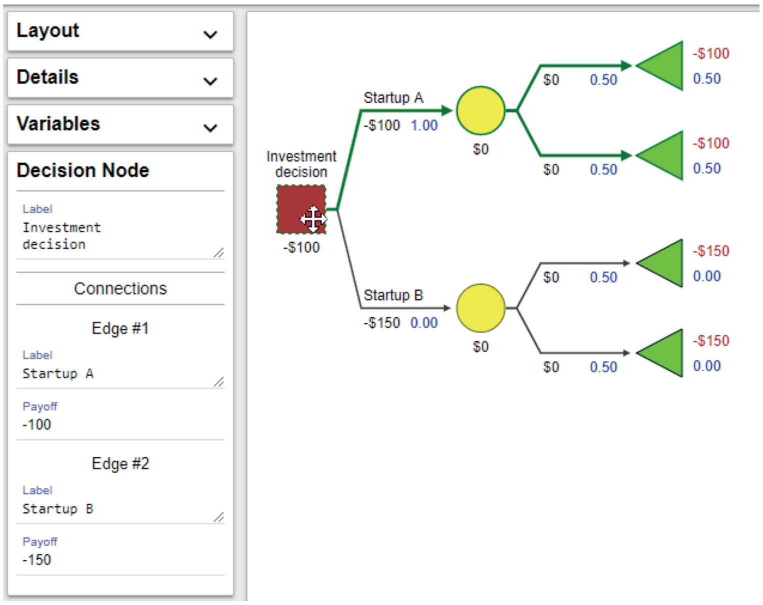
Now terminal nodes can be added to the constructed tree – see Figure 2.4.

Once we have understood how to add nodes to the tree, the next step is to learn how to edit the edges. Note that edges are added automatically where child nodes are being added to a parent node.

Let us start by adding to the decision tree node labels, as well as edges' labels and payoffs. Firstly, 🖱 left-click on the decision node to define decision labels. In the left panel, where the Decision Node dialog appears, enter the text `Investment decision`. Note that you can press ⌨ Enter after the word `Investment` to add a new line to the text.

Now it is time to edit edge labels. Note that the decision node remains selected. Therefore, in the left panel, we can see the list of all edges coming from it. We can edit their payoffs and labels. Type the label `Startup A` and the payoff `-100` in the *'Edge #1'* boxes. Then complete the *'Edge #2'* boxes also. Note that we may use ⌨ Tab to move to the next box. Similarly now, we type the label `Startup B` and the payoff `-150` in the Edge #2 boxes.

Note that another option to edit edge labels and payoffs is to 🖱 left-click an edge. The edge then becomes highlighted, and in the left panel, an Edge dialog appears. The properties of the edge (label and payoff) can be modified there.



Layout

Details

Variables

Decision Node

Label
Investment
decision

Connections

Edge #1

Label
Startup A

Payoff
-100

Edge #2

Label
Startup B

Payoff
-150

Figure 2.6: Decision labels can be defined after left-clicking (selecting) the decision node (note that the decision node has been marked as selected).

Please note that the decision tree that has been just created will find the optimal decision path as soon as all the necessary parameters are set (the optimal decision is marked with a thicker line).

Let us now define the probabilities. Firstly, left-click on the top chance nodes to see its dialog on the left. Right now, you can see a new option for edges to be changed: probability. Initially, the probability values on the left panel for any edge coming out of the chance node are by default set to the `#` sign. On the other hand, the probability displayed on the diagram is equal to 0.5. `#` is the default parameter for specifying probabilities: it is automatically assigned to each edge originating from the chance node – see Figure 2.7. Its exact value is calculated in such a way that the probabilities of all the edges coming from a given chance node sum up to 1. In our example, two edges are coming from the chance node, so the probability for each of them is computed to be 0.5. The user can enter their values into the probability boxes. However, `#` is the default value for probability definition. For each decision outcome, we assign a label along with either probability value or `#` – seeFigure 2.8.

Note that arithmetic expressions are allowed when specifying probabilities and payoffs. Instead of typing `100` as the payoff for the first terminal node from the top, the expression `(0 + 100)` may be entered and

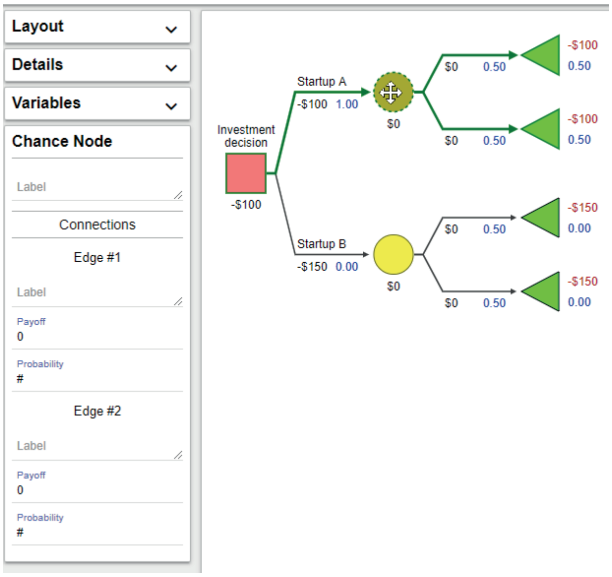

Layout
Details
Variables
Chance Node
Label
Connections
Edge #1
Label
Payoff
0
Probability
#
Edge #2
Label
Payoff
0
Probability
#

Figure 2.7: Probabilities can be defined after left-clicking (selecting) the chance node. Initially, the probabilities are represented with the `#` sign which means *undefined* – probability values are evenly distributed across all edges – hence the value visible at the edges is 0.5.

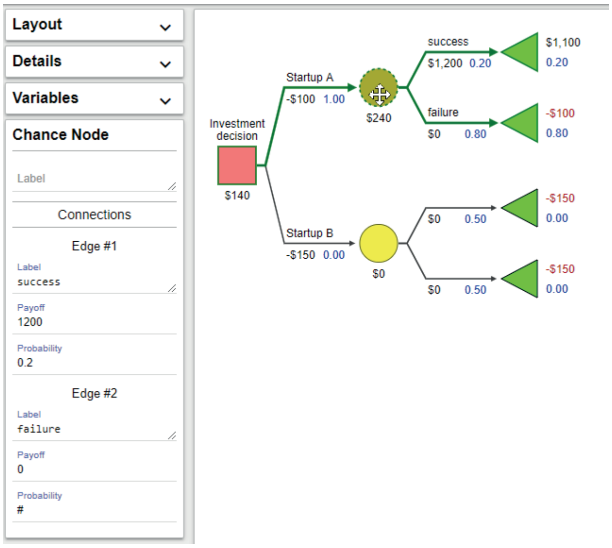

Layout
Details
Variables
Chance Node
Label
Connections
Edge #1
Label
success
Payoff
1200
Probability
0.2
Edge #2
Label
failure
Payoff
0
Probability
#

Figure 2.8: For each outcome in the chance node we define a label. Special expressions `#` can be used to automatically ensure that the probabilities sum up to 1.

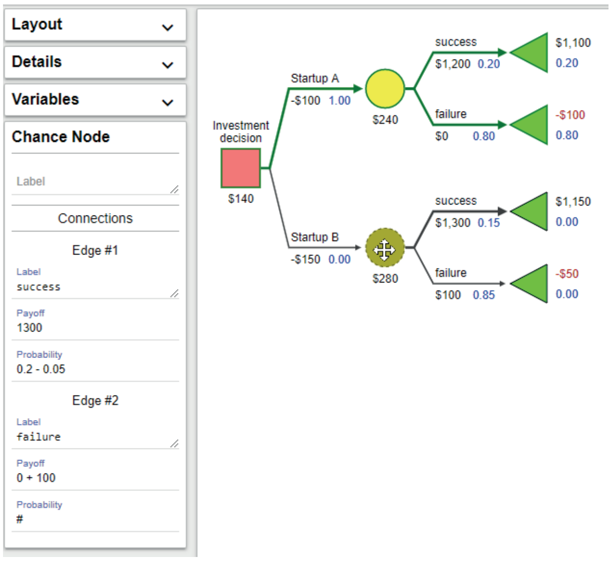

Layout
Details
Variables
Chance Node
Label
Connections
Edge #1
Label
success
Payoff
1300
Probability
0.2 - 0.05
Edge #2
Label
failure
Payoff
0 + 100
Probability
#

Figure 2.9: Arithmetic expressions are allowed when specifying probabilities and payoffs. Similarly, as for the `#` sign, expressions are presented in the boxes in the left panel, but computed values are displayed on the diagram.

`0.2-0.05` can be typed instead of `0.15`. Similarly, as for the `#` sign, expressions are presented in the boxes in the left panel, but computed values are displayed on the diagram – see Figure 2.9. To check if the value on the diagram is a number, `#` sign, or expression, one may select the proper node/edge and look at the left panel or move the mouse cursor over the probability/payoff value on the diagram. Additional information about the probability and payoff format as well as on `#` parameter can be found in subsection 2.1.3.

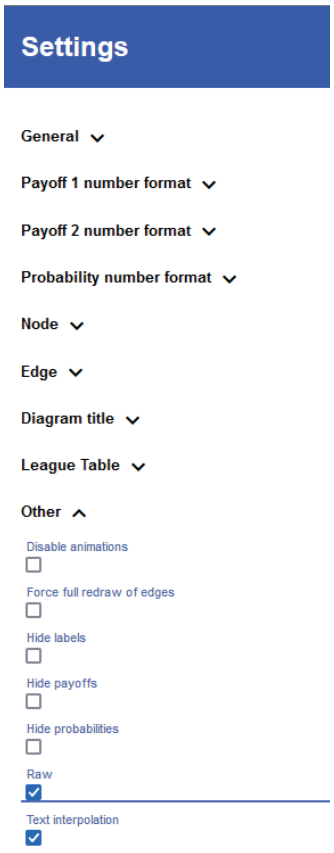
Finally, let us add the tree title. Simply left-click on the Details option in left panel and enter the text `My first investment decision model` in the title box. A title of the decision tree appears in the plotting canvas. Observe that the decision node we have edited has now become deselected, and its properties dialog has disappeared from the left panel.

The final decision tree is presented in Figure 2.11. Let us now discuss the numbers that we see. The thick green line recommends the "invest into *'Startup A'*" alternative with the expected pay-off for this decision of $140. In case of success, the *'Startup A'* will be worth $1,200. This means that the net profit is $1,100, and this value is visible on the right-hand side of the first terminal node. In case of startup failure, its shares are worth nothing. This means that the net result is $−100, and again, this value is visible in the tree. The values for other terminal nodes are calculated analogously. Each terminal node also has a value of probability 0.2 for the first terminal node, 0.8 for the second, and zeros for the two other nodes. This value represents the probability of the decision process ending at a particular terminal node, assuming that the decision-maker follows the recommendations in the decision tree. Chance nodes represent uncertainty. In our example, for each chance node, there is the expected value of income for that node – \$240 for the *'Startup A'* and \$280 for the *'Startup B'*. Making a decision (here investing in the stock) yields some costs (\$100 for the Startup A and \$150 for the Startup B). While selecting the optimal decision, the costs need to be deducted from the income. Finally, the expected value of the outcome for each decision can be calculated. The expected profit from investing in Startup A is $240 - 100 = 140$, and then for the Startup B is $280 - 150 = 130$. The software selects the decision with the highest expected value and highlights it with the green color. Note that it is possible that several choices can yield the same expected profit – in that case, all such decisions will be highlighted.

2.1.3 Important notes

When working with probabilities, please bear in mind the following facts:

- Payoffs and probabilities are always displayed on the tree diagram as numbers representing computed values (e.g., the result of arithmetic expressions or computed probabilities values for `#` parameters – both values are displayed as numbers). On the other hand, arithmetic expressions or the `#` sign may either be present in the left panel when an appropriate node/edge is selected or on the tree diagram after moving the mouse cursor over the probability/payoff value.
- If there are several branches coming out of a decision node that are equally good (have the same expected value), all of them will be highlighted in green by default as optimal, and all of the edges coming from this decision node that are on the optimal path will have their probability set to 1.00.
- in order to see formulas for tree nodes use the *'raw'* model setting – seeFigure 2.10.



Settings

General

Payoff 1 number format

Payoff 2 number format

Probability number format

Node

Edge

Diagram title

League Table

Other

Disable animations ☐

Force full redraw of edges ☐

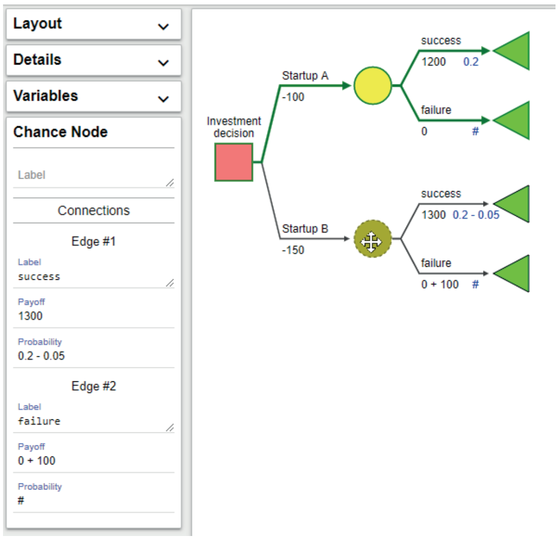
Hide labels ☐

Hide payoffs ☐

Hide probabilities ☐

Raw ✓

Text interpolation ✓



Layout

Details

Variables

Chance Node

Label

Connections

Edge #1

Label: success

Payoff: 1300

Probability: 0.2 - 0.05

Edge #2

Label: failure

Payoff: 0 + 100

Probability: #

(a) Application settings – raw tree view (b) Application settings – raw tree view

Figure 2.10: Raw tree view enables to see formulas on all tree nodes – it can be switched on and off in the application settings.

2.1.4 Shortcuts for creation of a decision tree

Below we present a list of shortcuts useful when building a decision tree. For the full list of keyboard shortcuts, see Appendix A.

- create the first node
  - 🖱 right click on a node and select either *'Add Chance Node'* or *'Add Chance Node'* option.
- add a Chance node to a tree
  - 🖱 right click on a node and select the *'Add Chance Node'* option or;
  - 🖱 left-click on a node and press ⌨ Ctrl-Alt-C
- add a Decision node to a tree

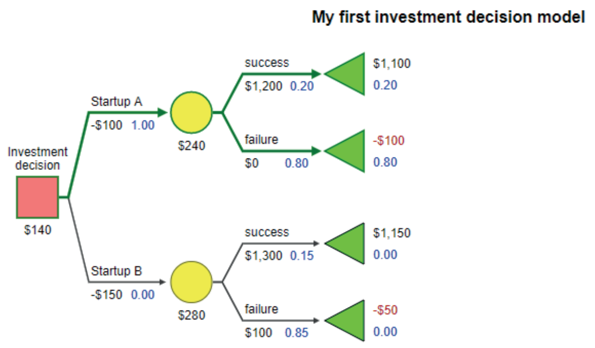


Figure 2.11: The decision tree created with the tutorial recommends the first option with the expected payoff of $140.

  - right click on a node and select the *'Add Decision Node'* option or;
  - left-click on a node and press Ctrl-Alt-D
- add a Terminal node to a tree
  - right click on a node and select the *'Add Terminal Node'* option or;
  - left-click on a node and press Ctrl-Alt-T

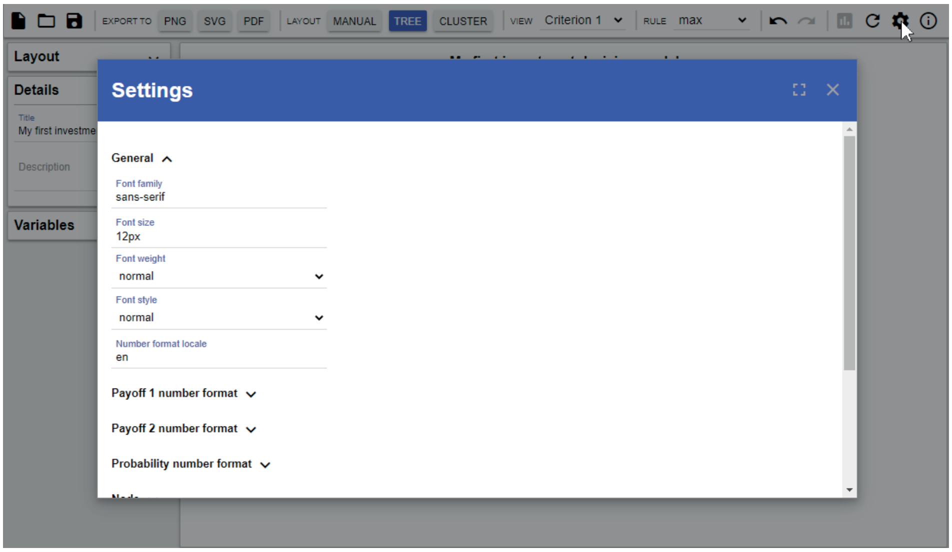
2.2 SilverDecisions: Tips & Tricks

In this section we present an overview of options available in the application menus.

2.2.1 SilverDecisions application settings

When you left-click the *'SETTINGS'* button in the top right corner, the following options are available (see Figure 2.12). In the General section, the font family and size, and the number format locale may be changed by typing values into the appropriate boxes. Try typing `Arial black` to the *'Font family'* box and `9px` to the *'Font size'* box. Note that the text on the tree will change. Number format locale requires typing the language tag to adjust to the desired format. Font weight and font size may be changed by selecting one of the options from the context menu.

A particularly interesting setting is the number format locale. Depending on this setting the value and currency will be differently displayed for example the value of `$1,300.5` (`en` locale)) can be displayed as `1300,5 USD` (`pl` or `fr` locales)) or `1.300,5 USD` (`it` locale)) or `1.300,5 $` (`de` locale)) or `US$ 1.300,5` (`pt` locale)). The full technical list of possible locale values is available at https://datatracker.ietf.org/doc/html/rfc5646.



Settings

General

Font family
sans-serif

Font size
12px

Font weight
normal

Font style
normal

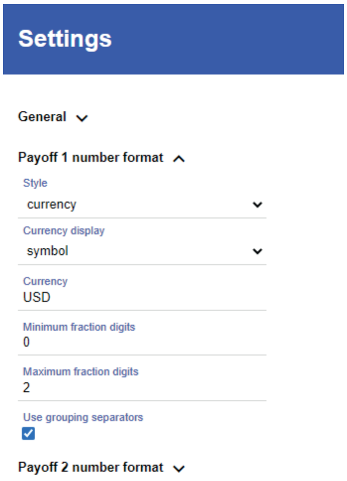
Number format locale
en

Payoff 1 number format

Payoff 2 number format

Probability number format

Figure 2.12: Settings screen can be opened by left-clicking the appropriate button at the top-right corner.



Settings

General

Payoff 1 number format

Style
currency

Currency display
symbol

Currency
USD

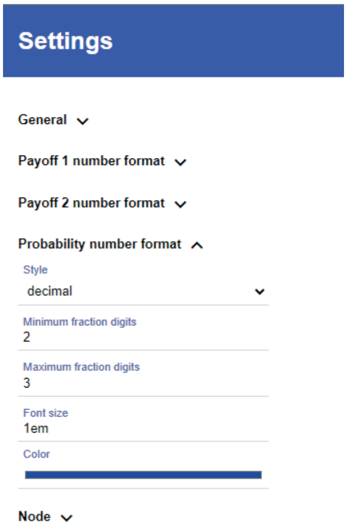
Minimum fraction digits
0

Maximum fraction digits
2

Use grouping separators
☑

Payoff 2 number format

(a) Setting for the payoff values can be customized.



Settings

General

Payoff 1 number format

Payoff 2 number format

Probability number format

Style
decimal

Minimum fraction digits
2

Maximum fraction digits
3

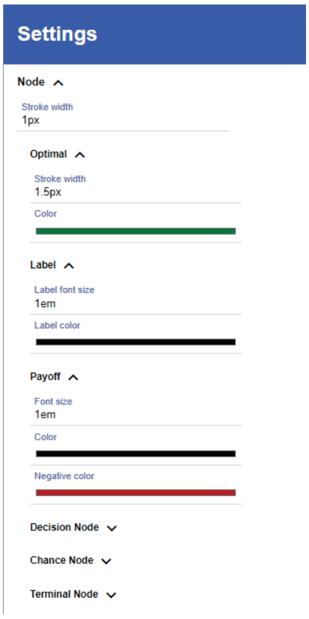
Font size
1em

Color

Node

(b) Setting for the probability values can be customized.

Figure 2.13: Setting options for payoff and probability.


Settings

Node
Stroke width
1px

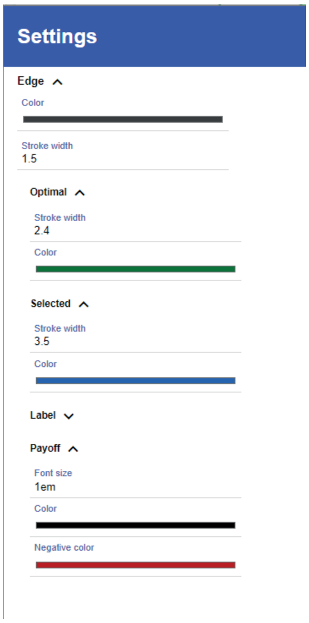
Optimal
Stroke width
1.5px
Color
🟩

Label
Label font size
1em
Label color
⬛

Payoff
Font size
1em
Color
⬛
Negative color
🟥

Decision Node
Chance Node
Terminal Node

(a) Setting for the nodes can be customized.


Settings

Edge
Color
⬛
Stroke width
1.5

Optimal
Stroke width
2.4
Color
🟩

Selected
Stroke width
3.5
Color
🟦

Label

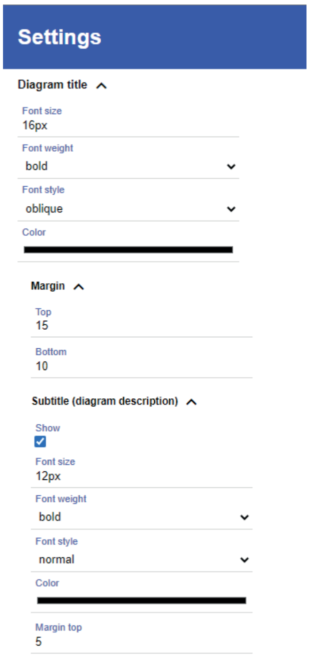
Payoff
Font size
1em
Color
⬛
Negative color
🟥

(b) Setting for the edges can be customized.

Figure 2.14: Setting options for nodes and edges.

Please note that currency formatting must be configured independently of the chosen language tag. Moreover, *'number format locale'* settings refer only to the text format displayed on the tree graph. In the left panel, a dot . is the expected decimal separator regardless of the chosen standard locales.

In the following two sections, the Payoff number format and Probability number format may be adjusted, including number style, fractions digits, or color – see Figure 2.13a. For payoffs, you can also change the grouping separators and currency options, including how the currency is displayed (symbol/code/name). Probability numbers may be displayed as decimals or percentages in different sizes or colors. Please note that two payoff configurations are available (*'Payoff 1 number format'* and *'Payoff 2 number format'*). By default, the visible configuration is configuration 1. The other configuration is used when the *'VIEW'* option in the main application menu is changed either to *'Criterion 2'* or *'Two criteria'*. The multi-criteria decision-making is described in Section 4.2.



Settings

Diagram title

Font size
16px

Font weight
bold

Font style
oblique

Color
⬛

Margin

Top
15

Bottom
10

Subtitle (diagram description)

Show
☑

Font size
12px

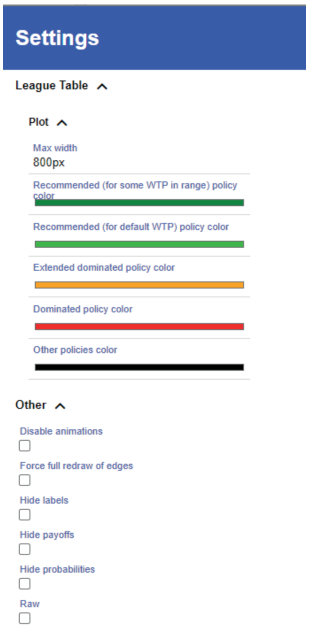
Font weight
bold

Font style
normal

Color
⬛

Margin top
5

(a) Setting for the diagram title can be customized.



Settings

League Table

Plot

Max width
800px

Recommended (for some WTP in range) policy color
🟩

Recommended (for default WTP) policy color
🟩

Extended dominated policy color
🟧

Dominated policy color
🟥

Other policies color
⬛

Other

Disable animations
☐

Force full redraw of edges
☐

Hide labels
☐

Hide payoffs
☐

Hide probabilities
☐

Raw
☐

(b) Setting for the league table along the "other".

Figure 2.15: Setting options for diagram title and league table.

The following settings are available for the payoffs formatting:

- *'Style'*: `currency` (default) or `decimal` can be chosen. Changing the style to `decimal` can be used to hide the currency symbol
- *'Currency display'*: currency symbol (default), code, or name can be chosen. The currency name depends on the *'locale'* setting discussed above. For example, for the `en` locale symbol, code and name will be displayed as `$`, `USD`, and `US dollars` respectively. For the `de` locale, the currency name will be displayed as `US-Dollar`
- *'Currency'*: a specified currency may be set by typing the currency code, `USD` by default (other typical currency codes are e.g.: `EUR`, `JPY`, `RUB`, `PLN`); the display of the currency symbol depends on the chosen language locale
- *'Minimum fraction digits'*: can be changed by typing the value or by clicking on the spinner on the right; the value 0 is set by default. However, if you want to see values of $3.50 instead of $3.5, you might consider setting this to 2.

- *' Maximum fraction digits'*: can be changed by typing the value or by clicking on the spinner on the right; 2 by default.
- *'Use grouping separators'*: to remove the thousands of separators, just uncheck the box

The next group of settings refers to the probability – see Figure 2.13b. Those options include:

- *'Style'*: `decimal` (default) or `percent` can be chosen.
- *'Minimum fraction digits'*: can be changed by typing the value or by clicking on the spinner on the right;
- *' Maximum fraction digits'*: can be changed by typing the value or by clicking on the spinner on the right; 2 by default.
- *'Font size'*: is used to set the size of probability font so those values can be clearly distinguished from others on the diagram
- *'Font color'*: is used to set the color of probability font so those values can be clearly distinguished from others on the diagram

The *'Node'* part enables the setting of different graphical options for different nodes: decision ones, chance ones, and terminal ones – see Figure 2.14a. Additionally, unique stroke width and color can also be adjusted for all the nodes on the optimal path. Payoff value colors are adjusted separately for positive and negative values and hence are going to change accordingly depending on the values in the decision tree. Similarly, the stroke color and width can automatically change for nodes that are located on the optimal path.

In the *'Edge'* section, analogous modifications can be made. The color of an edge changes dynamically depending on whether it is located on the optimal path (the default color is green) or not (the default color is black) – see Figure 2.14b. When editing a tree, an edge can be selected to edit its properties. Again, an edge changes color also for the time it is selected for editing.

The *'Diagram title'* section includes standard font options (size, weight, style, and color) as well as settings related to margin and subtitle options – see Figure 2.15a. The margin options make it possible to set the distance between the title and other diagram elements. The subtitle options make it possible to decide whether the subtitle is visible as well as to set the standard font options for the subtitle along with the distance between the title and the subtitle.

The *'League Table'* options allow to configure the size (width) and the colors for the league table. The *'Other'* options allow you to control how the tree is repainted (their main use is on system/browser configurations that have problems with correct handling of JavaScript repaint events) or disable of presentation of different elements of the tree. These options are particularly useful when you want to export the tree only with a subset of elements – it is possible to hide the labels, payoffs, and probabilities – see Figure 2.15b.

2.2.2 Tree branch folding

Tree folding allows one to navigate through large decision trees more easily and focus on branches of interest at a particular point in time. Whenever you right-click any Chance or Decision node, there is an option to Fold that folds down (hides) the subtree below that node. Once a node is in a folded state, a plus sign [+] is visible next to the node – see Figure 2.16. Clicking the [+] sign will unfold the subtree. When a node is in the folded state, no new child nodes can be added – it needs to be unfolded first.

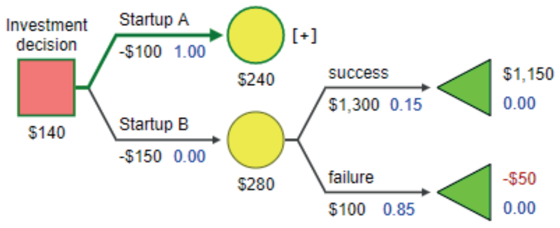


Figure 2.16: A partially folded tree – a plus sign [+] is visible in the first branch. Clicking it will unfold the tree.

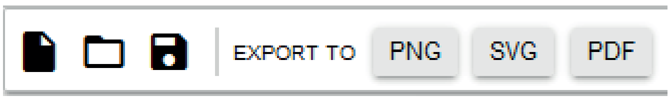


Figure 2.17: Toolbar for saving and loading decision trees.

2.2.3 Optimization modes

The software supports two-criteria decision-making. A common practical example would be evaluating an expected profit versus risk. In the *'RULE'* menu options depend on whether a single or multiple-criteria tree has been selected – see 2.18.

The following options are available for a single-criteria decision tree (more information on how decisions in standard decision trees are estimated can be found in Section 3.1):

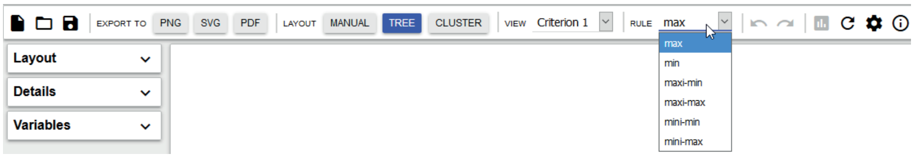
- *'max'* – *maximize* the expected pay-off value in the decision tree
- *'min'* – *minimize* the expected pay-off value in the decision tree
- *'maxi-min'* – assume the worst-*minimal* (pessimistic) outcome at each chance node and select the decision with the highest (*maxi*) outcome
- *'maxi-max'* – assume the best-*maximal* (optimistic) outcome at each chance node and select the decision with the highest (*maxi*) outcome
- *'mini-min'* – assume the best-*minimal* (optimistic) outcome at each chance node and select the decision with the lowest (*mini*) outcome
- *'mini-max'* – assume the worst-*maximal* (pessimistic) outcome at each chance node and select the decision with the lowest (*mini*) outcome

The following options are available for a multi-criteria decision tree (Multicriteria optimization is discussed in Section 4.2):

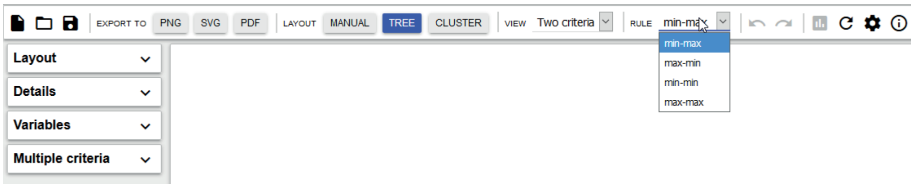
- *'min-max'* – *minimize* the expected pay-off value of the first criteria and *maximize* the pay-off value of the second criteria
- *'max-min'* – *maximize* the expected pay-off value of the first criteria and *minimize* the pay-off value of the second criteria
- *'min-min'* – *minimize* the expected pay-off value of the first criteria and *minimize* the pay-off value of the second criteria
- *'max-max'* – *maximize* the expected pay-off value of the first criteria and *maximize* the pay-off value of the second criteria

2.2.4 Saving and exporting a decision tree

The first toolbar actions (Figure 2.17) correspond to creating a new diagram (note that this option clears the whole diagram), opening an existing diagram, or saving the current one. By 🖱 left-clicking on *'Open an existing diagram'*, you can load a previously created tree from your disk. Note that only those files saved in JSON format may be loaded into the application. To save your tree, just 🖱 left-click on the *'Save current diagram'* – the tree is then automatically saved to disk in JSON format to a download folder specified in your browser. The tree is saved in tree metadata in its layout mode with its layout parameters, so whenever reopened, we should be able to continue work where we have finished it. Note that the naming convention for a file with the decision tree is the following: `decisiontree@2021.09.10_14.44.55.json` – the first part of the name is the word `decisiontree`, followed by the character `@`, then date, time, and the `.json` extension.



(a) The *'RULE'* menu in single criteria view.



(b) The *'RULE'* menu in two-criteria view.

Figure 2.18: The *'RULE'* menu contents change on whether a single or two optimization criteria have been selected.

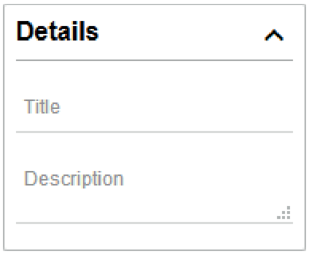


Figure 2.19: Editing labels for the decision tree.

The second toolbar panel (Figure 2.17) enables the export of the diagram representing the decision tree to a file in either PNG, SVG, or PDF format. SVG is the recommended export format as it guarantees the same image as seen in the browser and is the fastest way of exporting. PDF export is performed via an external converter. It is not guaranteed that the exported diagram looks the same as in the browser: fonts that are not on the PDF renderer server will be substituted by alternatives in the exported PDF file. Exporting preserves the selection: if the nodes or edges are selected at the time of exporting the diagram, they will also appear selected in the exported file.

2.2.5 Tree details

The *'Tree details'* section makes it possible to set the title and the description of the decision tree – see Figure 2.19.

2.2.6 Managing tree edit process

The application supports the standard editing features that we can expect from any editor. This includes selecting copy-pasting nodes and the Undo/redo functionality. A set of tree nodes can be selected by left-clicking and holding a mouse. It is possible to move the whole tree or any of its nodes by simply selecting them and moving the mouse cursor. To copy or paste the nodes, we can use keyboard shortcuts (ctrl+c, ctrl+v) or choose the corresponding options from the left-click mouse context menu. Any selected node gets copied along with all its subsequent nodes. This means that by copying any parent node, the whole of its subtree gets copied. It is possible to copy and paste multiple disconnected subtrees. Any selected nodes or subtrees can be copied to the canvas (as single nodes/subtrees) or the diagram structure. To paste copied nodes onto the canvas, just left-click anywhere on the canvas (nodes become deselected) and choose the paste option from the left-click context menu. Please note that the ctrl+v shortcut works only for pasting the nodes/subtrees to any part of the tree graph. To do this, just select an appropriate node and right-click and select *'Paste'* or press ctrl+v. The copied subtree should be added to this node.

2.2.7 Context menu actions

We can right-click on nodes, edges, or the white canvas to bring up the context menu that makes it possible to interact with the tree. Here is the list of context menu actions when an element is right-clicked (we do not comment on self-explanatory actions):

- White canvas: *'Add decision node'*, *'Add chance node'*, *'Add text'* – adds a label detached from the tree, *'Paste'*, *'Select all nodes'*
- Decision node: *'Add decision node'*, *'Add chance node'*,*'Add terminal node'* – any added nodes will be attached to the current node, *'Copy'*, *'Cut'*, *'Paste'*, *'Delete'*

For the tree flipping, see Section 4.1.

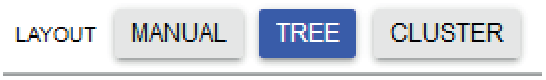


Figure 2.20: Three layout modes are available to manually edit the tree, the *'MANUAL'* option should be selected.

2.2.8 Layout options

There are three layout options on the top toolbar: *'MANUAL'*, *'TREE'* (default) and *'CLUSTER'* – see Figure 2.20. By default, the diagram has a *'TREE'* layout. It is recommended to construct and develop the tree in this layout.

The *'MANUAL'* layout is preferred when final corrections are made. In this mode, the user can manually place the nodes. Just left-click on any node, hold the button and move that node. Then, let us try to change the position of the whole tree. Place the mouse cursor somewhere on the canvas, left-click-and-hold-down to select all nodes of the tree. When all nodes of the tree are selected, the colors of its nodes become darker. Now, just left-click on any node, hold the button and move the tree. Note that we are allowed to position the nodes and the tree only if the *'MANUAL'* layout is selected. *'TREE'* and *'CLUSTER'* are automatic layouts of the tree nodes aligned to the left or to the right, respectively. By changing the layout again to *'TREE'* or *'CLUSTER'*, your decision tree will be automatically aligned once more. The *'TREE'* layout is a standard layout normally used throughout this book. The *'CLUSTER'* layout automatically moves all terminal nodes to the right and thus makes it easier to compare them for some trees.

Depending on the width of text labels, additional fine-tuning of the tree layout is often required to get a clear, readable tree. Hence, additional layout options may be found on the left in the Layout panel – see Figure 2.21. Here we can change the horizontal and vertical margin by simply left-clicking on the slider and moving it to the left or right. If we move the horizontal margin slider to the right, we will see that the left canvas margin gets wider. Try moving the vertical margin slider to the right – see how the top margin gets wider. Within the layout panel, the node size and edge slant can also be controlled. By moving the node size slider, you can scale the nodes, and by moving the edge slant slider, you can control the maximum slant for plotting the sloping part of the edge. For the *'TREE'* and *'CLUSTER'* layout modes, you can also see the tree height and width settings in the Layout panel on the left.

2.3 Working with variables and functions

In this section, we discuss SilverDecisions' support for variables and functions defined within a decision tree. SilverDecisions' computational mechanism is based on the *math.js* library that is available at http://mathjs.

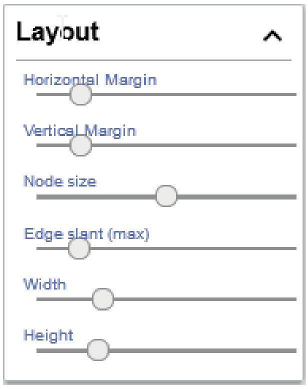


Figure 2.21: Layout settings in the left application panel.

org/. The *math.js* library is a very powerful computational tool, and our goal is two show how it can be used for decision analysis rather than discuss its full functionality. For a full reference of *math.js* functionality, you are advised to see its documentation at http://mathjs.org/docs/.

2.3.1 Working with variables

In SilverDecisions, any payoff or probability field can contain expressions next to numeric values. Have another look at Figure 2.9 discussed previously in Section 2.1.

More specifically, payoffs can be any proper single line expression of maths.js, for example:

```
exp(a)^2
```

Probabilities can be entered using the special sign `#` (as it was noted earlier `#` states that probability values should be evenly distributed across all edges) or any proper single line expression maths.js, example:

```
random()
```

This command will insert a random value between 0 and 1 in the field. This value will be regenerated each time the tree is recomputed (e.g., by clicking the '↻ *Recompute'* button at the taskbar). We have shown such a sample tree in Figure 2.22. More information on the available random functions in SilverDecisions can be found in Appendix A (the software extends the standard functionality of math.js in this regard). Please note that randomness can be later used as one of the ways sensitivity analysis, discussed in Section 3.3. However, for now, we ignore the distribution and simply perform only one computation of tree output.

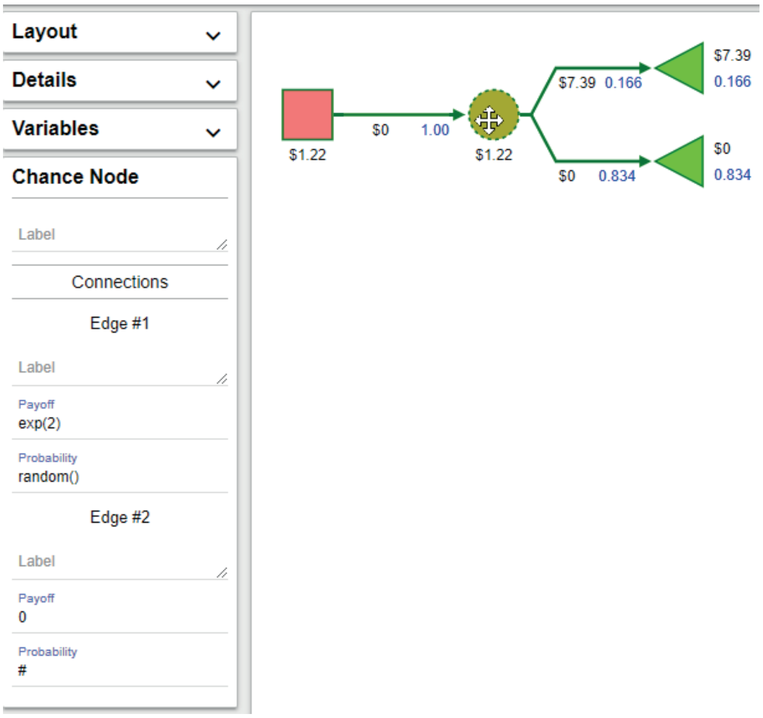


Figure 2.22: Different math.js expressions and functions can be contained within the decision tree.

Moreover, it is possible to define variables along with the functionality of the math.js library. The variable declaration can look like this (see Figure 2.23):

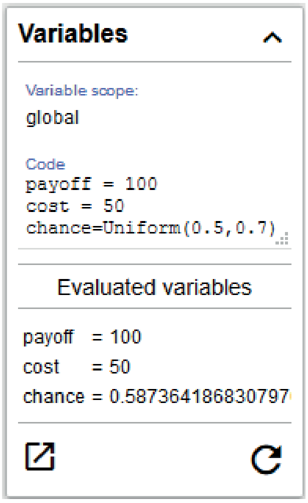
```
payoff = 100
cost = 50
# this value will be generated randomly
success_prob = Uniform(0.5, 0.7)
```

Note that lines starting with `#` are being treated as comments. This is different behavior than one observed in a probability field where the `#` character is used to denote putting a value that will sum up to one.

Like in any other language, math.js makes it possible to define vectors and matrices:

```
  v = [1, 2, 3]
  a = [1, 2; 3, 4]
```

The elements of a vector or an array are references similarly to other languages as, e.g., `v[2]` or `a[2, 1]`.



```
payoff = 100
cost = 50
chance=Uniform(0.5,0.7)
```

```
payoff   = 100
cost     = 50
chance = 0.587364186830797
```

Figure 2.23: Editing variables with SilverDecisions.

In general, any expressions allowed by http://mathjs.org/ can be used as code in SilverDecisions. In particular, functions defined in math.js module math are available, but constants are not. It should be noted that expressions can be used in the following places:

- payoff and probability values
- code blocks (global and defined in chance and decision nodes)
- all text labels in a decision tree via the interpolation mechanism discussed in Section 2.3.5.

The settings for variable definition are available all time on the left `Variables` panel (see Figure 2.23) and contains the following elements:

- *'Variable scope'* can be other *'global'* (when the white canvas was clicked before editing variables) or *'selected node and subtree'* when any node has been selected. For a discussion on how variable scoping works, see Section 2.3.2.
- *'Code'* – a valid math.js code used to generate variables. The recommended style is to place a single variable in each line. It is also possible to use semicolons to separate statements and hence to write `x=1; y=2`.
- *'Evaluated variables'* this contains a list of all variables that were obtained by evaluating the code
- *'Open'* dialog button that shows the settings on a new bigger window
- *'Recalculate'* button that does the recalculation of the entire tree (including variable values) and works in the same way that the '↻ *Recompute'* button in the application taskbar.

Note that payoff and probability definitions are not allowed to introduce new variables. Any code (in particular defining variables or functions) is allowed in code blocks. Code blocks can be edited in the left panel. If no node is selected, the global code block is edited. If the node is selected, the code for this node is edited. You can press the Open dialog box (on the bottom left of the Variables section in the left pane); a Variable definitions dialog box that allows for simpler work with longer definitions of variables.

Since we know how variables work, let us describe the recalculation mechanism more precisely now. Recalculation (defined as an evaluation of all code in code blocks) is triggered when:

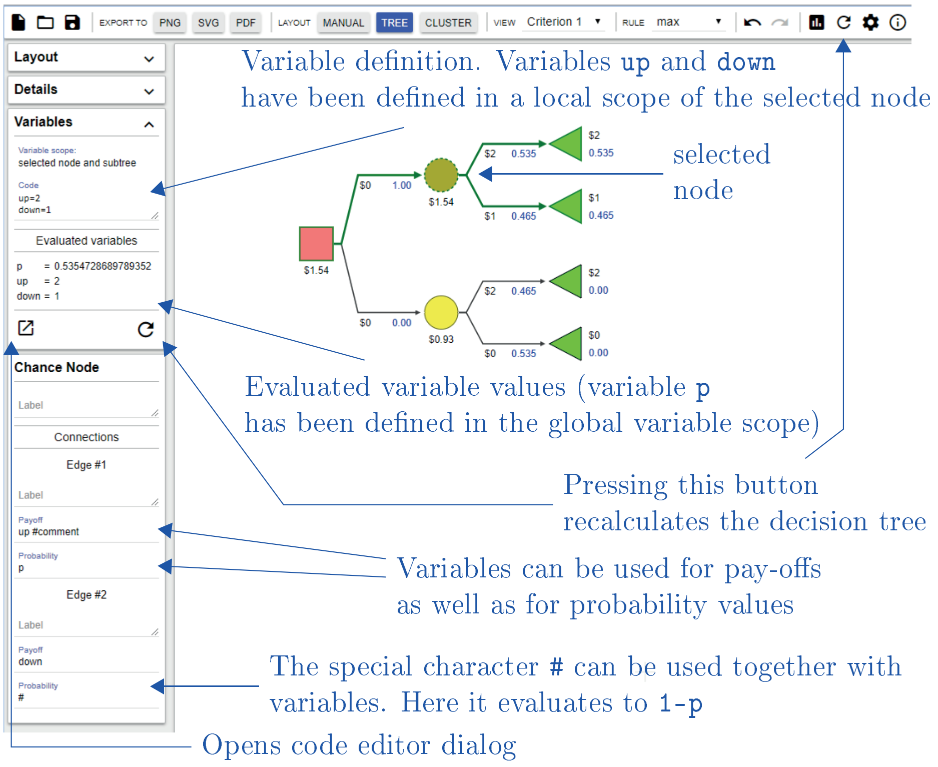
- Code block in left pane loses focus and code was changed;
- Variable definitions dialog is closed;
- When the manual Recalculation '↻ *Recompute'* button is pressed (this button is in the bottom right of the Variables section in the left pane, in the toolbar, and in the bottom left of the Variable definitions dialog box).

Figure 2.24 summarizes variable-related functionality by presenting a screenshot that shows key elements of the interface. Please note that in the section *'Variables selected node and sub-tree'*, the variable `p` was not defined. Actually, it has been defined at the global scope as `p = random()`, and at the scope of the node, it is only available for use. Hence, it is seen in the list of *'Evaluated variables'*.

2.3.2 Name scopes

`SilverDecisions` uses variable scoping, where variable scopes are defined for levels of a decision tree. The variables, defined when no tree node is selected, are defined as *global variables* and hence are visible to all nodes in the tree. At any time, you can deselect nodes (and thus select the global scope) by simply left-clicking the white canvas. Variables can also be defined locally in a particular node, in this case; they are only visible in a node and its subtree.

In order to fully understand the scoping mechanism, have a look at Figure 2.25. In the left sub-figure, two global variables have been defined: `x = 1` and `y = 2`. The right sub-figure shows the scope of the chance node. In the scope of the chance code, the variable `x` is overridden and now `x = 3`. Additionally, a new variable `p = 0.7` has been defined. In the *'Evaluated variables'* section, you can see that `x = 3`, `y = 2` and `p = 0.7`. Those values will be seen at the chance node and its children (a scope gets inherited from the upper level). They will be not seen however



Variable scope:
selected node and subtree

Code
up=2
down=1

Evaluated variables

p = 0.5354728689789352
up = 2
down = 1

Chance Node

Label

Connections

Edge #1

Label

Payoff
up #comment

Probability
p

Edge #2

Label

Payoff
down

Probability
#

Variable definition. Variables up and down have been defined in a local scope of the selected node

selected node

Evaluated variable values (variable p has been defined in the global variable scope)

Pressing this button recalculates the decision tree

Variables can be used for pay-offs as well as for probability values

The special character # can be used together with variables. Here it evaluates to 1-p

Opens code editor dialog

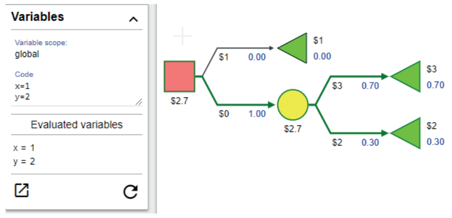
Figure 2.24: Editing variables with SilverDecisions. Remember to always check that the correct variable scope has been selected. Variable scoping is further discussed in Section 2.3.2.

neither by the decision node nor the upper terminal node. Those nodes will see the values defined in the global scope that is `x = 1` and `y = 2` (unless they redefine those values in their own scopes).

The variable scoping mechanism determines the way a tree is calculated. When Recalculate of code is triggered (by pressing the '↻ *Recompute*' button), global variables and the whole tree are recalculated using depth-first search order with global scope as a root. Each code block is recalculated only once.

Expressions in nodes are evaluated in the following order:

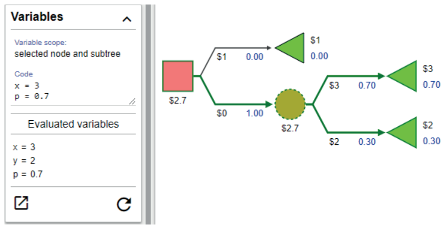
1. The scope is initialized as a deep copy of the parent scope;
2. Code evaluation;
3. Evaluation of "numeric expressions" for each child edge (payoff, probability).


Variables

Variable scope:
global

Code
```
x=1
y=2
```
Evaluated variables
```
x = 1
y = 2
```

(a) `x` and `y` have been defined in the global variable scope and are visible to all nodes throughout the tree.


Variables

Variable scope:
selected node and subtree

Code
```
x = 3
p = 0.7
```
Evaluated variables
```
x = 3
y = 2
p = 0.7
```

(b) `p` has been defined in the selected chance node and hence is visible only to the node and its children. `x` is redefined locally. This new value will be visible to the selected node and its children, other nodes will still see the global value of `x`.

Figure 2.25: Global vs. local variable scopes. If you are not sure which variable scope to use, choose the global scope.

Steps 1 and 2 are only performed when Recalculate of code is triggered; step 3 is recalculated every time a tree is changed (that is why it is not allowed to define variables in payoffs and probabilities).

This process ensures that the value of the variable defined (or redefined) in some node is visible only in this node and its children. Technically: variables are visible in the closures defined by subtrees rooted in a given node. Global scope is the outermost closure.

In particular, consider the following consequence of the redefinition of variables. Assume that in global scope, we have:

```
a = 1
b = a
```

and then in some node we redefine:

```
a = 2
```

then the value of b is still 1 as it has been already defined. If b should dynamically change its value as a change, it should be defined as a function with a parameter.

2.3.3 Variable types

We have signaled at the beginning of Section 2.3 that the SilverDecisions' computational mechanism is based on the math.js computational library. The following data types are supported by *math.js* (more information on *node.js* types is available at mathjs.org/docs/datatypes/, the definitions here are taken from math.js documentation):

- `Boolean` a logical `true`/`false` value. Note that, similarly to other languages, in numeric expressions, `true` is treated as `1`, and `false` is treated as `0`.
- `Number` number for fast floating-point arithmetic,
- `BigNumber` for calculations with arbitrary precision,
- `Complex` support of complex numbers is powered by the library *complex.js*,
- `Fraction` for calculations with fractions,
- `Array` a regular JavaScript array. A multi-dimensional array can be created by nesting arrays,
- `Matrix` a matrix implementation by math.js. A `Matrix` is an object wrapped around a regular JavaScript,
- `Unit` units (kg, cm, m, s, h) can be used to do calculations and perform conversions,
- `String` textual data

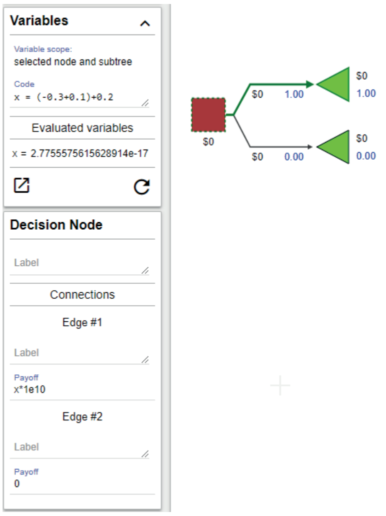
The default type when working with decision trees is `Number`. If payoff or probability evaluates to something else than a number, an error symbolized by a red !! is shown. If a non-number type is found, SilverDecisions tries to cast it to a number (e.g., `"5.5"` is of type `string`, but SilverDecision will cast it to a number, and the tree will be properly evaluated).

The `Number` type offers standard floating-point precision. However, in some cases, the exact computation value could be preferred. For this scenario consider the `Fraction` type.

Figure 2.26 presents a theoretical example where numerical accuracy is an issue in defining an optimal decision in a tree. The expression `(-0.3+0.1)+0.2` evaluates to $2.7755575615628914 \times 10^{17}$ (a number very close to 0) rather than the actual value of 0 (see Figure 2.26a). This is a typical problem in numerical computing – numbers being rational in a 10-base system are not rational numbers in the binary system, so rounding errors occur. This problem can be mitigated by using the `fraction` function to represent full precision in decision trees. In Figure 2.26b, the previous expression is replaced by

```
x = (fraction(-3, 10) + fraction(1, 10)) + fraction(2, 10)
```

and now `x` correctly evaluates to 0. Finally, note that the application uses the `Fraction` type by default when calculating probabilities and payoffs in a decision tree.



Variables

Variable scope:
selected node and subtree

Code
```
x = (-0.3+0.1)+0.2
```

Evaluated variables

x = 2.7755575615628914e-17

Decision Node

Label

Connections

Edge #1

Label

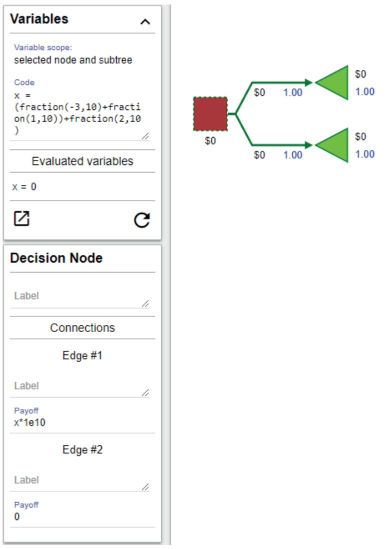
Payoff
x*1e10

Edge #2

Label

Payoff
0

(a) Numerical accuracy leads to a situation where one node is indicated as optimal despite both nodes having the same expected values



Variables

Variable scope:
selected node and subtree

Code
```
x =
(fraction(-3,10)+fracti
on(1,10))+fraction(2,10
)
```

Evaluated variables

x = 0

Decision Node

Label

Connections

Edge #1

Label

Payoff
x*1e10

Edge #2

Label

Payoff
0

(b) The `fraction` function makes it possible to use full precision for rational numbers

Figure 2.26: The `Fraction` data type makes it possible to use full precision for rational numbers in decision trees.

2.3.4 Functions

We can also define our own functions following the *math.js* standard. Please note that functions are evaluated in their lexical scope. Lexical scoping means that if we have the following global definition:

```
a = 1
f() = a
```

and then in some node in the tree we write:

```
a = 10
b = f()
```

then in that node `a=10` but `b=1` because when `f()` is evaluated, it used the value of a from the scope where it is defined.

Note that it is strongly discouraged to define functions that mutate (modify) existing variables like `f() = a[1] = a[1] + 1` when `a = [1,2]`. This way, we could modify variables in payoff, and probability definitions and the SilverDecisions engine assumes that this does not happen – see the discussion in Section 2.3.2 on node evaluation order.

In particular, such actions modify the lexical scope of the f and not in the scope of the node where f is called (which means that formulas in one part of the tree might modify calculations in a completely different part of the tree).

2.3.5 Text interpolation

The text interpolation functionality makes it possible to insert variables and expressions into various text elements of the decision tree. This functionality is enabled by default for any decision tree. However, it can be turned off in the *'Settings'* tab (see Figure 2.10).

This functionality uses a special syntax in the format:

```
${expr}
```

where `expr` can be any valid JavaScript expression.

The string interpolation is supported for the following elements of a decision tree:

- decision tree title
- decision tree description
- node labels
- edge labels
- floating texts

Let us start with a simple example presented in Figure 2.27. We can see that we have a global variable `x = 10` and it has been used in a text label using a special syntax such as:

```
x is equal to ${x} and x+5 is equal to ${x+5}
```

As we can see, the text interpolation mechanism supports simple variable insertion as well as any more complex expression. In fact, any JavaScript code is accepted. The access to variables is governed by the scoping rules as discussed in Section 2.3.2.

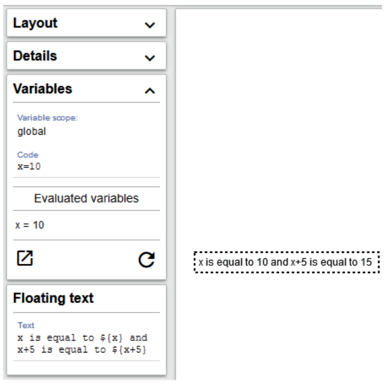
A typical scenario for presenting labels is to present various information regarding the tree states and the current solution. This can be easily achieved by accessing a set of predefined variables. The values of those variables change dynamically whenever the tree is recomputed. The variables are available at nodes and edges of the tree, and their values always depend on the actual scope.

The following variables are available for the presentation of the tree state:

- variables at nodes
  - `childrenPayoff` – pay-off values in decision and chance nodes

  - `aggregatedPayoff` – pay-off in terminal nodes
  - `payoff` – alias for `childrenPayoff` and `aggregatedPayoff`
  - `probabilityToEnter` – probability of entering a particular terminal node
  - `optimal` – `true` if the node is on the optimal path, `false` otherwise
- variables edges
  - `payoff` – pay-off at an edge
  - `optimal` – `true` if the edge is on the optimal path, `false` otherwise

Additionally, there is a special, context-depended variable named `this` which can be used to obtain the state of the sub-tree starting at a particular node or edge. These variables can be used together with JavaScript functions which we discuss later in this section.



Variable scope:
global

Code
x=10

Evaluated variables

x = 10

Text
x is equal to ${x} and x+5 is equal to ${x+5}

x is equal to 10 and x+5 is equal to 15

Figure 2.27: The text interpolation mechanism can be used to show variables in decision tree labels.

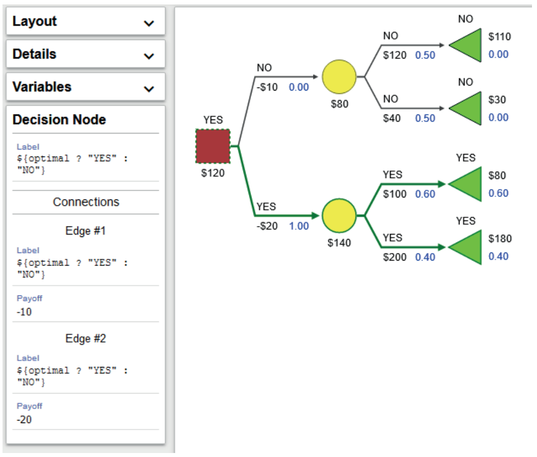
Now, let us see how those variables can be used. Consider the following text interpolation:

```
${optimal ? "YES" : "NO"}
```

This code, depending on the scope-specific value of optimal, will display whether any given node or edge is on the optimal path – for example, see Figure 2.28.

In order to provide convenient a bridge between the decision tree and JavaScript functionality, there is a number of helper functions available:

- `math(expr)` – evaluates a math.js string expression `expr` eg: `${math("random(1, 2)")}`
- `format(value, optionsOrMaxFractionDigits)` format number using JavaScript's Intl.NumberFormat



Decision Node

Label
```
${optimal ? "YES" : "NO"}
```

Connections

Edge #1

Label
```
${optimal ? "YES" : "NO"}
```

Payoff
-10

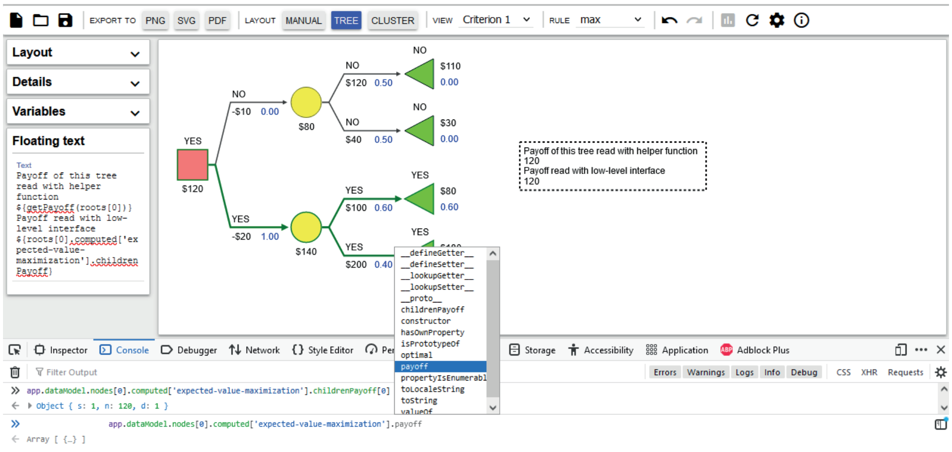
Edge #2

Label
```
${optimal ? "YES" : "NO"}
```

Payoff
-20

Figure 2.28: The inbuilt variables that can be used for presentation. Here we are showing whether any node or edge lies on the optimal path – each label is defined by the following JavaScript expression: `${optimal ? "YES" : "NO"}`.

- `mathFormat(value, options)` – format number using math.js format
- `payoffFormat(value, index)` – format number using payoff format options from app settings
- `probabilityFormat(value)` – format a number using probability format options from app settings
- `getPayoff(nodeOrEdge)` – returns the payoff at a given node, could be used as `getPayoff(this)` at a given node or edge (the same value is also available simply at the `payoff` variable)
- `printPayoff(nodeOrEdge, index)` shows payoff value formatted along with application's settings, can be simply used as `printPayoff(this)` in node or edge scope
- `getProbability(nodeOrEdge)` – returns probability for edges or nodes, can be simply used as `getProbability(this)` in edge/node scope
- `probabilityToEnter` – returns probability for terminal nodes, can be simply used as `probabilityToEnter(this)` in terminal nodes scope
- `printProbability(nodeOrEdge)` – return probability formatted along application's settings
- `optimalEdges(node)` – returns an array of optimal child edges
- `optimalEdge(node, index)` – returns optimal child edge at the given index, or first if the index is not specified



```
Payoff of this tree
read with helper
function
${getPayoff(roots[0])}
Payoff read with low-
level interface
${roots[0].computed['ex
pected-value-
maximization'].children
Payoff}
```

```
Payoff of this tree read with helper function
120
Payoff read with low-level interface
120
```

```
>> app.dataModel.nodes[0].computed['expected-value-maximization'].childrenPayoff[0]
<- ▸ Object { s: 1, n: 120, d: 1 }
>>              app.dataModel.nodes[0].computed['expected-value-maximization'].payoff
<- Array [ {…} ]
```

```
__defineGetter__
__defineSetter__
__lookupGetter__
__lookupSetter__
__proto__
childrenPayoff
constructor
hasOwnProperty
isPrototypeOf
optimal
payoff
propertyIsEnumerabl
toLocaleString
toString
valueOf
```

Figure 2.29: A low-level programmable interface allows the access internal tree representation via the `roots[0]` alias.

The above functions enable number formatting capabilities (which are self-explanatory) and make it convenient to access the JavaScript object representing the state of the decision tree.

To understand the internal representation of the decision tree, have a look at: Figure 2.29.

We can see that in the entire application scope, the state of the decision tree is available as `app.data.nodes[0]`. This state can be explored using the JavaScript console at *'Web Developer Tools'* available in any modern web browser. For example, at the time of writing this book in Mozilla Firefox, this functionality can be found in the main menu by clicking *'More Tools -> Web Developer Tools -> Console'* similarly in Google Chrome, the same tool is found under *'More Tools -> Developer Tools -> Console'*. In the text interpolation mechanism alias for `app.data.nodes[0]` is simply `roots[0]`. However, in some advanced scenarios one might want to use the `app.data.nodes[0]` to navigate around the internal data structure as shown in Figure 2.29.

Together with support functions, we can now read the final payoff outcome of the tree, such as `getPayoff(roots[0])`. It is also possible to access the tree state via navigating through the internal data structure (which is also possible, but we recommend using the support functions). Both approaches are presented in the code below:

```
Payoff of this tree read with helper function
${getPayoff(roots[0])}
Payoff read with low-level interface
${roots[0].computed['expected-value-maximization'].
    childrenPayoff}
```

In this section, we have shown the `${ }` interpolation mechanism along with some simple examples. This interface allows advanced users to increase the reporting capability of `SilverDecisions`. However, note that the interpolation functionality is for data presentation only. The user should not interpolate code that might have side effects on other variables, such as `${z = math("random(1, 2)")}`. Code of this kind is not guaranteed to work, and there is no mechanism for the control of the execution order (the evaluation order has been discussed in Section 2.3.2).

Chapter 3

Mathematical foundations of decision trees

Bogumił Kamiński
Paweł Prałat

In this chapter, we discuss the mathematical foundations of decision tree analysis. This treatment serves as a means of formalizing intuitive definitions given in Chapter 2 and is covered in Section 3.1.

In the following sections, we extend the analysis to situations where the parameters of the tree (probabilities or payoffs) are uncertain. This approach follows the observation that the decision-makers are often unsure about the exact parameters of such trees. This uncertainty can have probabilistic nature (a situation called risk) or non-probabilistic.

Risk-related uncertainty about the parameters of the model is typically a result of the imprecise measurement of model parameters. For example, consider a marketing campaign planner who wants to sell a new product to all current customers of her company. Assume that a pilot campaign was targeted at 1,000 randomly selected customers, and 5% of them have bought the product. This value is only an estimate of the probability of buying a product in a whole population, and this uncertainty can be taken into account when analyzing decision trees (if you do not have much experience with performing statistical analysis, please refer to a textbook on this topic, e.g., [16]).

On the other hand, non-probabilistic uncertainty has two sources. One of them is *true* uncertainty; for example, a manager wants to assess the value for which she can sell a unique ancient vase at an auction. She might be able to define the range of possible selling prices, but as this is a one-time event, it would be difficult for her to assign a probability distribution to this value. The second source is that some parameters can be a decision that does not influence the structure of the decision tree but nevertheless influences the payoffs. For instance, it could be a selling price of a good.

Collectively, the analysis of the influence of the change of parameters in a decision tree is called sensitivity analysis. In Section 3.2 we provide the mathematical underpinning of probabilistic sensitivity analysis. In Section 3.3, we discuss examples of probabilistic and non-probabilistic sensitivity analysis in SilverDecisions.

3.1 Formal model of a decision tree and associated payoffs

This section aims to provide precise and rigorous mathematical foundations of various concepts and definitions used in `SilverDecisions`. We start with a definition of a decision tree before discussing the associated concept of payoffs.

3.1.1 Decision trees

Let us start with an informal definition of a decision tree that a mathematically rigorous exposition will followed. A decision tree consists of three types of nodes: decision, chance, and terminal. Nodes are connected by arrows that will be called edges; we will say that the arrow points from source to destination. In `SilverDecisions`, by default, decision nodes are presented as red squares, chance nodes are yellow circles, and terminal nodes are green triangles. In Figure 3.1, we present an example of a decision tree in which we have one decision node called A, one chance node called B, and three-terminal nodes C, D, and E. There are the following rules for how arrows are drawn:

1. exactly one node is not a destination of any arrow; this node is called the root node of a tree;
2. apart from the root node of a tree, all other nodes are destinations of exactly one arrow;
3. terminal nodes cannot be sources of any arrow; decision and chance nodes have at least one arrow of which they are a source;
4. there is exactly one path from the root node to any other node in the decision tree.

Note that, under these conditions, there are no loops in the defined diagram. Typically, we draw the root node on the leftmost part of the plot, and then we draw arrows so that they point from left to right.

In Figure 3.1, one can additionally observe blue numbers that are written below the head of each arrow. These numbers have the following meaning:

1. for arrows having a chance node as their source, the number represents the probability of choosing that path when leaving a chance node; the probabilities going out from one chance node must add up to one;
2. for arrows having a decision node as their source, there are two values allowed: 1 and 0. The value of 1 means that following this arrow is an optimal decision, and the value of 0 means that following this arrow is not an optimal decision. By default, in `SilverDecisions`, optimal paths are additionally highlighted with bold green.

Now, let us formalize these concepts. (To learn more about graph theory, we direct the reader to e.g., [37] or [20].) A decision tree $\mathcal{T}$ consists of nodes forming set V that are connected by edges forming a set

$$E \subseteq \binom{V}{2} = \Big\{\{u, v\} : u, v \in V \text{ and } u \neq v\Big\}.$$

Each node $v \in V$ in a decision tree is associated with the following functions:

1. $T(v)$: type of a node, which can be *decision*, *chance*, or *terminal*;
2. $N(v)$: name of a node.

Since each node $v \in V$ has exactly one of the three values assigned to $T(v)$, the set of nodes V is partitioned into three sets: $V_d = \{v \in V : T(v) = \textit{decision}\}$, $V_c = \{v \in V : T(v) = \textit{chance}\}$, and $V_t = \{v \in V : T(v) = \textit{terminal}\}$; in particular, $V = V_d \cup V_c \cup V_t$.

On the other hand, each edge $e \in E$ in a decision tree is associated with the following functions:

1. $Y(e) \in \mathbf{R}^n$ is a vector of n payoffs associated with this edge;
2. $v_1(e)$ is a start node of this edge;
3. $v_2(e)$ is an end node of this edge;
4. if $v_1(e) \in V_c$, then $p(e) \in [0, 1]$ is the probability of selecting this edge;
5. if $v_1(e) \in V_d$, then $d(e) \in \{0, 1\}$ is the decision about selecting this edge;
6. $N(e)$: name of an edge.

Since each edge, $e \in E$ is a pair of nodes, one of which is a start node and the other one is an end node, a decision tree is, in fact, a directed tree. That is, each edge e is oriented from $v_1(e)$ to $v_2(e)$. (Such orientations are visualized on our figures as arrows). As a result, each node $v \in V$ has associated in- and out-degree ($\deg^{in}(v)$ and, respectively, $\deg^{out}(v)$), which

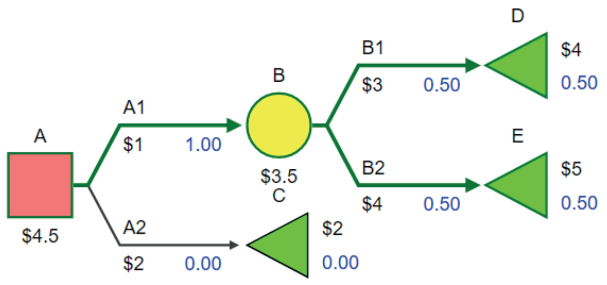


Figure 3.1: An example of a decision tree $\mathcal{T}_1$.

count the number of edges with v as their end node and, respectively, the number of edges with v as their start node. Formally, for each node $v \in V$,

$$\begin{aligned} \deg^{in}(v) &= \Big|\{e \in E : v_2(e) = v\}\Big| \\ \deg^{out}(v) &= \Big|\{e \in E : v_1(e) = v\}\Big|. \end{aligned}$$

In Figure 3.1, we present a simple decision tree $\mathcal{T}_1$. It has five nodes forming set $V = \{A, B, C, D, E\}$. Node A is a *decision* node (represented as a red square), node B is a *chance* node (represented as a yellow circle), and the remaining nodes are *terminal* nodes (represented as green triangles); this association defines function T. In our example, function N for nodes is simply the identity function, that is, $N(v) = v$. The decision tree $\mathcal{T}_1$ has 4 edges forming set $E = \{\{A, B\}, \{A, C\}, \{B, D\}, \{B, E\}\}$. The table below presents the values of all the functions associated with edges:

function	$\{A, B\}$	$\{A, C\}$	$\{B, D\}$	$\{B, E\}$
Y	\$1	\$2	\$3	\$4
v_1	A	A	B	B
v_2	B	C	D	E
p	–	–	0.5	0.5
d	1	0	–	–
N	$A1$	$A2$	$B1$	$B2$

A well-defined decision tree $\mathcal{T} = (V, E)$ is a pair consisting of the set of nodes V and the set of edges E, together with the associated functions defined above, that additionally satisfy the following seven properties:

1. there exists a unique node with in-degree equal to 0, that is,

$$\Big|\{v \in V : \deg^{in}(v) = 0\}\Big| = 1;$$

such a unique node will be called the root of $\mathcal{T}$ and will be denoted by $r(\mathcal{T})$;

2. all nodes but the root have in-degree equal to 1, that is,
$$\forall v \in V \setminus \{r(\mathcal{T})\} \ \deg^{in}(v) = 1;$$
3. terminal nodes have out-degrees equal to 0, that is,
$$\forall v \in V_t \ \deg^{out}(v) = 0;$$
4. there exists a unique path from the root to any of the other nodes, that is,
$$\forall v \in V \setminus \{r(\mathcal{T})\} \ \exists (v_0, v_1, \ldots, v_k) \quad : \quad \begin{array}{l} v_0 = r(\mathcal{T}), v_k = v, \text{ and} \\ \forall i \in \{1, \ldots, k\} \ \{v_{i-1}, v_i\} \in E; \end{array}$$
5. decision and chance nodes have non-zero out-degrees, that is,
$$\forall v \in V_d \cup V_c \ \deg^{out}(v) \geq 1;$$
6. probabilities of selecting edges going out from any *chance* node add up to one, that is,
$$\forall v \in V_c \sum_{e \in E : v_1(e) = v} p(e) = 1;$$
7. decisions of selecting edges going out from any *decision* node add up to one, that is,
$$\forall v \in V_d \sum_{e \in E : v_1(e) = v} d(e) = 1;$$
in other words, precisely one of the edges going out of any *decision* node is selected.

It is easy to verify that the tree $\mathcal{T}_1$, presented in Figure 3.1, satisfies all the required properties. In particular, its root is $r(\mathcal{T}_1) = A$. For the only *decision* node, node A, precisely one value of function d is equal to 1. For the only *chance* node, node B, the probabilities sum up to $1 = 0.5 + 0.5$.

As was already mentioned in Chapter 2, `SilverDecisions` simplifies the process of assigning probabilities to chance nodes so that they add up to one. This is achieved by using the `#` sign. Suppose that $v \in V$ is a chance node of out-degree k and edges E_1, E_2, ..., E_k are going out of v. For any index $i \in \{1, \ldots, k\}$, the probability $p(E_i)$ assigned to edge E_i defined in `SilverDecisions` is either a numerical value from the interval $[0, 1]$, or it remains undefined, that is, the `#` sign is put in `SilverDecisions` interface. Let us denote by $I_v^{\#}$ the set of indices corresponding to probabilities with the `#` value and the remaining indices from set $I_v = \{1, \ldots, k\} \setminus I_v^{\#}$. First, note that for the decision tree to be well defined, it must be the case that $S_v = \sum_{i \in I_v} p(E_i) \leq 1$, that is, the already assigned probabilities cannot sum to the value greater than one. Then, for all $i \in I_v^{\#}$ we define $p(E_i) = (1 - S_v)/|I_v^{\#}|$. By doing this, we ensure that all probabilities add up to 1 and are non-negative. Moreover, all edges with a `#` sign assigned to them have the same probability.

In this formal description, we assumed that payoffs (Y) and probabilities (p) take arbitrary but concrete values. However, in `SilverDecisions` they may depend on values of some other defined variables, as was explained in Chapter 2. This feature is especially useful for performing sensitivity analysis as it allows decision-makers to quantify the uncertainty about payoffs and probabilities using variables. This topic will be discussed in detail in Section 3.3.

3.1.2 Payoffs

So far, we have defined what a proper decision tree is. However, a crucial element of modeling using decision trees is the calculation of payoffs for a given decision tree and the process of selecting optimal decisions. We will now explain how those two features are implemented in `SilverDecisions`.

Let $\mathcal{D}$ be the family of all functions d associated with edges going out of *decision* nodes. Since there are $\deg^{out}(v)$ options to choose from in each *decision* node $v \in V_d$, the number of possible decisions that can be made is equal to

$$|\mathcal{D}| = \prod_{v \in V_d} \deg^{out}(v).$$

Having said that, note that by making a decision for a given *decision* node, we select exactly one branch, and the remaining branches are not going to be reachable. As a result, values of function d on nodes present in those unreachable branches will be negligible. Formally, one may define an equivalence relation between functions from $\mathcal{D}$ that differ only on nodes present in unreachable branches and consider one member from each equivalence class. For simplicity, we will use the whole family $\mathcal{D}$ in further discussion.

Our ultimate goal will be to find a function $d \in \mathcal{D}$ that is optimal from some point of view. But for now, we will concentrate on computing various evaluation criteria for a given function $d \in \mathcal{D}$.

We assume that the manager-making decisions may select any function $d \in \mathcal{D}$ that reflects her strategy for handling the *decision* nodes. However, she cannot affect random events modeled by the *chance* nodes. We assume that each *chance* node $v \in V_c$, with edges $E_1, E_2, \ldots, E_{\deg^{out}(v)}$ going out of v, selects one of these edges randomly; edge E_i is selected with probability $p(E_i)$. The events associated with different *chance* nodes are independent of each other. (If, in the original decision problem, the evens associated with different *chance* nodes are *not* independent, then one must restructure the tree to ensure that this property holds. Note that this is always possible and easy to do, but this operation increases the size of a decision tree.) The manager does not know the outcomes of these random experiments, and she needs to make a decision based only on the decision tree $\mathcal{T}$ before these experiments are performed. (In fact, some random experiments might never get executed as they might fall in the decision tree branch that the manager does not select.)

For a given function $d \in \mathcal{D}$, after the outcome of random experiments explained above is made available, there is a unique path $P = (v_0, v_1, \ldots, v_k)$ from the root $r(\mathcal{T})$ to one of the *terminal* nodes. Path P follows the decisions made for *decision* nodes and the outcome of random experiments for *chance* nodes. Let X_d be the random variable equal to the payoff incurred by the random path P, that is,

$$X_d(\mathcal{T}) = \sum_{i=1}^{k} Y(\{v_{i-1}, v_i\}),$$

where k also is a random variable that depends on the realization of P. (Note that decision trees are typically not symmetric; in particular, *terminal* nodes might have different distances from the root.)

Note that the assumption made in `SilverDecisions` is that the payoff assigned to a given path P is a sum of the payoffs along with it. However, in practice, one can encounter other aggregation schemes such as discounting of payoffs. In such situations, the user of `SilverDecisions` is responsible for designing a tree in such a way so that the payoffs along the path can be aggregated with addition when evaluating it.

`SilverDecisions` supports three modes for calculation of the payoff for a given decision tree $\mathcal{T}$, assuming that specific decisions in the tree are already made (again, such decisions are captured by a given function d), but the uncertainty in chance nodes is not realized yet. For simplicity, in the definitions below, we assume that Y has a single dimension; that is, only one criterion for evaluation is used. We will discuss the multiple-criteria problem later. The three modes are:

1. expected payoff $P_E(\mathcal{T}, d)$, that is, $P_E(\mathcal{T}, d) := \mathbb{E}[X_d]$;
2. chance-max payoff $P_{\max}(\mathcal{T}, d)$, that is, $P_{\max}(\mathcal{T}, d)$ is the smallest $M_d \in \mathbf{R}$ such that deterministically $X_d \leq M_d$;

3. chance-min payoff $P_{\min}(\mathcal{T}, d)$, that is, $P_{\min}(\mathcal{T}, d)$ is the largest $m_d \in \mathbf{R}$ such that deterministically $X_d \geq m_d$.

The expected payoff $P_E(\mathcal{T}, d)$ is the most natural approach as it gives us a piece of information about what is a typical outcome of the decision associated with the function d. The chance-max payoff $P_{\max}(\mathcal{T}, d)$ and the chance-min payoff $P_{\min}(\mathcal{T}, d)$ tell us the best and, respectively, the worst-case scenarios that are typically substantially far away from each other.

Finally, depending on the application at hand, we might want to select a specific decision function d to maximize or minimize one of the three measures of the quality of decisions. In total, we have six options for an optimization approach, and all of them are available in `SilverDecisions` software:

1. expectation maximization; with optimal payoff $P_E^{\max}(\mathcal{T}) := \max_{d \in \mathcal{D}} P_E(\mathcal{T}, d)$;
2. expectation minimization; with optimal payoff $P_E^{\min}(\mathcal{T}) := \min_{d \in \mathcal{D}} P_E(\mathcal{T}, d)$;
3. maxi-min; with optimal payoff $P_{\min}^{\max}(\mathcal{T}) := \max_{d \in \mathcal{D}} P_{\min}(\mathcal{T}, d)$;
4. mini-min; with optimal payoff $P_{\min}^{\min}(\mathcal{T}) := \min_{d \in \mathcal{D}} P_{\min}(\mathcal{T}, d)$;
5. maxi-max; with optimal payoff $P_{\max}^{\max}(\mathcal{T}) := \max_{d \in \mathcal{D}} P_{\max}(\mathcal{T}, d)$;
6. mini-max; with optimal payoff $P_{\max}^{\min}(\mathcal{T}) := \min_{d \in \mathcal{D}} P_{\max}(\mathcal{T}, d)$.

Since $|\mathcal{D}|$ can be large and for a given $d \in \mathcal{D}$ the number of associated random events that affect payoff X_d can be significant, let us now discuss how the above-defined payoffs are efficiently computed in `SilverDecisions` software.

For a given decision tree $\mathcal{T}$ and a given decision function $d \in \mathcal{D}$, one may simply consider all random paths P that can be created during the random experiments associated with the *chance* nodes, and use them to compute the desired payoff. Note that every path P terminates at some *terminal* node. Moreover, for any *terminal* node $v \in V_t$, there is a unique path $P_v = (v_0, v_1, \ldots, v_k)$ from the root $v_0 = r(\mathcal{T})$ to $v_k = v$. Let $q(v)$ be the probability of obtaining path P_v during the experiment, and let $Y(v)$ be the payoff associated with path P_v, that is, $Y(v) = \sum_{i=1}^{k} Y(\{v_{i-1}, v_i\})$ sum.

Of course, a given function d might prevent P_v from being generated during the random process. If this is the case, then we fix $q(v) = 0$. The remaining *terminal* nodes are put into $V_t(d) \subseteq V_t$, indicating that these nodes are reachable for this particular function d. We may easily compute $q(v)$ for such nodes by multiplying the probabilities of selecting edges from *chance* nodes on the path P_v (since the experiments associated with *chance* nodes are independent). Note that it might be the case that $q(v) = 0$ for some reachable nodes in $V_t(d)$ but, typically, this is not the case. On the other hand, by definition, $q(v) = 0$ for all $v \notin V_t(d)$. It follows that

$$\begin{aligned}
P_E(\mathcal{T}, d) &= \mathbb{E}[X_d] = \sum_{v \in V_t} q(v)\ Y(v) = \sum_{v \in V_t(d)} q(v)\ Y(v) \\
P_{\max}(\mathcal{T}, d) &= M_d = \max_{v \in V_t(d)} Y(v) \\
P_{\min}(\mathcal{T}, d) &= m_d = \min_{v \in V_t(d)} Y(v).
\end{aligned}$$

Fortunately, there is no need to investigate all possible paths to compute these values. One can do it much more efficiently by applying some simple recursive formulas. Before we do that, we need one additional definition. For an appropriately given defined decision tree $\mathcal{T}$ and any node $v \in V$, we define a sub-tree selection operator $\mathcal{T}(v)$, which selects a sub-tree of the original tree $\mathcal{T}$ that is rooted in v. Obviously, for any $v \in V$ we have that $\mathcal{T}(v)$ is a properly defined decision tree and, in particular, $\mathcal{T} = \mathcal{T}(r(\mathcal{T}))$.

With this definition at hand, we can recursively compute the payoffs of an adequately defined decision sub-trees, finishing with the original tree $\mathcal{T}$. The three options differ in how one aggregate in chance nodes the payoffs corresponding to various sub-trees. In the expected value case, one needs to calculate the weighted mean of payoffs of sub-trees attached to every edge. The chance-max payoff aggregation chooses the maximum payoff from payoffs in the sub-trees. Similarly, the chance-min aggregation chooses the minimum payoff.

The expected payoff $P_E(\mathcal{T})$ can be calculated the following way:

$$P_E(\mathcal{T}, d) = \begin{cases} 0 & \text{if } r(\mathcal{T}) \in V_t \\ \displaystyle\sum_{e \in E: v_1(e) = r(\mathcal{T})} d(e)\,(y(e) + P_E(\mathcal{T}(v_2(e)), d)) & \text{if } r(\mathcal{T}) \in V_d \\ \displaystyle\sum_{e \in E: v_1(e) = r(\mathcal{T})} p(e)\,(y(e) + P_E(\mathcal{T}(v_2(e)), d)) & \text{if } r(\mathcal{T}) \in V_c. \end{cases}$$

Similarly,

$$P_{\max}(\mathcal{T}, d) = \begin{cases} 0 & \text{if } r(\mathcal{T}) \in V_t \\ \displaystyle\sum_{e \in E: v_1(e) = r(\mathcal{T})} d(e)\,(y(e) + P_{\max}(\mathcal{T}(v_2(e)), d)) & \text{if } r(\mathcal{T}) \in V_d \\ \displaystyle\max_{e \in E: v_1(e) = r(\mathcal{T})} (y(e) + P_{\max}(\mathcal{T}(v_2(e)), d)) & \text{if } r(\mathcal{T}) \in V_c, \end{cases}$$

and

$$P_{\min}(\mathcal{T},d) = \begin{cases} 0 & \text{if } r(\mathcal{T}) \in V_t \\ \displaystyle\sum_{e\in E:v_1(e)=r(\mathcal{T})} d(e)\left(y(e) + P_{\min}(\mathcal{T}(v_2(e)),d)\right) & \text{if } r(\mathcal{T}) \in V_d \\ \displaystyle\min_{e\in E:v_1(e)=r(\mathcal{T})} \left(y(e) + P_{\min}(\mathcal{T}(v_2(e)),d)\right) & \text{if } r(\mathcal{T}) \in V_c. \end{cases}$$

The above formulas assume that decisions captured by function d have already been made in the decision tree. Below, we describe how optimal function d is found by `SilverDecisions` in any of the six optimization criteria explained above. For example, to compute the optimal payoffs $P_E^{\max}(\mathcal{T})$ and $P_E^{\max}(\mathcal{T})$, we apply the following recursive formulas:

$$P_E^{\max}(\mathcal{T}) = \begin{cases} 0 & \text{if } r(\mathcal{T}) \in V_t \\ \displaystyle\max_{e\in E:v_1(e)=r(\mathcal{T})} \left(y(e) + P_E^{\max}(\mathcal{T}(v_2(e)))\right) & \text{if } r(\mathcal{T}) \in V_d \\ \displaystyle\sum_{e\in E:v_1(e)=r(\mathcal{T})} p(e)\left(y(e) + P_E^{\max}(\mathcal{T}(v_2(e)))\right) & \text{if } r(\mathcal{T}) \in V_c, \end{cases}$$

and

$$P_E^{\min}(\mathcal{T}) = \begin{cases} 0 & \text{if } r(\mathcal{T}) \in V_t \\ \displaystyle\max_{e\in E:v_1(e)=r(\mathcal{T})} \big(y(e) + P_E^{\min}(\mathcal{T}(v_2(e)))\big) & \text{if } r(\mathcal{T}) \in V_d \\ \displaystyle\sum_{e\in E:v_1(e)=r(\mathcal{T})} p(e)\big(y(e) + P_E^{\min}(\mathcal{T}(v_2(e)))\big) & \text{if } r(\mathcal{T}) \in V_c. \end{cases}$$

These formulas yield the optimum payoffs, but, at the same time, the minimum or the maximum operator applied to *decision* nodes provides us with an optimal decision, namely, function d. The remaining four optimum payoffs and their corresponding optimal decisions may be computed analogously.

Let us briefly comment on the time complexity of the proposed solution. Recall that a naive way to find the optimal decision would require investigating $|\mathcal{D}| = \prod_{v\in V_d} \deg^{out}(v)$ options. On the other hand, using the proposed recursive formulas, the computational complexity of finding an optimal solution is much smaller, namely, in the order of $\sum_{v\in V_d} \deg^{out}(v)$.

It is easy to see that several different functions d from $\mathcal{D}$ may lead to the same optimum payoff; see, for example, the decision tree presented in Figure 3.2. `SilverDecisions` considers such a possibility, and if this is the case, then all optimal paths are highlighted; see, again, Figure 3.2. We can observe that decisions $A1$ and $A2$ give the same expected payoff.

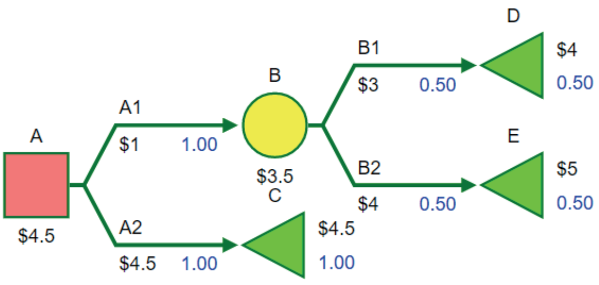
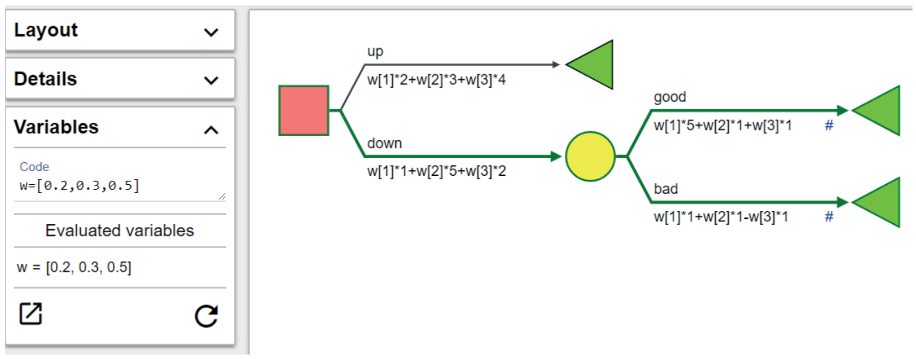


Figure 3.2: A decision tree in which both branches give the same payoff.



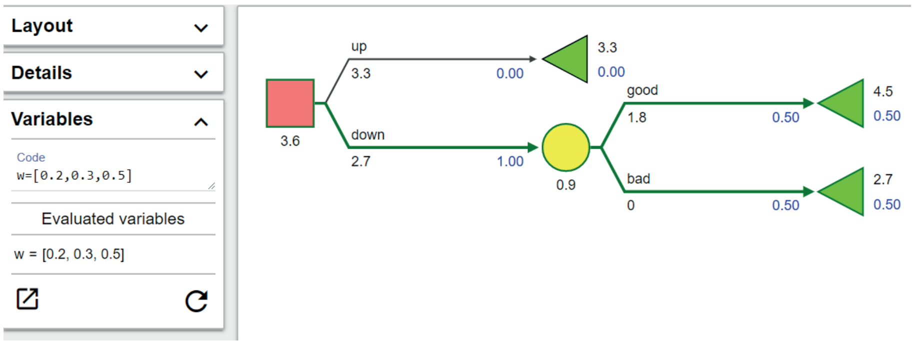
```
w=[0.2,0.3,0.5]
```

w = [0.2, 0.3, 0.5]

Figure 3.3: A view of a tree with three criteria in raw mode.

Therefore, they are both highlighted in green. On the other hand, for the decision tree presented in Figure 3.1, only decision *A*1 was optimal under the maximization criterion. It should, however, be stressed that if such a situation is encountered, then it *does not* means that both decisions are chosen. This would violate rule 7 of proper decision trees described above that in *decision* nodes, only one decision can be chosen. If two (or more) paths are highlighted in `SilverDecisions`, then this implies that the payoff of every highlighted path is equivalent and yields the optimal payoff. In practice, the decision-maker will have t choose one of them eventually.

If a user needs to deal with a multiple-criteria problem, then it can be easily handled in `SilverDecisions` by using criteria weightings. Indeed, suppose that the problem involves n criteria. Then, one can define a vector variable `w=[w1,w2,...,wn]` in the global scope in `SilverDecisions` (as explained in Chapter 2); values `w1`, `w2`, . . ., `wn` should be concrete numerical values but here we use symbols for the generality of the notation. Next, in every edge where the payoff is defined, one can calculate payoff using the formula `w[1]*c1+w[2]*c2+...+w[n]*cn`, where `c1`, `c2`, . . ., `cn` are again some concrete numerical values.



```
w=[0.2,0.3,0.5]
```

w = [0.2, 0.3, 0.5]

Figure 3.4: A view of a tree with three criteria in normal mode.

We show how this can be done in `SilverDecisions` in Figures 3.3 and 3.4. The first plot is presented in the *raw* mode (described in Chapter 2) and shows what exactly formulas should look like. The second plot presents the result of the computation.

The benefit of this approach is that one can easily change the weights of the criteria in one place (global variable scope) and immediately see the impact of the new weights on the payoffs and optimality of decisions. The above process is still somewhat cumbersome, especially for large trees and multi-criteria problems. In practice, however, two-criteria problems are encountered very often, and there are some standard methods to analyze them. In Section 4.2 we will get back to this topic and describe specialized functionalities of `SilverDecisions` for handling such two-criteria decision problems.

3.2 Introduction to Monte Carlo simulation

Before we move to sensitivity analysis and describe various features provided by `SilverDecisions`, we summarize some standard results from the theory of Monte Carlo simulation. (A reader interested in a more detailed exposition of Monte Carlo simulation tools techniques can find their exposition in, e.g., [22] or [33].) The results presented in this section provide us with theoretical foundations of probabilistic sensitivity analysis. In particular, the bounds introduced can be used to assess the precision of the results of the simulation performed and reported by `SilverDecisions`.

Monte Carlo experiments belong to a broad family of computational algorithms and techniques that rely on repeated random sampling to obtain numerical results. Monte Carlo methods, in principle, may be used to solve any problem having a probabilistic interpretation. In our application, we use them to model various phenomena with significant uncertainty in inputs, such as the calculation of risk assessment associated with the process of making decisions.

3.2.1 Example decision tree

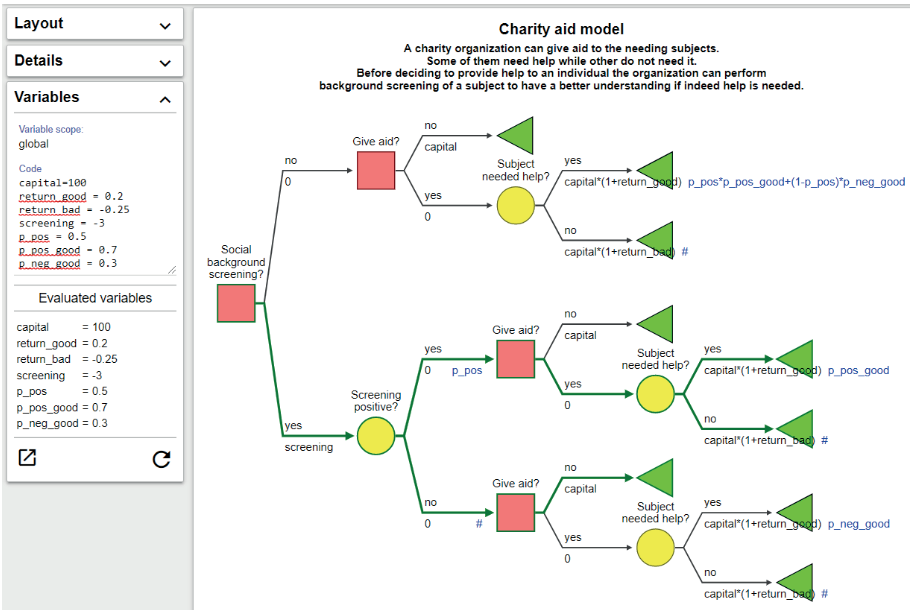
In this section, and the next one, we use the following example of a decision tree. On the one hand, it is easy to describe and interpret, but, on the other hand, it is complex enough to create a few issues that require simulation.

The decision problem we want to analyze is aimed at defining an optimal policy for charity aid organizations (see Figure 3.7). Assume the organization has a fixed `capital` amount of money to spend per one subject. The organization can either decide not to give help, in which case it keeps the money and may use it for other purposes. Alternatively, it can give the money to the subject. The subject might need help, in which case the organization gains `return_good` return (which is positive), or in other words, it assesses its benefit as `capital * (1 + return_good)`. It is also possible that the subject does not need help. In this case, the organization has a negative `return_bad` return and gets a `capital * (1 + return_bad)` benefit. (In the example, we assume that all outcomes are measured in monetary terms; of course, in practice, these returns should be considered to be *monetary equivalents* of the utility of the charity aid organization gains associated with various actions.)

As an optional action, the organization can spend a `screening` amount to do an additional investigation if the subject needs help or not. The screening result can be positive or negative, affecting the probability that the subject requires support. Assume that `p_pos` is the probability of a positive screening result. If the screening is positive, then the probability that the subject indeed requires help is `p_pos_good`. If it is negative, then this probability is `p_neg_good` instead. Note that these probabilities imply that, without screening, the probability that the subject requires help is equal to:

```
p_pos * p_pos_good + (1 - p_pos) * p_neg_good.
```

Additionally, we assume that after making a decision on the policy on how the subjects should be screened, the organization will apply this policy to many subjects it considers helping. This specification allows us to describe the decision problem using the tree presented in Figure 3.5 (in raw mode). If you do not remember how to switch raw mode on and off, you can see it in Figure 3.6.



Layout

Details

Variables

Variable scope:
global

Code
```
capital=100
return_good = 0.2
return_bad = -0.25
screening = -3
p_pos = 0.5
p_pos_good = 0.7
p_neg_good = 0.3
```

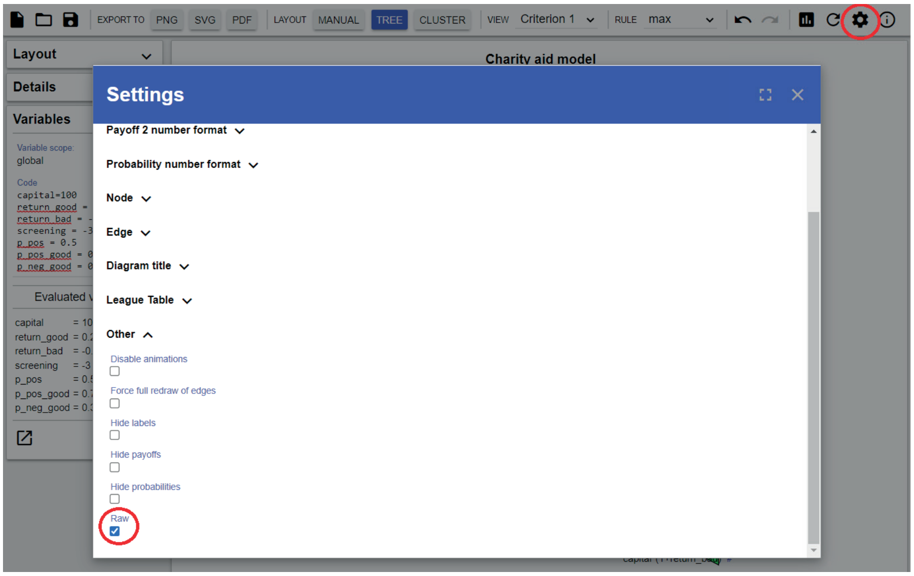
Evaluated variables

```
capital     = 100
return_good = 0.2
return_bad  = -0.25
screening   = -3
p_pos       = 0.5
p_pos_good  = 0.7
p_neg_good  = 0.3
```

Charity aid model

A charity organization can give aid to the needing subjects.
Some of them need help while other do not need it.
Before deciding to provide help to an individual the organization can perform
background screening of a subject to have a better understanding if indeed help is needed.

Figure 3.5: Charity model (in SilverDecisions raw mode).



Settings

Payoff 2 number format

Probability number format

Node

Edge

Diagram title

League Table

Other

Disable animations ☐

Force full redraw of edges ☐

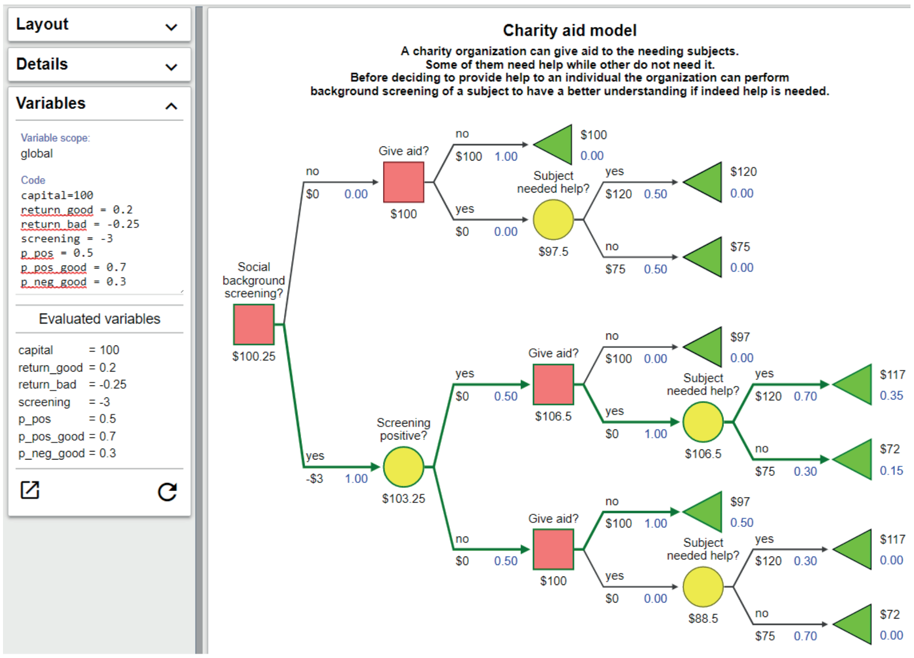
Hide labels ☐

Hide payoffs ☐

Hide probabilities ☐

Raw ☑

Figure 3.6: Turning on/off the raw mode.



```
capital=100
return_good = 0.2
return_bad = -0.25
screening = -3
p_pos = 0.5
p_pos_good = 0.7
p_neg_good = 0.3
```

Evaluated variables

```
capital     = 100
return_good = 0.2
return_bad  = -0.25
screening   = -3
p_pos       = 0.5
p_pos_good  = 0.7
p_neg_good  = 0.3
```

Charity aid model

A charity organization can give aid to the needing subjects.
Some of them need help while other do not need it.
Before deciding to provide help to an individual the organization can perform
background screening of a subject to have a better understanding if indeed help is needed.

Figure 3.7: Charity model: optimal path for reference parameterization.

In Figure 3.7, we present the initial parameterization of our model (raw mode is disabled). We keep all the parameters of the model as variables, and you can see their values in the left pane in the *Variables* section. It can be observed that for the reference parameterization, the optimal policy is to perform screening. In this case, if screening is positive, we give aid, while if it is negative, we do not give it and instead spend money on other activities.

In the sensitivity analysis, we aim is to analyze how sensitive the optimal policy we have found is to changes in the parameters of our model. Before we move to theoretical analyses, let us comment on the general problem setting we consider here. We assume that the decision-maker is uncertain about the parameters of the model. However, the policy, once chosen, is applied many times. In all the examples considered, we assume that the decision-maker is interested in the maximization of the expected value of the benefit. The problem the decision-maker typically faces is that the parameterization of the associated decision tree is uncertain. For instance, in our example scenario the probabilities `p_pos`, `p_pos_good`, and `p_neg_good` can most likely only be assessed with a limited degree of precision. Therefore, it is natural to ask how sensitive the optimal decision is to the changes in the values of uncertain parameters. Clearly, the most desired scenario is when the decision-maker would be able to conclude that the optimal decision remains the same even if the parameters of the decision tree were slightly changed. This is exactly the objective of sensitivity analysis in `SilverDecisions`.

This framework and the problem structure are suitable and natural for our example problem: the charity organization provides help to many potential subjects. However, this flexible and general framework is applicable to many other situations. For example, assume that we developed some drug, and we obtained an initial assessment of its efficiency based on the results of some clinical trials. Using this information, our goal is to devise optimal treatment plans for patients, potentially conditional on the results of diagnostic tests. However, while making this decision, we know that our assessment of the effect of the drug is uncertain, so the decision-maker would like to know if the optimal decision might be affected if the actual efficiency of the drug is not exactly the one as observed in the clinical trial but probably not too far from it.

3.2.2 Theoretical foundations of probabilistic sensitivity analysis

Let X be any random variable, and assume that we can sample from it; that is, the computer can generate a value x from the domain of X in such a way that the probability of generating x is in accordance with $p(x)$ the probability distribution function associated with X. (Readers that do not have much experience with probability theory can find its introductory exposition in, e.g., [8] and comprehensive treatment in [32].) In our analysis, X is typically a parameter of a decision tree. We also assume that P is some transformation of this parameter. This is usually the payoff associated with a tree under a given decision in our decision trees. We assume that P is a deterministic function, but $P(X)$ is a random variable. Technically, in `SilverDecisions`, parameters that can serve the purpose of X discussed here are variables defined in global scope; see Chapter 2 for a discussion of various ways to define variables in `SilverDecisions`.

In Monte Carlo simulation, we draw n independent samples from X, where n is a sufficiently large natural number. Let us denote the random variables representing the results of transformations of the independent draws by $P_1, P_2, \ldots, P_n$. Furthermore, we assume that the expected value and variance of $P(X)$ exist and are finite. To simplify the notation, let us denote $E(P(X)) = \mu$ and $D^2(P(X)) = \sigma^2$. Note that the assumption of the existence of these two values holds in most typical situations when decision trees are used to model decision problems. In particular, if one knows that the payoff from a decision tree is bounded from above and from below by some constants, then both the expected value and the variance of $P(X)$ are guaranteed to exist. (We do not consider non-standard random variables without this desired property as they are not supported in `SilverDecisions`.)

Let us define $\bar{P} = \frac{1}{n}\sum_{i=1}^{n} P_i$. This random variable represents the average value of n independent random variables P_i. It is easy to see that

$$E(\bar{P}) = \mu,$$

and

$$D^2(\bar{P}) = \frac{\sigma^2}{n}.$$

A trivial but quite important consequence of those formulas is twofold. First of all, since the expected value of this average is equal to μ, this random variable is not biased. Second, if one increases the sample size (the value of n), then the precision of the estimate increases. For instance, increasing the sample size four times decreases the variance of $\bar{P}$ two times. Additionally, note that the variance tends to zero as $n \to \infty$. This last observation can be described in a much more precise and informative way by the following laws that work under the assumptions given above:

1. Weak law of large numbers:

$$\forall \varepsilon > 0 : \lim_{n\to\infty} \Pr\left(\left|\bar{P} - \mu\right| > \varepsilon\right) = 0.$$

2. Strong law of large numbers:

$$\Pr\left(\lim_{n\to\infty} \bar{P} = \mu\right) = 1.$$

3. Central limit theorem:

$$\lim_{n\to\infty} \sup_{z\in\mathbf{R}} \left|\Pr\left(\sqrt{n}\left(\bar{P} - \mu\right) \leq z\right) - \Phi(z/\sigma)\right| = 0,$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal random variable, that is, $\Phi(x) = \int_{-\infty}^{x} f(x)$, where

$$f(x) = \frac{\exp(-x^2/2)}{\sqrt{2\pi}}.$$

4. Law of iterated logarithm:

$$\Pr\left(\limsup_{n\to\infty} \frac{\bar{P} - \mu}{\sigma\sqrt{n \log\log n}} = \sqrt{2}\right) = 1.$$

Those four important laws can be collectively, but informally, summarized as follows: if n is sufficiently large, then $\bar{P}$ is near μ, its distribution is approximately normal, and deviations of $\bar{P}$ from μ almost surely are not larger than $\sigma\sqrt{2n \log\log n}$. However, in practice, one does not have the situation in which $n \to \infty$, that is, one cannot look at the infinite sequence of random samples. Fortunately, several inequalities allow us to calculate the risk of high deviations from the mean for a given value of n.

The first inequality from this family that can be used for any value of n is *Chebyshev's inequality*. Using this inequality, one can estimate the probability of deviation of $\bar{P}$ from μ using the following formula:

$$\Pr\left(|\bar{P} - \mu| \geq k\sigma\right) \leq \frac{1}{nk^2}.$$

Additionally, if $P(X)$ has a finite third moment (which is also often the case; in particular, if $P(X)$ is bounded, then it exists given the same assumptions we made for the expected value and the variance above), then *Berry-Esseen theorem* can be used that can be viewed as an extension of the Central Limit Theorem to finite samples:

$$\sup_{z \in \mathbf{R}} \left|\Pr\left(\sqrt{n}(\bar{P} - \mu) \leq z\right) - \Phi(z/\sigma)\right| \leq \frac{E|P(X) - \mu|^3}{2\sigma^3\sqrt{n}}.$$

Before we move forward, let us make a few observations. In the above formulas, the mean (μ) and the standard deviation (σ) of the distribution of the $P(X)$; variable is often used. In the context of sensitivity analysis, it is often challenging to determine them precisely, as they will typically depend on the values of the parameters that are changing in the sensitivity analysis. However, it is usually simple to determine some lower and upper bounds for $P(X)$. For instance, in our example decision tree depicted in Figure 3.7, assuming that one may only change probabilities in sensitivity analysis but keep payoffs fixed, the maximum possible payoff is 120, and the minimum payoff is 72; both can be seen in the payoffs presented in the corresponding terminal nodes.

Let us denote by P_ℓ the lower bound for the random variable $P(X)$ and by P_u its upper bound. Once some explicit values of P_ℓ and P_u are available, then one can use several additional inequalities. First, let us provide the formula for the *Chernoff bound*: for any $\delta > 0$,

$$\begin{aligned}
\Pr\left(\bar{P} \geq (1+\delta)\mu\right) &\leq e^{-n\frac{2\delta^2\mu^2}{(P_u - P_\ell)^2}} \\
\Pr\left(\bar{P} \leq (1-\delta)\mu\right) &\leq e^{-n\frac{\delta^2\mu^2}{(P_u - P_\ell)^2}}.
\end{aligned}$$

The above inequalities bound the probability of *percentage* deviations from the mean. If one is interested in the assessment of *absolute* deviations, then the following observation can be used. From *Popoviciou's inequality* we know that:

$$0 \leq \sigma \leq (P_u - P_\ell)/2.$$

The lower bound is achieved if the whole probability is concentrated in one point, and the upper bound is achieved when half of the probability is concentrated at each of the two bounds of the distribution. After substituting these bounds for the Chebyshev's inequality, we get that for any $\delta > 0$,

$$\Pr\left(|\mu - \bar{P}| < \frac{P_u - P_\ell}{2\sqrt{n\delta}}\right) \geq 1 - \delta.$$

Alternatively, if n is large enough (typically, $n > 1{,}000$ is enough), we can assume that the distribution of $\bar{P}$ is approximately normal by the Central Limit Theorem and Berry-Esseen theorem. Therefore, we can say that approximately

$$\Pr\left(2(|\bar{P} - \mu|) < \Phi^{-1}(1 - \delta/2)\frac{P_u - P_\ell}{2\sqrt{n}}\right) \gtrsim 1 - \delta.$$

Let us start with a simple example. Consider a decision tree given in Figure 3.8 (the plot is made in the raw mode). It has only one parameter, namely, p. We know that $p \in [0, 1]$, thus the expected payoff of the tree is bounded to the interval $[1, 2]$. Suppose that we can sample from some probability distribution p and want to find some bounds of the probability that the average value of 1,000 independent samples lies within $\pm 10\%$ of the true mean (Note that we do not need to specify this distribution!). We can calculate it using the above Chernoff bound, where we have $\delta = 0.1$, $n = 1{,}000$, and $P_u - P_\ell = 1$ to get that this probability is at least

$$1 - \exp(-20\mu^2) - \exp(-10\mu^2).$$

Using again the fact that the payoff is bounded to the interval $[1, 2]$, we can be sure that $|\mu| \geq 1$, so the probability can be further bounded as follows:

$$1 - \exp(-20\mu^2) - \exp(-10\mu^2) \geq 1 - \exp(-20) - \exp(-10) \geq 0.9999.$$

We can conclude that running $n = 1{,}000$ samples is enough to get an assessment of the mean that will be within the $\pm 10\%$ range of the true mean with the probability of at least 99.99%.

Now, let us switch back to the example model presented in Figure 3.7. For example, when one decides not to do the screening and give aid, the

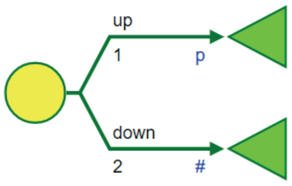


Figure 3.8: A view of a simple decision tree with only one parameter p (in `SilverDecisions` raw mode).

payoff is guaranteed to fall within $[75, 120]$. To find the number of samples that guarantee that the obtained mean $\bar{P}$ will be within $\pm 1\%$ of the true value of μ with at least 0.99 probability, one can use Chernoff bound. To do that, we need to solve the following equation for n, where we know that the mean is not less than 75:

$$\exp(-2n \cdot 0.01^2 \cdot 75^2/(120-75)^2) + \exp(-n \cdot 0.01^2 \cdot 75^2/(120-75)^2) \leq 0.01$$

It can be checked that this inequality is satisfied for $n \geq 16{,}615$.

On the other hand, if one is interested in an absolute deviation instead of a percentage deviation, we can use, for example, Chebyshev's inequality. In our example, the upper bound for σ is $(120-75)/2 = 22.5$. Then, assuming this bound, our goal is to find an assessment within ± 1 unit from the mean with a probability of at least 0.99. We get that $k = 1/\sigma = 1/22.5$ and thus $n = (22.5^2/0.01) = 50{,}625$. Finally, if we use the normal approximation for the same parameters, then we get that $\Phi^{-1}(0.01/2) = 2.576$, and so $n = 2.576^2(120-75)^2/3 = 3360$.

Note that we got various values of n as some of these bounds are stronger than others, but the winner might depend on a particular scenario considered. Indeed, all these bounds provide an upper bound for the probability, not an exact value. So, the values of n we obtained above guarantee that the probability is within the desired range, but perhaps it is also true for smaller values of n. There are some lower bounds for the probability that the random variable deviates from its expectation which, in turn, give us lower bounds for n, but here, we concentrate on sufficient conditions without investigating how optimal they are. In practice, because Chebyshev's and Chernoff's bounds are overly conservative, it is usually enough to use the normal approximation bound.

Finally, let us comment that computations performed using a computer have bounded precision. Therefore, despite the fact that the above bounds imply that, theoretically, one can get an arbitrarily good approximation by increasing the value of n; usually, one should not expect to get a relative precision of computations greater than 10^{-8}. Indeed, in practice, the numbers of samples that would lead to such a precision level are unreachable as the value of n would have to be proportional to the square of the inverse of this number, too large for a simulation to finish in a reasonable time.

3.2.3 Estimation of statistical parameters

The discussion in the previous sub-section concerned the analysis before the Monte Carlo experiment was performed. It is necessary to assess how many independent simulation runs (that is, the value of n) are needed to get the desired accuracy of the results. In this section, we turn out attention to the analysis after the experiment results are already collected.

Assume that we have a finite sample of size n from the simulation of $P(X)$ and that observations are independent (exactly such sampling is performed by `SilverDecisions`). Additionally, similarly to in the previous sub-section, in what follows, we will silently assume that the expected value and the variance of $P(X)$ exit. Denote this sample by p_1, p_2, . . . , p_n.

To assess the statistical parameter of interest, one can take one of the two following approaches: *point estimation* and *confidence interval* calculation. Point estimation aims to provide a single number that is the *best guess* of the true but unknown statistical parameter. On the other hand, the confidence interval is specified by a construction procedure returning two numbers. The *confidence level* of the confidence interval construction procedure is the proportion of obtained confidence intervals that contain the true value of unknown but fixed statistical parameters. Typically. 90%, 95% and 99% confidence intervals are constructed. (See [16] for an introduction to the theory of statistics.)

The formulas of the most popular point estimators are:

1. expected value: $\bar{p} = \sum_{i=1}^{n} p_i/n$;
2. median: median of sample p_i;
3. variance: $s_p^2 = \sum_{i=1}^{n}(p_i - \bar{p})^2/(n-1)$.

Procedures for creating confidence intervals are more complex. One simple rule exists for confidence intervals of mean for relatively large values of n. This rule is safe to apply when $n > 1{,}000$ as a rule of thumb, and such a number of replications can be easily performed in `SilverDecisions`. We choose the confidence level of the interval and select z^* equal to 1.645 for a 90% confidence interval, 1.96 for a 95% confidence interval, and 2.576 for a 99% confidence interval. Then, the confidence interval is given by:

$$\left(\bar{p} - z^*\sqrt{\frac{s_p^2}{n}}, \bar{p} + z^*\sqrt{\frac{s_p^2}{n}}\right).$$

Finally, let us mention a general procedure for any statistical parameter, bootstrapping. (For a complete exposition of this topic see, e.g., [7].) Here, we describe the simplest procedure, namely, the percentile bootstrap. As in all bootstrap methods, the percentile bootstrap relies on a rather simple and intuitive idea. Instead of making assumptions about the underlying distribution (unknown to us) from which our observations are sampled, one assumes the collected data is a true exact distribution (even though it is only its approximation). Informally speaking, we do several "fake experiments" using the observations from our sample as if they constituted the whole general population we should have sampled from if we could. For each of these experiments, we may compute any estimate of interest, for instance, the mean. Because of random and independent sampling, we get different means from different experiments, with some values more likely than others. After enough experiments, we get a distribution of bootstrap estimates which approximates the distribution of estimates from the true and unknown distribution.

It is advisable to have $n > 1{,}000$ independent replications of simulation generated by `SilverDecisions`. Next, we select the number of bootstraps replications B, which typically is taken as 10,000, sometimes even slightly more. In one bootstrap replication i, we sample n elements from the set $p_1, p_2, \ldots, p_n$ *with replacement* and calculate the point estimator of a statistical parameter of interest. Denote this value as b_i^p. Since we sample with replacement each time, we expect

$$n(1 - 1/n)^n \approx n/e$$

elements *not* to be selected in a single bootstrap sample, and some elements are selected more than once. After this procedure, we have B values b_i^p. To calculate A percent confidence interval (typically A is 0.9, 0.95 or 0.99, as discussed above), we calculate $(1 - A)/2$ and $(1 + A)/2$ percentile of bootstrapped point estimators b_i^p. Those percentiles define the lower and upper end of the required confidence interval.

To conclude this section, let us describe the bootstrapping procedure a bit more formally. Assume that P is a random variable whose cumulative distribution function is F. We are interested in analyzing the distribution of some statistics s of its independent n-element sample. Assume that the actual n-element sample of P is collected. Above, we have denoted the sampled elements as $p_1, p_2, \ldots, p_n$. Now, we may compute this sample's empirical cumulative distribution function and denote it $\hat{F}$. Note that the random variable describing the distribution of statistics P in an independent n element sample is a random variable $s(P_1, P_2, \ldots, P_n)$. As noted above, it is challenging to get the distribution of this random variable if F is not known. However, we have $\hat{F}$ that can be easily used. Denote the random variable following this distribution function as $\hat{P}$. Since for n large enough $\hat{F}$ is close to F; we may take that instead of sampling from P to get the distribution of $s(P_1, P_2, \ldots, P_n)$, we may sample from $\hat{P}$ (which is close to P). As a result, we get the distribution of $s(\hat{P}_1, \hat{P}_2, \ldots, \hat{P}_n)$, which then will be close to the distribution we are looking for. The crucial trick why this procedure is so simple and works well in practice is that it is very easy to obtain the sample of $s(\hat{P}_1, \hat{P}_2, \ldots, \hat{P}_n)$. Indeed, note that $\hat{P}_i$ is just a single element sampled from $p_1, p_2, \ldots, p_n$. So, all we need is to sample n elements from this set *with replacement* since we want $\hat{P}_1, \hat{P}_2, \ldots, \hat{P}_n$ to be independent.

In Section 3.3, we provide an example of the calculation of confidence interval using bootstrapping.

3.2.4 Common Random Numbers

So far in our discussion, we have considered only the analysis of a single payoff, that is, a single decision in the decision tree. However, it is often required to compare different decisions under uncertain parameters. Technically, two different decisions correspond to two transformations P_A and P_B of the underlying random variable X. Of course, one could consider random variables $P_A(X)$ and $P_B(X)$ independently, but often, in the analysis of decision trees, random variables $P_A(X)$ and $P_B(X)$ are positively correlated. This will happen if the same factor influences different decisions in a similar direction (although it does not have to be always true). For instance, in our decision tree from Figure 3.7, changing parameter `p_pos` either influences all the decisions in the same way or does not affect them (this happens for a decision where we decide to do nothing).

To understand the consequence of the positive correlation of $P_A(X)$ and $P_B(X)$, let us observe that:

$$D^2(P_A(X) - P_B(X)) = \sigma_A^2 + \sigma_B^2 + 2\sigma_A\sigma_B\rho,$$

where σ_i^2 is the variance of $P_i(X)$, and ρ is the correlation between $P_A(X)$ and $P_B(X)$. In consequence, the variance of $P_A(X) - P_B(X)$ is less than the sum of the two variances of underlying transformations if $\rho > 0$ (that is, assuming a positive correlation).

In such cases, if one wants to compare the expected payoff of P_A and P_B, it is desired to investigate the distribution of $P_A - P_B$ by evaluating P_A and P_B for the same realizations of random variable X. Having such data, one can, for instance, compare the mean of the distribution of $P_A - P_B$ with zero. Such a technique is known in the literature as Common Random Numbers (CRN) (or correlated sampling) and allows one to reduce the number of required replications of simulation. However, let us stress that if $\rho < 0$ (that is, $P_A(X)$ and $P_B(X)$ are negatively correlated), then CRN can "backfire", that is, it increases the variance, not decreases it, as intended.

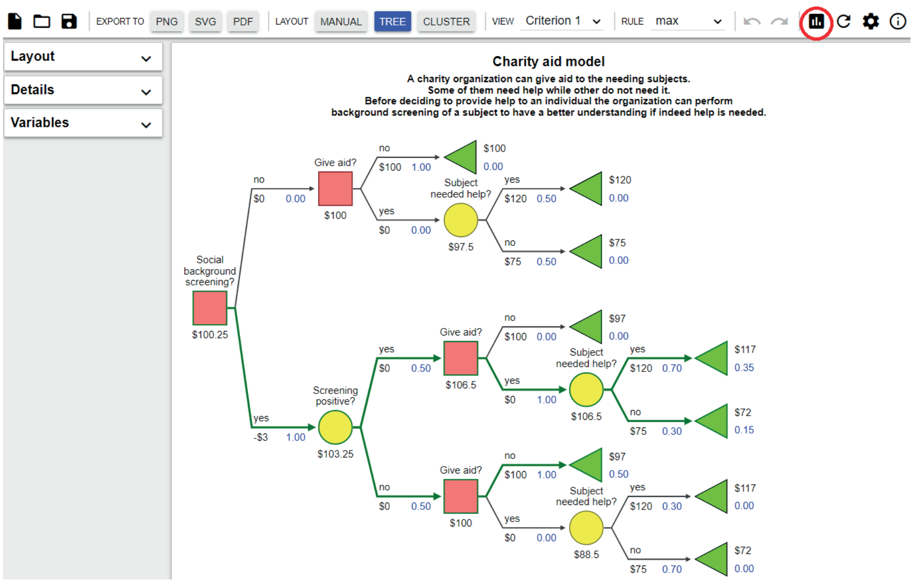
In `SilverDecisions`, when probabilistic sensitivity analysis is performed, samples are generated using Common Random Numbers. As it is described in the next section, one can export a CSV file from `SilverDecisions` and take advantage of CRN in the analysis of decision trees.

3.3 Sensitivity analysis of decision trees

Let us show an application of the theory presented in the previous subsection for the example model presented in Figure 3.6. Assume that there is some level of uncertainty about the values of the parameters `p_pos` (probability of positive screening), `p_pos_good` (probability of a good outcome, provided that screening is positive), and `p_neg_good` (probability of a good outcome, provided that screening is negative). Assume that for each parameter, the decision-maker assesses that they could deviate from their default values by 0.1 in both directions, so the ranges are: `p_pos`: $[0.4, 0.6]$, `p_pos_good`: $[0.6, 0.8]$, and `p_neg_good`: $[0.2, 0.4]$. For simplicity's sake, we assume that the probabilistic assessment of their uncertainty is that they are drawn from a uniform distribution over these intervals.

To perform the sensitivity analysis in `SilverDecisions`, one needs to click the button in the toolbar that is marked with the red circle (see Figure 3.9). The simplest way to check how sensitive the results are to the parameterization of the model is to use *'N-way sensitivity analysis'*. In such an analysis, as can be seen in Figure 3.10, one needs to specify which variables are to be analyzed, the range of variability of each of the variables, and the granularity of screening of these variables. In our example, we specified 11 levels for each variable, so they are all separated by 0.02. For instance, for the variable `p_pos` we considered the following values: 0.4, 0.42, 0.44, 0.46, 0.48, 0.5, 0.52, 0.54, 0.56, 0.58, and 0.6.

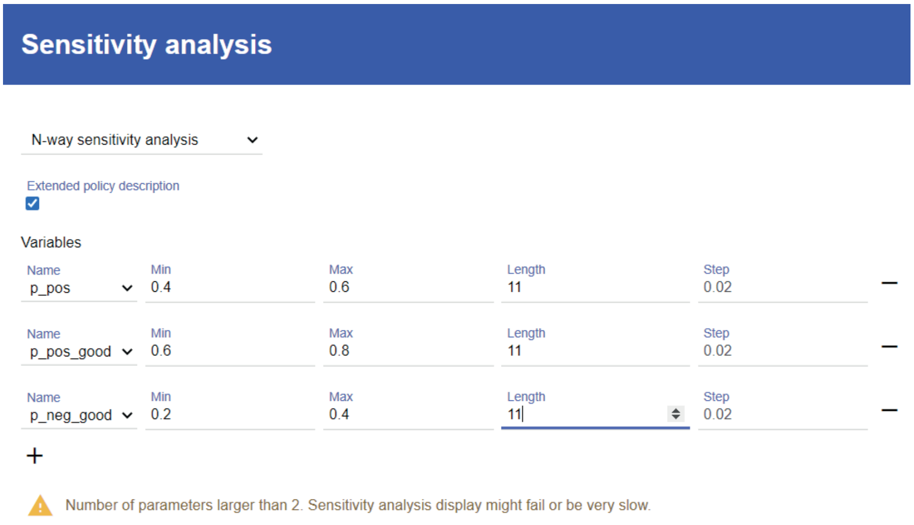
In the N-way sensitivity analysis, `SilverDecisions` considers all combinations of possible values of the analyzed variables. It will be $11^3 = 1{,}331$ scenarios to be explored in our case. Then, the expected payoff is calculated for each scenario and each possible decision rule. The results of the calculation are presented as a pivot table; see Figure 3.11. The highest payoffs are highlighted with red, while the lowest ones are highlighted with blue.



Charity aid model

A charity organization can give aid to the needing subjects.
Some of them need help while other do not need it.
Before deciding to provide help to an individual the organization can perform
background screening of a subject to have a better understanding if indeed help is needed.

Figure 3.9: Starting sensitivity analysis module.



Sensitivity analysis

N-way sensitivity analysis

Extended policy description ✓

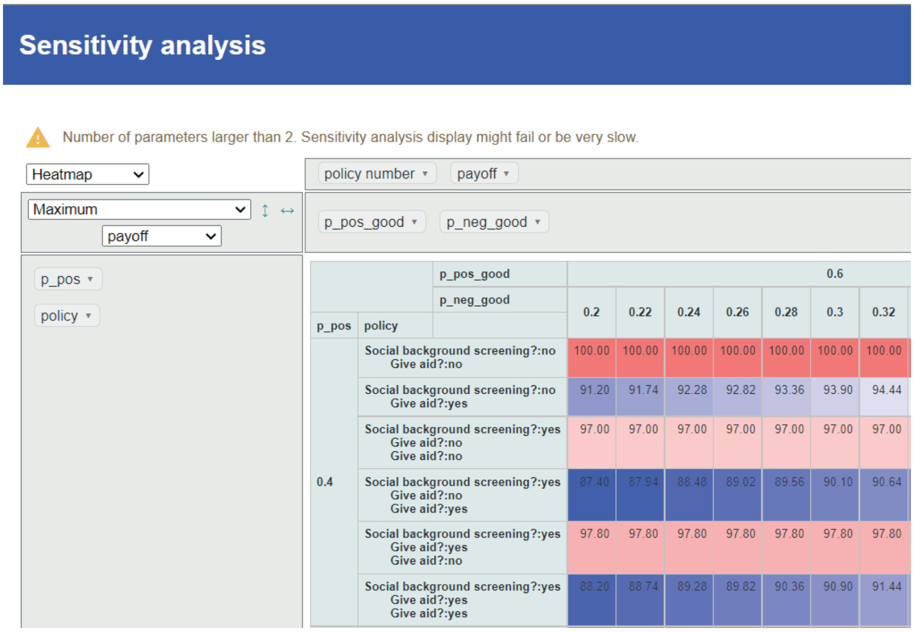
Variables

Name	Min	Max	Length	Step
p_pos	0.4	0.6	11	0.02
p_pos_good	0.6	0.8	11	0.02
p_neg_good	0.2	0.4	11	0.02

Number of parameters larger than 2. Sensitivity analysis display might fail or be very slow.

Figure 3.10: N-way sensitivity analysis configuration.

The pivot table summarizing the N-way sensitivity analysis contains a piece of complete information that the decision-maker might be interested in. However, as one can see, it might not be easy to assess it quickly, as it provides a lot of information. To simplify the analysis, `SilverDecisions` provides some alternative methods.



Sensitivity analysis

Number of parameters larger than 2. Sensitivity analysis display might fail or be very slow.

Heatmap

Maximum

payoff

policy number ▾ payoff ▾

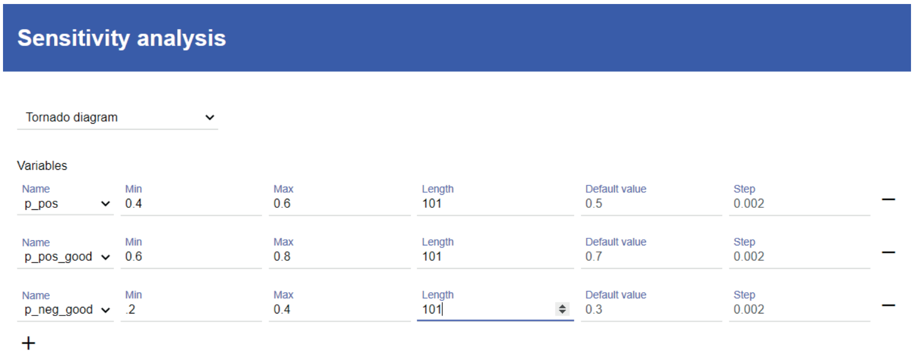
p_pos_good ▾ p_neg_good ▾

p_pos ▾

policy ▾

		p_pos_good	0.6						
		p_neg_good	0.2	0.22	0.24	0.26	0.28	0.3	0.32
p_pos	policy								
0.4	Social background screening?:no Give aid?:no		100.00	100.00	100.00	100.00	100.00	100.00	100.00
	Social background screening?:no Give aid?:yes		91.20	91.74	92.28	92.82	93.36	93.90	94.44
	Social background screening?:yes Give aid?:no Give aid?:no		97.00	97.00	97.00	97.00	97.00	97.00	97.00
	Social background screening?:yes Give aid?:no Give aid?:yes		87.40	87.94	88.48	89.02	89.56	90.10	90.64
	Social background screening?:yes Give aid?:yes Give aid?:no		97.80	97.80	97.80	97.80	97.80	97.80	97.80
	Social background screening?:yes Give aid?:yes Give aid?:yes		88.20	88.74	89.28	89.82	90.36	90.90	91.44

Figure 3.11: N-way sensitivity analysis results.



Sensitivity analysis

Tornado diagram

Variables

Name	Min	Max	Length	Default value	Step
p_pos	0.4	0.6	101	0.5	0.002
p_pos_good	0.6	0.8	101	0.7	0.002
p_neg_good	.2	0.4	101	0.3	0.002

Figure 3.12: Tornado diagram analysis configuration.

The first of them is the *'Tornado diagram'*. In this diagram, as before, one needs to select variables to analyze, their ranges, and the granularity of the analysis, see Figure 3.12. However, this timeless information is provided as an output that might be easier to comprehend. First, the analysis is performed separately for each variable. For instance, if `SilverDecisions` analyzes the `p_pos` variable, it assumes that the remaining variables have their default values. The result of the Tornado diagram calculation is presented in Figure 3.13.

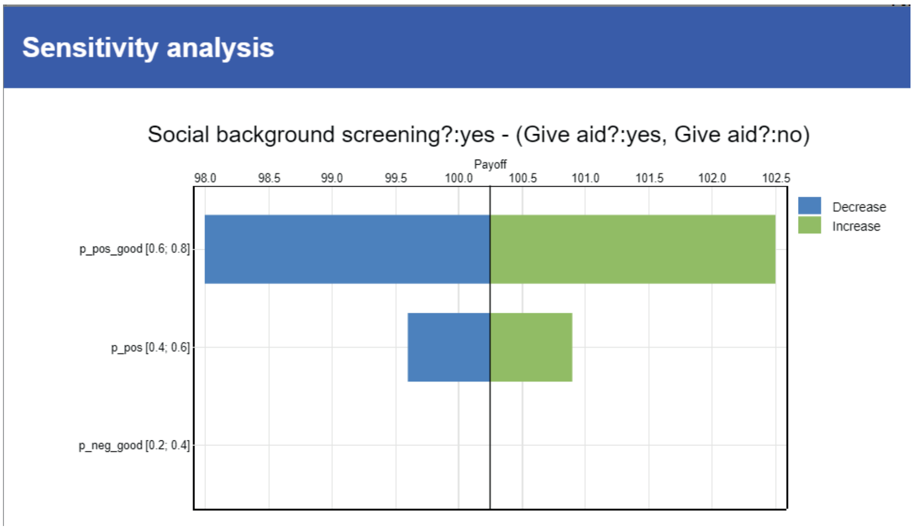


Figure 3.13: Tornado diagram analysis results.

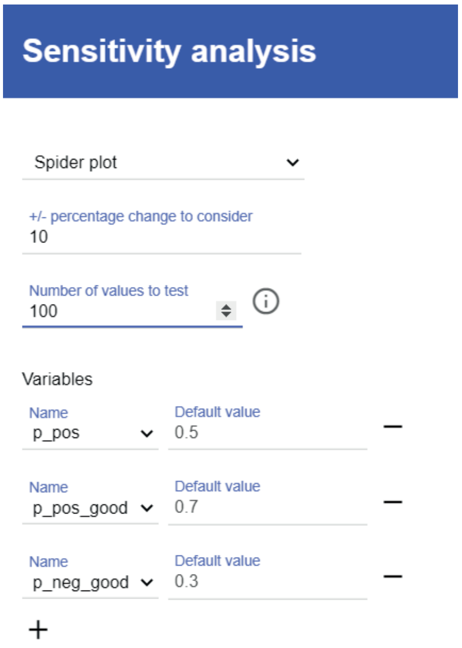
Moreover, the outcome is also more compact than what was provided by the N-way sensitivity analysis. Indeed, only the results for the optimal policy that is chosen for the default values of the parameters are presented. For such a default policy, we are informed about the range of outcomes that changing a given parameter can produce under the selected policy.

Another possibility, the *'Spider plot'*, is useful when the decision-maker is interested in the shape of the reaction of the model payoff to changes in the parameters. To create such plots, one needs to provide a common percentage deviation for each variable, its starting reference value, and the number of sampled points, see Figure 3.14.

As a result, the user can investigate how the payoff, under the default decision policy, changes as the given variable changes. In Spider plots, similarly to Tornado diagrams, changes to each variable are considered separately; see Figure 3.15. As one can note, for each variable, we get a line of different colors showing how the payoff changes with the percentage change of a variable.

Based on both the Tornado diagram and Spider plot, our results are not sensitive to changes in the `p_neg_good` variable. The reason is obvious as this option is not selected under the default policy. However, it can also be observed that the results are more sensitive to `p_pos_good` than to `p_pos`.

Now let us turn to *'probabilistic sensitivity analysis'* in which we assume that all parameters can change at the same time given their distributions. Recall that in our example, for simplicity, we assume that the distribution is uniform.



Sensitivity analysis

Spider plot

+/- percentage change to consider
10

Number of values to test
100

Variables

Name	Default value
p_pos	0.5
p_pos_good	0.7
p_neg_good	0.3

Figure 3.14: Spider plot analysis configuration.

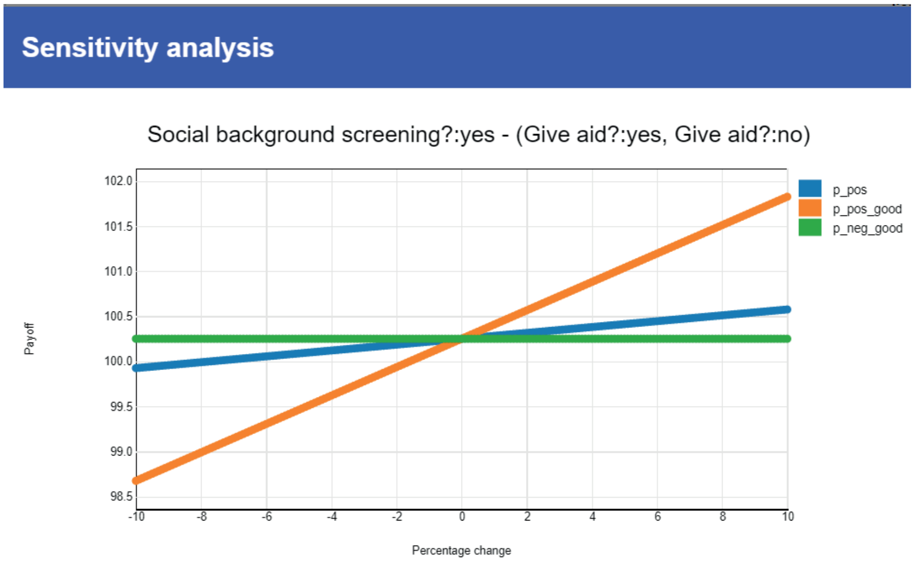
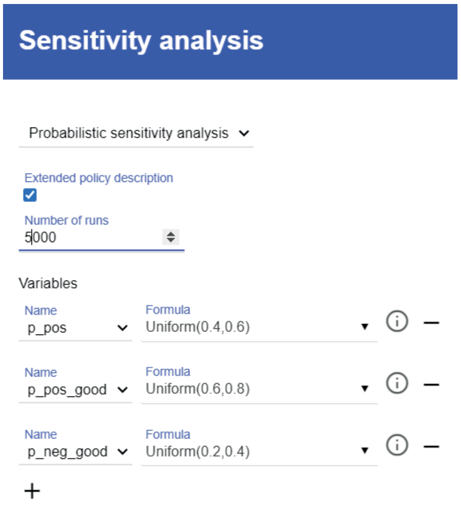


Figure 3.15: Spider plot analysis results.


Sensitivity analysis

Probabilistic sensitivity analysis

Extended policy description ✓

Number of runs 5000

Variables

Name	Formula
p_pos	Uniform(0.4,0.6)
p_pos_good	Uniform(0.6,0.8)
p_neg_good	Uniform(0.2,0.4)

Figure 3.16: Probabilistic sensitivity analysis configuration.

A first step in the probabilistic sensitivity analysis is a determination of the number of replications of the simulation that are needed to guarantee a high quality of an approximation. In the whole model, the range of possible payoffs is from 72 to 120; therefore, assuming our goal is to have the mean estimated with the deviation of at most 1 for each policy with 99% probability, using the normal approximation method, we calculate that we need at least $n = 3{,}823$ samples. In our experiment, we set the number of samples to 5,000, as this should still be fast enough to compete. The configuration of probabilistic sensitivity analysis is presented in Figure 3.16.

From Figure 3.17, we observe that under the assumed uncertainty initially optimal option, which is the best on average, is

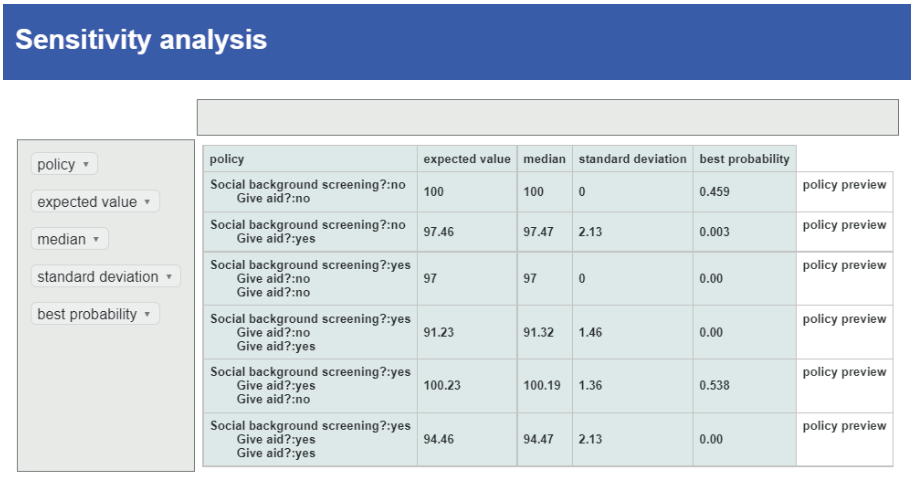
```
Social background screening?:yes - (Give aid?:yes, Give
                         aid?:no)
```

However, the probability that it is best is only 53.8%, and the option

```
Social background screening?:no - (Give aid?:no)
```

is only slightly worse on average but has a 45.9% probability of being optimal. All other options are not competitive. This evidence still favors the default option, but the decision-maker would probably, in such a situation, closely consider which one of the available options to choose, given that the advantage of one over the other is not significant.

Before we conclude this chapter, let us show how one could analyze data exported from probabilistic sensitivity analysis. This can be done



policy	expected value	median	standard deviation	best probability	
Social background screening?:no Give aid?:no	100	100	0	0.459	policy preview
Social background screening?:no Give aid?:yes	97.46	97.47	2.13	0.003	policy preview
Social background screening?:yes Give aid?:no Give aid?:no	97	97	0	0.00	policy preview
Social background screening?:yes Give aid?:no Give aid?:yes	91.23	91.32	1.46	0.00	policy preview
Social background screening?:yes Give aid?:yes Give aid?:no	100.23	100.19	1.36	0.538	policy preview
Social background screening?:yes Give aid?:yes Give aid?:yes	94.46	94.47	2.13	0.00	policy preview

Figure 3.17: Probabilistic sensitivity analysis results.

by clicking the Export CSV button in the probabilistic sensitivity analysis result plot. Below we show a sample of the exported CSV file (we show only the first few lines and have cropped the output on the right):

```
"policy_number","policy","p_pos","p_pos_good","p_neg_good","payoff"
1,"Social background screening?:no -- (Give aid?:no)",0.4826760816314803,0.7036809
2,"Social background screening?:no -- (Give aid?:yes)",0.4826760816314803,0.703680
3,"Social background screening?:yes -- (Give aid?:no, Give aid?:no)",0.48267608163
4,"Social background screening?:yes -- (Give aid?:no, Give aid?:yes)",0.4826760816
5,"Social background screening?:yes -- (Give aid?:yes, Give aid?:no)",0.4826760816
6,"Social background screening?:yes -- (Give aid?:yes, Give aid?:yes)",0.482676081
1,"Social background screening?:no -- (Give aid?:no)",0.4518324999084586,0.6782963
2,"Social background screening?:no -- (Give aid?:yes)",0.4518324999084586,0.678296
3,"Social background screening?:yes -- (Give aid?:no, Give aid?:no)",0.45183249990
4,"Social background screening?:yes -- (Give aid?:no, Give aid?:yes)",0.4518324999
5,"Social background screening?:yes -- (Give aid?:yes, Give aid?:no)",0.4518324999
6,"Social background screening?:yes -- (Give aid?:yes, Give aid?:yes)",0.451832499
```

As one can see, the columns consist of policy number (automatically generated by `SilverDecisions`), policy description, the sampled values of variables that are changing in the analysis (in our case, this is `p_pos`, `p_pos_good`, and `p_neg_good`), and the payoff for the given configuration of parameters. Note that `SilverDecisions` uses the same configuration of parameters for all policies before moving to the next configuration. So, as a result, we deal with a simulation that uses Common Random Numbers for all policies, as advised in the previous sub-section.

Such exported data can now be analyzed in a spreadsheet or using a programming language such as Python, R, or Julia. Below we show how one can use the Julia language to calculate confidence intervals for the payoff for all available options, both using the parametric and the bootstrapped confidence intervals. We used Julia 1.6.3 [4], additional packages `DataFrames.jl` 1.2.2 [19] and `CSV.jl` 0.9.3 [28]. Suppose the reader is interested in learning how to use the Julia language to analyze their datasets. In that case, we recommend starting with the tutorials that can be found in the Julia Academy https://juliaacademy.com/. Here, we just briefly explain the necessary steps we make in our analysis.

In the code block below, we first load the required packages, namely, `CSV`, `DataFrames`, and `Statistics` (they should be installed first). Then, we read the `charity_psa.csv` file into a data frame, where the file should be generated by `SilverDecisions` probabilistic sensitivity analysis module and saved to the working directory where the analysis is run. Next, we check the number of rows in our data frame. We see 30,000 rows, which exactly is as expected, as we have six policies and requested 5,000 repetitions of the simulation for each of them. Finally, we make sure that we have read the data frame column names correctly by displaying them.

```
julia> using CSV, DataFrames, Statistics

julia> data = CSV.read("charity_psa.csv", DataFrame);

julia> nrow(data)
30000

julia> names(data)
6-element Vector{String}:
 "policy_number"
 "policy"
 "p_pos"
 "p_pos_good"
 "p_neg_good"
 "payoff"
```

As we now have the data imported into Julia, we may proceed with the analysis. We start with the parametric confidence intervals. The `ci99` function takes a vector and returns three values: its mean, the lower and the upper end of the 99% confidence interval for the mean following the formula we have discussed in the previous sub-section. Next, we want to apply this function to our data frame. To do so, we group our data by `policy_number`, pass the `payoff` column to our `ci99` function, and collect the result back as a table having three columns returned by this function. As a result, we get a data frame with six rows. In each row, we have any information about the policy number, the mean payoff, and its 99% confidence interval.

```
julia> function ci99(v)
           p = mean(v)
           s2 = var(v)
           n = length(v)
           z = 2.576
           return (mean=p,
                   lo=p-z*sqrt(s2/n),
                   hi=p+z*sqrt(s2/n))
       end
ci99 (generic function with 1 method)

julia> combine(groupby(data, "policy_number"),
               "payoff" => ci99 => AsTable)
6x4 DataFrame
 Row | policy_number  mean      lo        hi
     | Int64          Float64   Float64   Float64
 ----+------------------------------------------------
   1 |             1  100.0     100.0     100.0
   2 |             2   97.4623   97.3845   97.54
   3 |             3   97.0      97.0      97.0
   4 |             4   91.2284   91.1753   91.2816
   5 |             5  100.234   100.184   100.283
   6 |             6   94.4623   94.3845   94.54
```

Next, we turn our attention to computing these intervals using bootstrapping. To do so, we define the `boot99_10000` function that does compute a 99% confidence interval using 10,000 bootstrapping replicates. Note that the expression `rand(v, length(v))` in the code draws random elements from vector `v` (with replacement) the number of times equal to its length, exactly as required in the bootstrapping procedure. The rest of the code is similar to the one above. As one can see, for our example decision tree and 5,000 replicates of simulation, the parametric and the bootstrapped confidence intervals are very similar.

```
julia> function boot99_10000(v)
           b = [mean(rand(v, length(v))) for i in 1:10000]
           return (p=mean(v), lo=quantile(b, 0.005),
                              hi=quantile(b, 0.995))
```

```
       end
boot99_10000 (generic function with 1 method)

julia> combine(groupby(data, "policy_number"),
               "payoff" => boot99_10000 => AsTable)
6x4 DataFrame
 Row | policy_number  p          lo         hi
     | Int64          Float64    Float64    Float64
-----+------------------------------------------------
   1 |             1  100.0      100.0      100.0
   2 |             2   97.4623    97.3832    97.5434
   3 |             3   97.0       97.0       97.0
   4 |             4   91.2284    91.1757    91.2822
   5 |             5  100.234    100.183    100.283
   6 |             6   94.4623    94.3844    94.5403
```

There are, of course, many more other analyses that can be performed using the produced data. We encourage the reader to try to perform some of them. Our presentation and explanation of the commands we used to produce confidence intervals were very brief, as in this book, we concentrate on `SilverDecisions`. The reader interested in learning more about how one can analyze data using `DataFrames.jl` in Julia is encouraged to investigate its documentation which can be found at https://dataframes.juliadata.org/.

Chapter 4

Advanced topics

Michał Jakubczyk

In the current chapter, we present how `SilverDecisions` may be used in slightly more advanced contexts. First, in Section 4.1, we show how to calculate the value of the real option and the value of information. Such measurements are useful to understand the value of available courses of action to the decision-maker. Then, in Section 4.2, we show how `SilverDecisions` can be used to support decision-making with multiple criteria (e.g., cost and effect). In this way, `SilverDecisions` can be used, for instance, in a health technology assessment context where typically both economic and clinical aspects are accounted for in decision-making. Finally, in Section 4.3, we show how the time dimension of the decision problems can be captured more fully by allowing for the discounting, i.e., the future pay-offs have less value from the perspective of the current moment. We also show how non-standard discounting methods can be implemented in `SilverDecisions`.

4.1 The value of real option & of information

Typically, we solve decision problems that represent (with some simplifications) the actual state of affairs. Then, the structure of a decision tree represents the possible actions of the decision-maker and the reactions they may face. The probabilities in chance nodes represent the decision-maker's information about what reactions to expect.

The decision trees may also be used to learn whether (how much) we would profit from having additional actions to choose from (or how much we would lose if we lost a possibility to act in a given way) or how much we would gain by learning more about what state of the world to expect. In other words, we will learn now how to calculate the value of a real option and the value of information.

4.1.1 The value of real option

We introduce this concept using an example.[1] A family-run company producing wooden toys needs to decide what type of machine to lease. Machine A costs \$3000 a month and, depending on the wood quality, will produce 1000 (with bad wood, probability 20%) or 2000 (with good wood, probability 80%) toys. Each toy yields a profit (excl. lease cost) of \$2. Machine B costs \$5000 per month and will produce 1500 (bad wood) or 3000 toys (good wood), \$2 profit each.

The whole monthly batch can be sold locally or be exported at a fixed cost of \$1000. In the latter case, the revenue (excl. lease & export cost) will double with a probability of 30% (or stay unchanged, otherwise).

The wood is purchased once per month, and the quality is only learned after the purchase (i.e., it cannot impact which machine is chosen). The firm cannot change the machine based on learning the wood quality. We assume that the next batch's quality is independent of the previous batches (so the quality of previous batches cannot impact the selection of machines to lease in subsequent months).

The company thinks on a long-term basis and wants to maximize the expected profit.

The above situation is represented in Figure 4.1, and the decision tree can be downloaded from the book website as a `toys.json` file. It is a good practice to define parameters in the global scope, where possible, or locally in individual nodes if we update them because of past actions/reactions. In the present example, we defined the following parameters globally:

```
cA = 3000
cB = 5000
cEx = 1000
p = 2
ExSP = 2
prExS = 0.3
prGW = 0.8
```

denoting cost of A, cost of B, cost of export, revenue, multiplier due to export success, probability of export success, and probability of good wood, respectively. To simplify working with the tree, in all the non-root decision nodes, we define locally the variable `quant` equal to (from top to bottom, respectively) 1000, 2000, 1500, and 3000. Then this variable can be used in

[1] Motivated by [35], example 16.13.

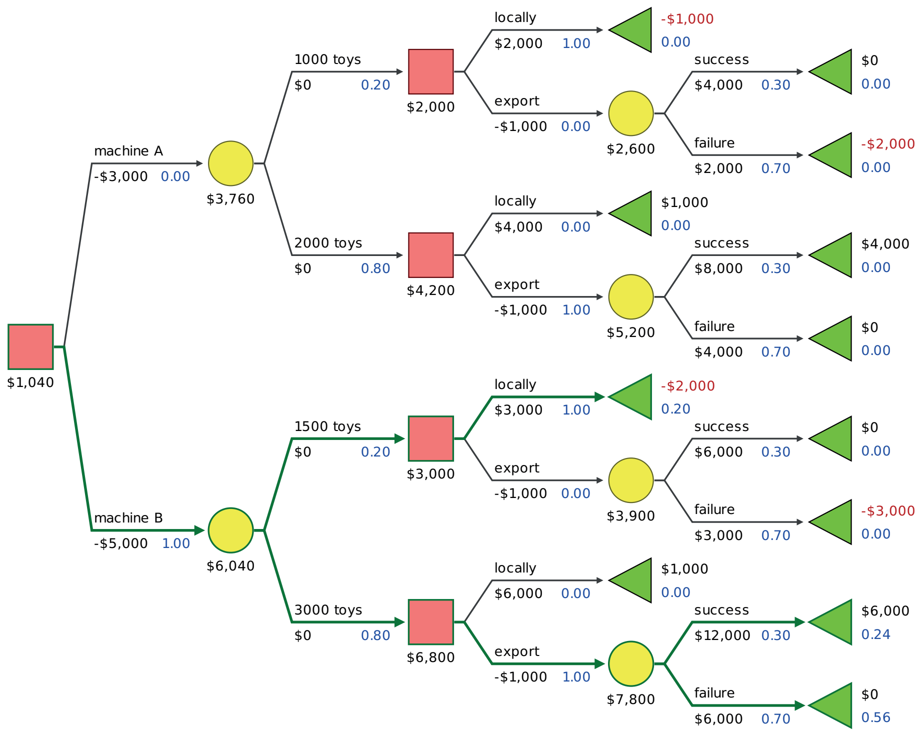


Figure 4.1: A toy-producing company tree.

formulas defining pay-offs: `quant * p` for local sale or export failure, and `quant * p * ExSP` for export success. This approach allows for speeding up the tree construction by copying and pasting parts of the tree.

As it is shown in the tree, the optimal strategy involves leasing machine B and exporting the output if 3000 toys have been produced. Because of the assumptions made in the task, the solution can be treated as a short-term one (what to do in a given single month) or a long-term one (which machine to keep on leasing): past experience with wood quality does not change the optimal strategy (assuming that the probability of wood quality being good is known to be 80%). The one-month expected profit amounts to $1040.

Treating the solution to the problem as a departure point, we will now value three real options the company has. First, how much is the real option to lease A (at the current market price) worth? The answer seems (and is) intuitive and easy. The company is not using this option in an optimal strategy. Hence, losing this option does not harm the company: the value of this option is zero. A more automatic approach to valuing a real option is to delete it from the tree and calculate the decrease in the value of the problem (expected pay-off of optimal strategy). The reader can immediately verify that deleting the part of the tree starting with the branch *'machine A'* does not change the value of the whole problem.

Secondly, how much is the real option to lease B (at the current market price) worth? The company is using this option; hence, losing it would probably make the company worse of. Thanks to the simplicity of the current problem, we can immediately see that then the company would have to lease A. The diagram shows that the expected profit would amount to \$3760 − \$3000 = \$760. The value of the problem decreases by \$1040 − \$760 = \$280. Hence, the real option of B being offered for lease (at the price of \$5000) is worth \$280. We can also verify this by increasing the cost of B (`cB=5280`): then there are two optimal solutions, one involving leasing A and the other involving leasing B.

Third, how much is the real option to export worth? We now refer to the option deeper inside the tree, and answering this may be more difficult. A naïve analysis would go as follows: *'we only use export after producing 3000 toys with B. Then the expected pay-off after export amounts to \$6800 (7800 − 1000), and the pay-off of selling locally amounts to \$6000. This means a loss of \$800; hence, the real option of export is worth \$800'.* The correct analysis must account for two additional factors: producing 3000 toys is not guaranteed (the loss of \$800 may not materialize), and the worsened prospects after choosing B may lead to not choosing B in the first place. The best way to value a real option is to remove it from a tree and see the impact on the value of the problem. The reduced tree is presented in Figure 4.2. Without the possibility to export, the firm will lease machine A and make, on average, \$600. Hence, the loss amounts to \$440. In other words, the company should be willing to pay up to \$440 every month to have the possibility to export.

It is important to appreciate we have calculated the value of the possibility to export from the perspective of the starting moment of the whole problem, before knowing if the company would be willing to export, i.e., before knowing if enough toys have been produced to make export attractive. It is a different question how much the possibility to export is worth once the company has already picked machine B and found 3000 toys have been produced. Then, the naïve reasoning presented above becomes the correct one. To put it differently, if machine B is selected, then the cost of export can increase by up to \$800, and the toys would still be exported if 3000 have been manufactured. The break-even situation is presented in Figure 4.3.

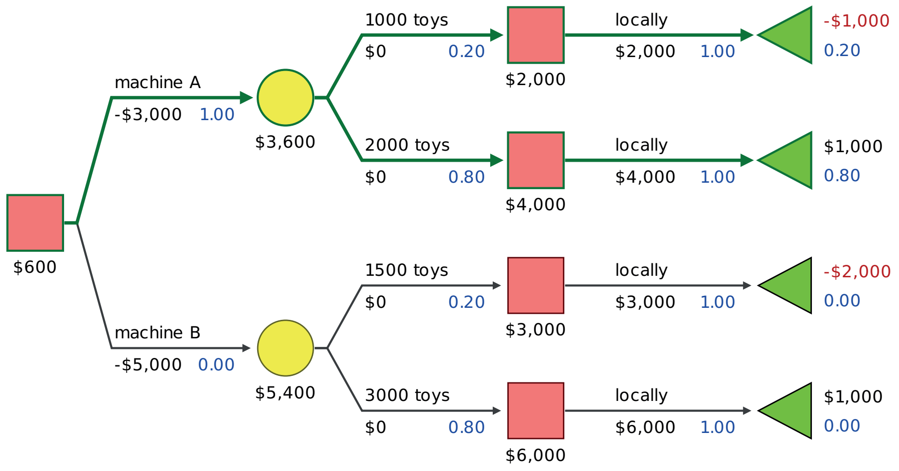


Figure 4.2: A toy-producing company tree, with no export.

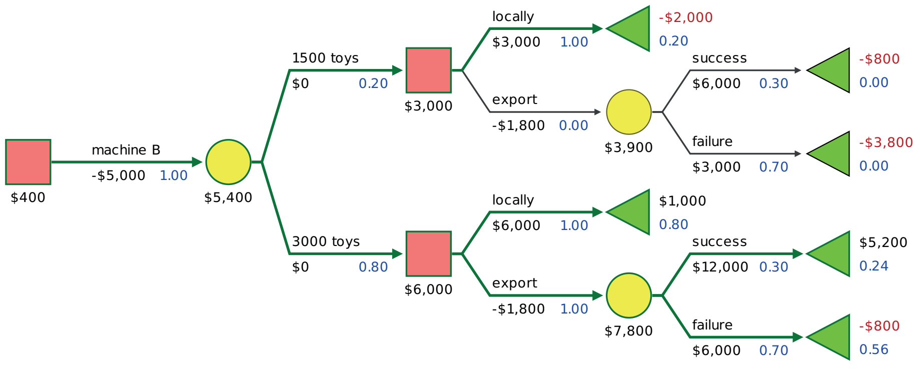


Figure 4.3: A toy-producing company tree, with machine B only and an increased cost of export.

4.1.2 The value of perfect information

Using decision trees, we can also calculate the value of knowing something in advance, i.e., the value of information. We will start with perfect information and then proceed with imperfect one.

Continuing our example from the previous subsection, we ask: what is the value of learning the wood quality in advance? The 'in advance' part denotes that the company finds out what the actual quality is, and only then does it have to choose the machine, while in the original problem, the company had to choose first and learn the quality later. This reversal can be represented by starting a decision tree with a chance node representing the learning of uncertain wood quality. The chance node should have to edge stemming out, representing the good or bad quality.

It is not always immediately clear to everyone what the probabilities assigned to these edges should be. Importantly, the structure of the quality of batches has not changed, i.e., still 80% are good. The company simply learns the quality earlier; hence, in 80% of the cases, the perfect information should sound ‘good quality’.

After learning the quality, the actions available to the company do not change. Therefore, it is typically easiest to copy-paste the original tree as subtrees after the root chance node.

There is only one more thing to take care of: the original tree encompassed the possibility of the quality turning out to be good/bad after the machine has been chosen. We must make sure the outcome of these chance nodes is consistent with what the company has learned in the root node (the information obtained in the beginning is perfect). This is done by setting the probabilities to 0% or 100%, as needed. For example, in the half of the new tree, after learning the quality would be bad, we need to make sure machine A produces 1000 toys and machine B produces 1500 toys.

The resulting tree is illustrated in Figure 4.4 (file `toysPIquality.json`). Using variables makes things easier: we copy-paste the original tree to the new root chance node; we can use `prGW` to set the probabilities in the root node; in root’s children we update either `prGW=0` (upper child) or `prGW=1` (lower child). The formulas take care of the rest. Obviously, we can simplify the structure by removing the parts of the tree which are not reached due to some probabilities equal to zero. Nevertheless, when working with software, it is actually easier to leave the pasted tree unchanged and simply overwrite the values of the parameters.

The value of the new problem amounts to \$1240: hence, the perfect information leads to an increase of \$200. This difference measures the expected difference in outcomes between two situations: 1) when the company has to decide not knowing the batch quality (and will choose machine B, see Figure 4.1), 2) when the company learns the quality first and then selects either A (20% of the time) or B. This value is typically called the *expected value of perfect information (EVPI)*. Importantly, the information is valuable because it impacts the decision-maker’s actions. Should the decision-maker make the same choices, whatever content the perfect pieces of information contain, then the information would have no economic value (and would only satisfy the curiosity).

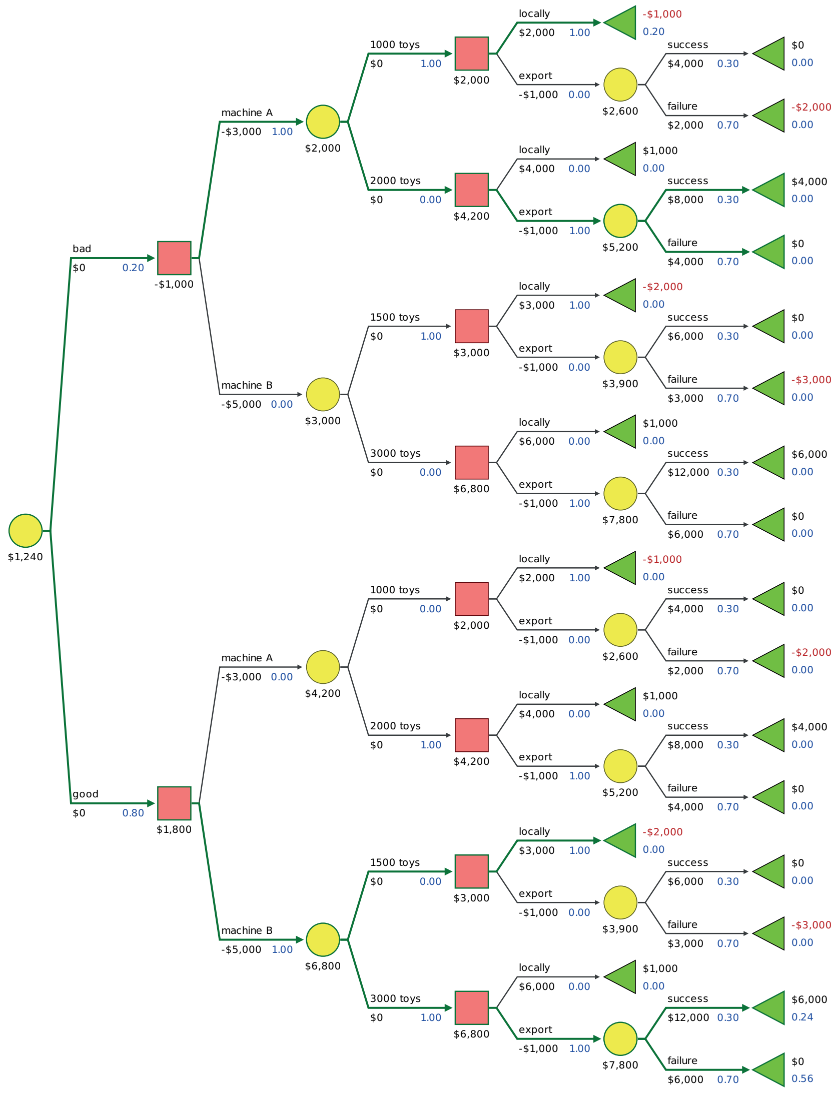


Figure 4.4: A toy-producing company tree, with perfect information on the wood quality in advance.

The reader is encouraged to calculate the EVPI on the export's success (or not). Then, the structure of the tree will be identical to Figure 4.4, except for the labels (also, the parameters will differ). Finally, the reader may calculate the EVPI of two uncertainties simultaneously: the wood quality and export outcome. Then we need to represent that the decision-maker learns two things before having to make any decision, which requires a root chance node with more edges or two consecutive chance nodes. The idea of a solution is hinted at in Figure 4.5.

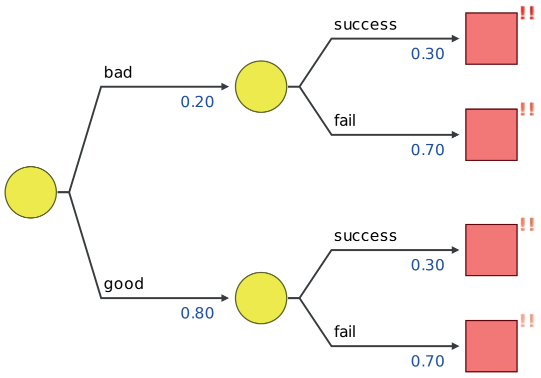


Figure 4.5: A part of the tree for toy-producing company with two pieces of info

4.1.3 Imperfect information, working with conditional probabilities

In the previous subsection, we have learned how to calculate the value of perfect information. The information is rarely perfect in real life: what really happens may occasionally differ from what the information foretold. Still, EPVI is a useful tool in decision-making, as it gives the upper bound on how much it is worth to pay for any information. In this subsection, we present how to precisely calculate the value of imperfect info.

We work with a new example, a version of a standard problem often used to teach decision trees [2]. An investor owns a land plot that may contain shale gas layers (as half of the neighboring plots contain gas, the investor estimates the probability of layers at 50%). If the transaction is finalized immediately, the investor has just got an offer to sell for \$750,000. The investor can also build a facility to exploit the layers for \$250,000. If gas is present, the revenue will amount to \$2,500,000. Otherwise, there is no revenue, and the plot cannot be sold (trying to sell a plot with a facility built can only mean there is no gas). The investor is risk-neutral and wants to base her decisions on expected value maximization.

[2] For example, see [31].

The above problem is illustrated in Figure 4.6 (all pay-offs in '000s dollars). The owner should build the facility, which yields \$1 million on average.

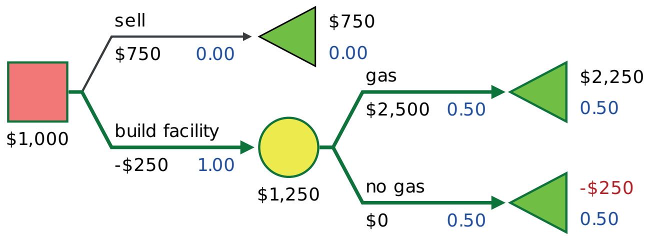


Figure 4.6: A simple decision problem of a land-plot owner (pay-offs in '000s dollars.)

Based on the considerations in the previous subsection, we create a tree that represents the problem with perfect information on the presence of gas, see Figure 4.7. The owner should immediately sell if she learns there is no gas and should exploit otherwise. The EVPI amounts to \$500,000 (the difference between the value of two problems). The standard interpretation holds: if we imagine multiple companies facing a decision problem as described, the first group of companies does not know if the gas is present (hence, all of them decide to build the facility, and on average half of them find gas) and the second group learns about the presence in advance (about half of them build the facility and find gas, and half of them sell the empty plot), then on average, the first group will make \$500,000 more.

The information is imperfect if it might disagree with the true state of affairs. For example, in Figure 4.4, the information may forecast the wood quality as being good, while in reality, it is bad. In the current example, we can imagine a geological test that attempts to detect the presence of gas but may occasionally give false results. When speaking of diagnostic tests, we can measure the performance of the test by two parameters: *sensitivity* and *specificity*. The readers can find a more detailed introduction to the concepts related to conditional probabilities in [13].

In our case, sensitivity measures how good a test is at detecting gas when the gas truly is present (i.e., how sensitive the test is to the presence of gas). More formally, sensitivity is measured as the probability of a test giving positive results (suggesting *yes*) given that the oil is present:

$$\text{sensitivity} = P(\text{test positive} \mid \text{gas present}), \tag{4.1}$$

where the vertical line, |, means 'given that' or 'conditioned on'.

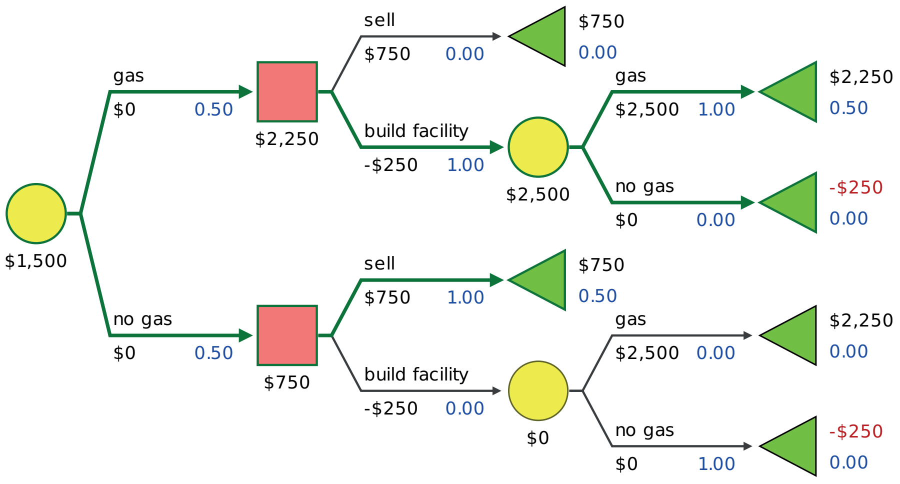


Figure 4.7: The decision problem of a land-plot owner with perfect information (pay-offs in '000s dollars.)

Specificity measures how good a test is at detecting a lack of gas when the gas truly is absent (i.e., how the test reacts specifically to the presence of gas, and not some other things). More formally:

$$\text{specificity} = P(\text{test negative} \mid \text{gas absent}). \tag{4.2}$$

If sensitivity is lower than 100%, then sometimes the result will be negative even for a gas-bearing field; hence, the negative result does not preclude the gas presence. Analogously, if specificity is lower than 100%, then the result may be positive even if gas is missing; hence, the positive result does not guarantee the presence of gas.

In the present example, we assume sensitivity = 90% and specificity = 70%. From the decision-maker's perspective, we are rather interested in the probability of gas presence/absence given the test result is positive/negative, i.e., we need conditioning in the opposite direction than in the definitions of sensitivity/specificity. `SilverDecisions` helps in recalculating the probabilities in the following way. We can represent the relations between two random events – whether or not the gas is present and what the result of the test is – with a *probability tree*, i.e., a simplified decision tree lacking decision nodes. The current situation is visualized in Figure 4.8 (upper part). The probabilities in the first layer are unconditional: they are the probabilities of gas being present/absent without any additional information. The probabilities in the second layer are conditional on the presence (upper part) or absence (lower part) of gas; for example, in the tree, we see the probability of a positive test result condition on gas being present amounts to 90%.

If we right-click on the root and select the *'Flip subtree'* option, the ordering of events will be changed. Now, the first layer presents the unconditional probability of the test being positive/negative, and the second layer presents the probabilities of gas being present/absent condition on the test result. For example, the probability of a test being positive amounts to 60%. If the test is positive, the probability of gas being present increases (from a priori 50%) to 75% (called a posteriori probability, after additional information has been gathered). If the test is negative, the probability decreases to 12.5%.

In our case, we had two possible states of the world (gas present or absent), and the test could yield two results (positive or negative); hence, the probability tree had $2 \times 2 = 4$ terminal nodes. We could generally consider bigger trees with k states of the world and m test outcomes (hence, $k \times m$ terminal nodes). To flip such trees, we need to make sure all the first (k) layer chance nodes have the same number of children (m), and the labels along edges leading to these children are the same (SD must be informed which conditional events denote the same thing). Flipping the tree twice recovers the original tree structure.

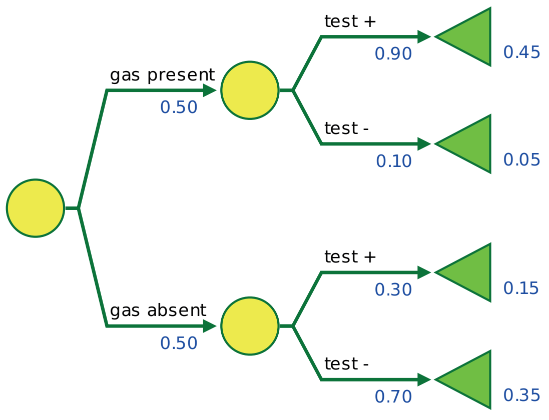
In the current version of `SilverDecisions` (as of January 2022), if there are formulas in the probability tree, they are replaced with values when flipping (and will not be recovered if the tree is flipped again). Probability trees are usually used with no pay-offs. If there are pay-offs, after the flip, the pay-offs are attached to the nodes leading to terminal nodes. In this respect, flipping the tree twice may not recover the original pay-offs (but the total pay-offs of terminal nodes will not change).

Because the formulas are lost when flipping a tree, it is worthwhile to learn the formulas used in the operation, in order to construct the flipped trees manually using parameters. In general case, there are k states of the world (exactly one obtains), denoted $S_1, \ldots, S_k$. There are m test results (exactly one obtains): $T_1, \ldots, T_m$. The input parameters are $P(S_i)$ for $i = 1, \ldots, k$, and $P(T_j|S_i)$ for $i = 1, \ldots, k$ and $j = 1, \ldots, m$. The formulas for conditional probability yield:

$$P(T_j|S_i) = \frac{P(T_j \cap S_i)}{P(S_i)}. \tag{4.3}$$

We then immediately get

$$P(S_i|T_j) = \frac{P(S_i \cap T_j)}{P(T_j)} = \frac{P(T_j|S_i) \times P(S_i)}{P(T_j)}. \tag{4.4}$$



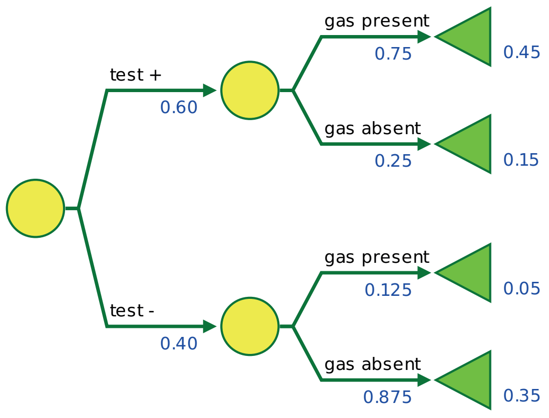


Figure 4.8: Probability tree for the gas-plot example (upper part) and its flipped version (lower).

Further, notice that

$$P(T_j) = P(\bigcup_{i=1,\dots,k} T_j \cap S_i) = \sum_{i=1,\dots,k} P(T_j \cap S_i) = \tag{4.5}$$

$$= \sum_{i=1,\dots,k} P(T_j|S_i) \times P(S_i). \tag{4.6}$$

Returning to our example, $k = 2$ and $m = 2$, S_1 (S_2) denotes gas present (absent), T_1 (T_2) denotes positive (negative) test result. $P(S_1) = P(S_2) = 0.5$, $P(T_1|S_1) = 0.9$, $P(T_2|S_1) = 0.1$, $P(T_1|S_2) = 0.3$, and $P(T_2|S_2) = 0.7$.

Using the above formulas:

$$\begin{aligned}
P(T_1) &= 0.9 \times 0.5 + 0.3 \times 0.5 = \\
&= \text{sensitivity} \times \text{gas} + (1 - \text{specificity}) \times \text{no gas} = \\
&= 0.6 \\
P(T_2) &= 0.1 \times 0.5 + 0.7 \times 0.5 = \\
&= (1 - \text{sensitivity}) \times \text{gas} + \text{specificity} \times \text{no gas} = \\
&= 0.4 \\
P(S_1|T_1) &= \frac{0.9 \times 0.5}{0.6} = 0.75 \\
P(S_2|T_1) &= \frac{0.3 \times 0.5}{0.6} = 0.25 \\
P(S_1|T_2) &= \frac{0.1 \times 0.5}{0.4} = 0.125 \\
P(S_2|T_2) &= \frac{0.7 \times 0.5}{0.4} = 0.875
\end{aligned}$$

It is important to understand that the necessity to recalculate the probabilities in a specific decision problem often arises. Even though we need $P(S_i|T_j)$ rather than $P(T_j|S_i)$, the latter is more easily available: it measures only the characteristic of the test (and not of the specific decision problem at hand) and can be provided, e.g., by a test manufacturer. The former depends on the $P(S_i)$ and cannot be given abstracting from a specific context.

Having calculated the probabilities, we can come back to valuing the imperfect information (i.e., the test with non-perfect sensitivity or specificity). As when valuing real options, we need to construct a tree that represents the situation of a decision-maker who has the possibility to use the test. Just as with perfect information, the information may be valuable because the decision-maker can choose their behavior depending on the signal received. In our example, the decision-maker can sell the plot or try to exploit the plot. Just as with perfect information, we assume the imperfect information arrives immediately and does not change the selling price (or any other pay-off). Then the problem is represented in Figure 4.9.

SD enhances building a decision tree from a probability tree by injecting decision nodes. If you start with a flipped probability tree (lower part of Figure 4.8), then inject a decision node in the edges stemming out of the root and see how this operation speeds up, arriving at Figure 4.9. The value of the new problem (with imperfect information) amounts to \$1.275 million (the pay-offs in the figure are presented in ’000s), and so the EVII = \$1.275 million − \$1 million = \$275,000.

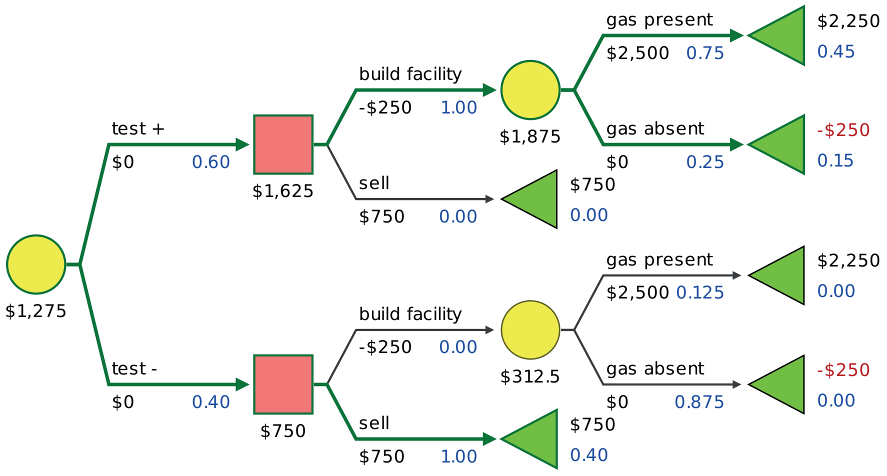


Figure 4.9: Representing the imperfect information in gas-plot problem.

4.2 Multiple criteria

4.2.1 Defining the problem with two criteria

Decision trees are often presented as tools supporting the managers in making business-oriented decisions, where the pay-offs are expressed in monetary terms only. Sequential decision problems with uncertainty also prevail in other contexts. An important example is health technology assessment (HTA), where available health technologies must be compared (e.g., to determine which should be reimbursed by the public payer) using two criteria: health and money. We present how two-criteria problems can be handled in `SilverDecisions` using an example from this area. Similar problems can arise in other contexts of public decision-making: e.g., environmental policy (where environmental gains must be juxtaposed with monetary cost), safety-related problems (e.g., crime rate vs. protection cost), or transport-related problems (delay time vs. cost of improving the infrastructure).

In HTA, the comparison must involve economic criteria (available resources are limited) and clinical ones (how much health can be bought). Sequentiality may stem from the clinical process encompassing several steps, e.g., consecutive lines of treatment, a possible sequence of complications or adverse events, or the diagnosis process preceding the treatment decisions. Uncertainty is an inherent part of the clinical process: a patient may be cured or not, the treatment may require hospitalization, the diagnostic test may end up positive or negative, etc. Please, see [10] for more information.

Costs of treatment are measured in monetary terms, and effects are typically expressed as quality-adjusted life-years (QALYs): a measure combining the longevity and health-related quality of life as a single number. If the decision-maker had a clear trade-off coefficient between the two, i.e., were able to precisely state their willingness-to-pay (WTP) for a unit of effect; then, both criteria could be merged into a single one: the net benefit, NB, is given as NB = effect × WTP − cost. Typically, the decision-maker has only an imprecise assessment of WTP and would like to learn the impact of WTP on the recommended strategy. For this reason, both criteria should be presented explicitly in the tree and the description of possible decision alternatives (strategies, policies).

`SilverDecisions` allows using two criteria in a tree. The user must switch the criterion to *'Two criteria'* in the upper part of the screen. Then, *'Multiple criteria'* section appears in the left pane. In that section, the user can change the names of the criteria and switch how they are ordered. To be consistent with the usual approach used in HTA, we should switch the name with the two-arrow button to make *'Effect'* the first criterion. The user can also switch the decision-making rule, selecting one of *'min-max'* (when the cost is the first criterion and effect the second one), *'max-min'* (with the order reversed), *'min-min'* (e.g., when cost and pollution are used as measured), or *'max-max'* (when two wanted goods are used, e.g., the throughput on highways and local roads) rules in the upper part of the screen. Below we will use the *'max-min'* rule to match the natural interpretation and order of criteria in HTA. The remaining cases are handled analogously, taking the reciprocal of WTP or adding the two criteria to form the NB, if needed.

We consider the following example. A patient with a given disease can be treated with one of four available treatments, A-D. They differ for the average (in a cohort of patients) cost ($) and effect (QALYs). The values are presented in Table 4.1. We can also wait to see if the condition improves without any treatment (probability 30%), which would provide 4 QALYs at no cost. If the condition does not improve (probability 70%), then the treatments A-D will cost slightly more and yield a smaller effect (also, see Table 4.1).

We assume the decision-maker's WTP=2, i.e., a gain of 1 QALY is considered worthy of spending two monetary units more. We assume for

	Immediate use		Delayed use	
Treatment	effect	cost	effect	cost
A	2	2	1	3
B	3	3	2	5
C	4	5	3	7
D	5	8	4	11

Table 4.1: An example of two-criteria problem, a list of decision alternatives A-F.

now (but will change this at the end of this subsection) that the decision-maker is entirely precise in this assessment; hence, the lower and upper bounds for WTP amount to 2. The problem is represented as a decision tree in Figure 4.10. Immediately in the graph, we see that it is NB-optimal to immediately use treatment B, yielding 3 QALYs for 3 monetary units (hence, the net benefit is equivalent to $2 \times 3 - 3 = 3$ monetary units). Assuming the decision-maker waits and the patient is still ill, it is optimal to use delayed treatment A (as indicated by blue 1.00 along the edge). If needed, this waiting and using A yields an average of 2.2 QALYs and costs 2.1 monetary units. In principle, we want to find the NB-maximizing strategy, but we also want to understand better the trade-offs between the cost and effect.

In total, there are eight strategies: an immediate or delayed use of A, B, C, or D. Each is characterized by the expected effect and cost (for immediate use of A-D, these are simply given as A-D's characteristics, for the delayed use the expected values come from averaging with the consequences of improvement without treatment). Just as in regular decision trees, the number of policies can increase enormously if several decision nodes can be reached along different paths, and the numbers of decisions in each are multiplied.

4.2.2 Interpreting the league table

SD presents the results of the two-criteria analysis in a *'league table'*, see Figure 4.11, and the plot in the cost-effectiveness plane (CE-plane), see Figure 4.13. In the league table, all the available strategies are enumerated. They are briefly described with actions taken in initial decision nodes. They are additionally enumerated to make it easier to refer to each strategy to its representation in the CE-plane (description below). The strategies are ordered concerning the expected effect, and the expected effect and cost are presented (we omit the word 'expected' below for brevity).

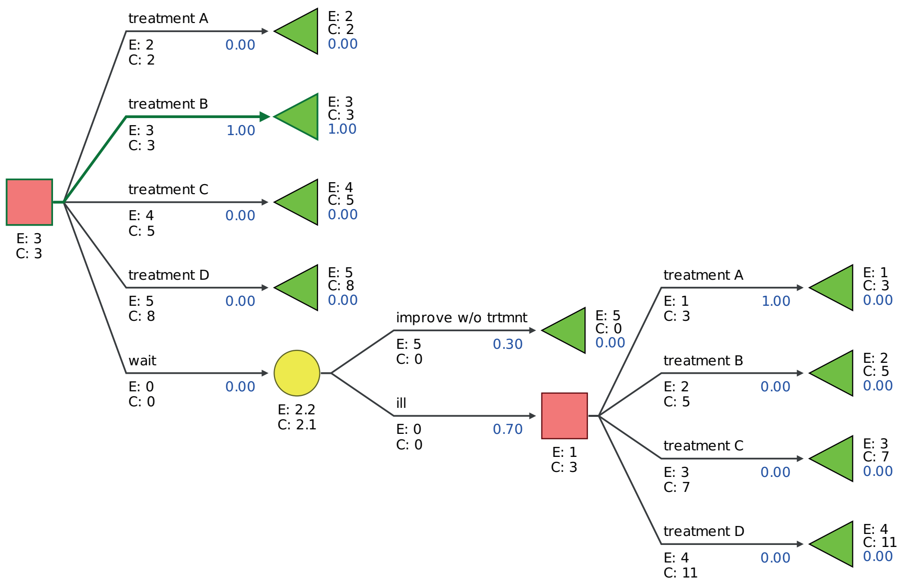
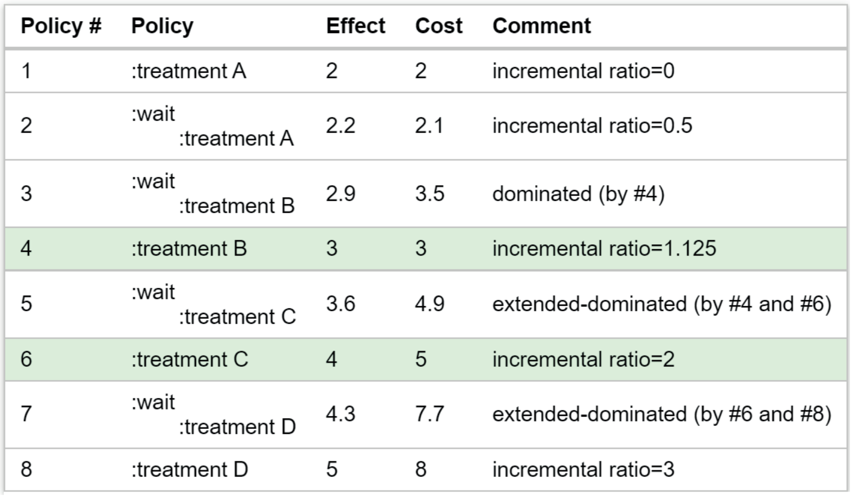


Figure 4.10: Choosing between multiple medical treatments using a decision tree with two criteria.



Policy #	Policy	Effect	Cost	Comment
1	:treatment A	2	2	incremental ratio=0
2	:wait :treatment A	2.2	2.1	incremental ratio=0.5
3	:wait :treatment B	2.9	3.5	dominated (by #4)
4	:treatment B	3	3	incremental ratio=1.125
5	:wait :treatment C	3.6	4.9	extended-dominated (by #4 and #6)
6	:treatment C	4	5	incremental ratio=2
7	:wait :treatment D	4.3	7.7	extended-dominated (by #6 and #8)
8	:treatment D	5	8	incremental ratio=3

Figure 4.11: League table for the problem of medical treatment selection for the willingness-to-pay equal to 2.

A strategy X is dominated if there is another strategy Y such that Y is not worse than X with respect to cost and effect and strictly better considering cost or effect. Then X yields smaller NB for any WTP > 0.

Dominated strategies are clearly labeled in the league table (e.g., strategy #3 is dominated by strategy #4).

A strategy X is extended-dominated if there are two strategies, Y and Z, such that X is dominated by some weighted average of Y and Z. The rationale behind such a definition is being that the decision-maker can create a derived policy of randomizing between Y and Z (with probabilities proportional to the weights in the average), and this resulting policy dominates X in the standard sense (a more compelling rationale is presented below). Extended dominance is clearly indicated in the league table; for example, strategy #5 is extended-dominated by strategies #4 and #6, with weights of 25% and 75%, respectively (the weights are not unique and are not shown in the league table).

The remaining (not dominated and not extended-dominated) strategies are considered in the effect-increasing order (also cost-increasing, since the dominated strategies are removed from consideration). If two (or more) strategies have identical expected effects and costs, they are given a single number in the league table and are considered collectively. Moving from one policy to the next results in an increase in cost and effect. We can then calculate the ratio of the cost increase to the effect increase to calculate the *incremental cost-effectiveness ratio (ICER)*, measuring how much the decision-maker needs to pay additionally for an extra unit of effect when switching to a more effective policy. More formally, if we consider a switch from policy i to $i+1$, and denote the cost and effect of policy i as C_i and E_i, respectively, then

$$\mathrm{ICER}_{i+1 \text{ vs } i} = \frac{C_{i+1} - C_i}{E_{i+1} - E_i}. \tag{4.7}$$

We often denote it by ICER_{i+1} for brevity, assuming it is calculated versus the previous in the order (not dominated and not extended-dominated) policy. In our example, $\mathrm{ICER}_2 = \frac{2.1-2}{2.2-2} = 0.5$, means that if a decision-maker decides to switch from policy #1 to policy #2, she will obtain an additional effect for an average price of 0.5. If the decision-maker is willing to pay this extra price, i.e., ICER $<$ WTP, then the switch is recommended. Notice that if $\frac{C_{i+1}-C_i}{E_{i+1}-E_i} < \mathrm{WTP}$, then also $E_{i+1} \times \mathrm{WTP} - C_{i+1} > E_i \times \mathrm{WTP} - C_i$, hence NB of policy $i+1$ is greater than of policy i: selecting the policy based on ICER is equivalent to maximizing NB. If ICER $=$ WTP, then switching or not are equally recommended. If ICER $>$ WTP, then not switching is recommended.

If the extended-dominated policies have been removed, ICERs will come in a non-decreasing sequence. Keeping the extended-dominated policies would violate this property. In our example, if policy #5 had not been removed we, would get

$$\mathrm{ICER}_{5\text{ vs }4} = \frac{C_5 - C_4}{E_5 - E_4} = \frac{4.9 - 3}{3.6 - 3} = 3.1(6), \tag{4.8}$$

$$\mathrm{ICER}_{6\text{ vs }5} = \frac{C_6 - C_5}{E_6 - E_5} = \frac{5 - 4.9}{4 - 3.6} = 0.25. \tag{4.9}$$

If $3.1(6) < \mathrm{WTP}$ and the switch to #5 is recommended, then also $0.25 < \mathrm{WTP}$ and a subsequent switch to #6 are recommended. Hence, an extended-dominated strategy is never the optimal one (the NB maximizing one). This property is another rationale for removing the extended strategies from considerations in the league table: the ICERs are calculated versus the last non-dominated (in any sense) strategy. In our example, ICER for strategy #6 is calculated versus strategy #4:

$$\mathrm{ICER}_{6\text{ vs }4} = \frac{5 - 3}{4 - 3} = 2. \tag{4.10}$$

In `SilverDecisions`, all the recommended strategies are highlighted in the league table. In our example, strategies #4 and #6 both maximize the NB for the assumed value of $\mathrm{WTP} = 2$. To make this clear, we present the NB of consecutive policies, NB_i, (also the dominated ones):

$$\begin{aligned}
\mathrm{NB}_1 &= 2.0 \times 2 - 2.0 = 2.0,\\
\mathrm{NB}_2 &= 2.2 \times 2 - 2.1 = 2.3,\\
\mathrm{NB}_3 &= 2.9 \times 2 - 3.5 = 2.3,\\
\mathrm{NB}_4 &= 3.0 \times 2 - 3.0 = 3.0,\\
\mathrm{NB}_5 &= 3.6 \times 2 - 4.9 = 2.3,\\
\mathrm{NB}_6 &= 4.0 \times 2 - 5.0 = 3.0,\\
\mathrm{NB}_7 &= 4.3 \times 2 - 7.7 = 0.9,\\
\mathrm{NB}_8 &= 5.0 \times 2 - 8.0 = 2.0.
\end{aligned}$$

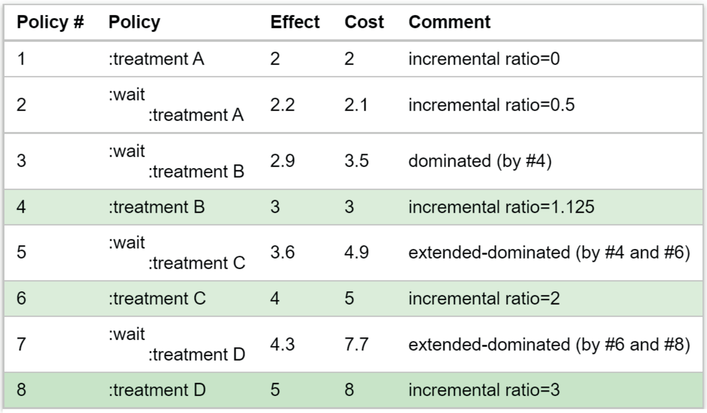
Two strategies maximize the NB and are recommended: #4 and #6, which confirm the conclusions of the ICER analysis (and these two have to agree, as shown above algebraically).

Notice that some dominated (#3) or extended-dominated (#5) strategies yield NB greater than some non-dominated strategies (e.g., #1). The logic behind removing the (extended-)dominated strategies is that they are not to maximize the NB for any value of WTP.

In our example, the WTP was defined as a single value, WTP=2. Still, two strategies, #4 and #6 are highlighted because both maximize the NB.

Typically, for a single-valued WTP, only a single strategy will be recommended (the peculiarity of our example was since $\text{ICER}_{6 \text{ vs } 4}$ was exactly equal to WTP).

The decision-maker may not precisely define the acceptable trade-offs between cost and effect. Then in the left pane of `SilverDecisions`, in the *'Multiple criteria'* section, the user may provide a lower and upper bound for WTP, differing from the default value, effectively defining a range of WTP values. This imprecision does not change the result of dominance (or extended-dominance) analysis. It may, however, be the case that depending on the selection of a specific value of WTP from the range of different strategies, maximize the NB. In the league table, all the strategies that maximize the NB for at least one specific value of WTP from the range will be highlighted, see Figure 4.12. For example, if we changed the lower/upper bounds to 1.5 and 3, respectively, then additionally strategy #8 will be highlighted. In the main pane in the tree, the strategy optimal for the default value of WTP is marked.



Policy #	Policy	Effect	Cost	Comment
1	:treatment A	2	2	incremental ratio=0
2	:wait :treatment A	2.2	2.1	incremental ratio=0.5
3	:wait :treatment B	2.9	3.5	dominated (by #4)
4	:treatment B	3	3	incremental ratio=1.125
5	:wait :treatment C	3.6	4.9	extended-dominated (by #4 and #6)
6	:treatment C	4	5	incremental ratio=2
7	:wait :treatment D	4.3	7.7	extended-dominated (by #6 and #8)
8	:treatment D	5	8	incremental ratio=3

Figure 4.12: League table for the problem of medical treatment selection for the non-degenerate willingness-to-pay range (1.5–3).

Summing up this part, the league table presents the consequences of all strategies available in a decision tree. It is shown which strategies can never be recommended for any WTP (are dominated). For the remaining strategies, the decision-maker is shown how the recommendation varies with accepted trade-offs between cost and effect.

4.2.3 Interpreting the CE-plane

The results of the league table are additionally presented graphically in the CE-plane, see Figure 4.13.

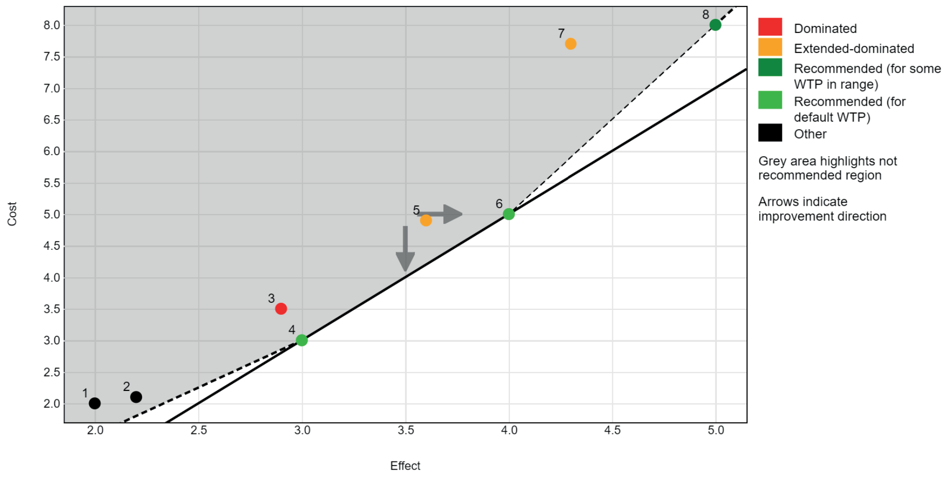


Figure 4.13: Cost-effectiveness plane for the non-degenerate willingness-to-pay range (1.5–3).

Typically, in HTA, the (expected) effect and cost define the abscissa and the ordinate, respectively. Individual policies are shown as points. The grey arrows indicate the direction of improvement along both dimensions. For example, if a *'max-min'* rule is used, then moving in the southeast direction (increasing the effectiveness and reducing the cost) is an improvement.

A strategy is dominated, if another strategy is located somewhere in the southeast. All dominated strategies are marked in red. For example, #3 is dominated by #4. A strategy is extended-dominated, if it lies to the northwest of a part of a segment connecting the points representing two other strategies. All extended-dominated strategies are marked in orange. If a strategy maximizes the NB for at least one WTP from the range, it is highlighted in green. Additionally, it is represented as light green if it maximizes the NB for the default WTP. All others (not dominated, extended-dominated, or recommended) strategies are drawn in black. They could become optimal if the decision-maker changed her preferences, i.e., changed her WTP.

The green points are connected with dashed segments, whose slope represents the ICER values. In our example, the segment connecting points representing strategies #4 and #6 has a slope equal to 2. From the point which corresponds to the cheapest of the recommended strategies (#4 in our example), a dashed ray is drawn in the southwestern direction with a slope equal to the lower bound of WTP. From the point representing the most effective of the recommended strategies (#6 in our example), another dashed ray is drawn in the northeaster direction with a slope equal to the upper bound of WTP. The region to the northwest of the rays and line segments is marked in grey. If any strategy is added to the problem that would fall inside (not on the border of) this region, the optimal decision will not change. If any point is added on the border of this region, it will also be recommended (for some value of WTP from the range). Still, the shape of the whole region would not change (and so the currently highlighted strategies would remain highlighted). If any point is added outside of this region, it would become recommended, and the region's shape will change (some of the currently recommended strategies might cease being recommended).

Finally, a solid line is drawn through the light-green point (i.e., a point representing the strategy optimal for the default value of WTP) with a slope equal to the default WTP.

4.3 Discounting

In all the previous sections, the the tree's structure only defined the ordering of feasible actions and reactions (which events follow which). Still, the actual time was separating the events. In particular, two edges did not necessarily represent the same amount of time (for example, a path of two edges might represent a shorter duration than some other single edge).

In real-life decision problems, the delayed pay-offs are often treated as less important: an investor may prefer a pay-off of \$1 million now, to \$1.1 million in 3 years. This time preference may result from uncertainty (catastrophic risk), the possibility of investing the money (e.g., in the bank account), or simple impatience. To introduce the time preference in decision-making, the present value (PV) of future pay-offs is often calculated, i.e., the future pay-offs are discounted. The PVs are then averaged with probabilities to calculate the expected PV of a policy, which is being maximized.

The present section shows how to use variables in order to introduce discounting in `SilverDecisions`. In three subsections, we show how to apply standard discounting, introduce hyperbolic discounting (more closely related to how regular people discount the future but result in inconsistent behavior), and discount pay-offs in two-criteria problems (and what inconsistencies it may cause).

4.3.1 Standard (exponential) discounting

We use the following example. A company may invest its free cash in developing and marketing new products in two ways: a conservative and a risky way. A conservative choice costs \$900,000 that must be spent immediately. Most likely (probability 90%), this investment will be successful and will yield \$500,000 after one year. If it is successful in the first year, it may continue being successful (probability 80%) and will yield an additional \$700,000 after another year. Similarly, it may (probability 50%) yield \$400,000 after three years.

A risky alternative requires \$400,000 immediately to cover the initial research and development cost. There is a 10% risk that the whole project will fail and have will to be cancelled; then, no revenue will be obtained. Otherwise (it will be known after half a year), an additional \$400,000 will be needed. Then, with a probability of 50%, the project will yield \$700,000 after the 1st, 2nd, and 3rd year, each (or no revenue with a probability of 50%).

The company is risk-neutral and discounts the future pay-offs with a discount rate of 5% annually and a standard formula (explained below).

The structure of the tree is rather straightforward, and the complete tree is presented in Figure 4.14 (focus on the structure and probabilities, for now). The chance nodes have been labeled to make them easier to refer to in the description (*'U1'*, *'U2'*, *'U3'* in the upper part, and *'D1'*, . . . , *'D4'* in the lower part). The tree can be downloaded from the book's website (`discount.json`).

The standard discounting assumes the present value (PV) of pay-off v_t obtained in time t, is given by $v_t \times \delta^t$, where $0 < \delta \leq 1$ is the discount factor. Often $\delta = \frac{1}{1+r}$, where $r \geq 0$ is the discount rate (clearly, $r = 0$ results in no discounting). For example, the annual (if t is expressed in years) discount rate of $r = 0.05$ denotes that \$100 now is equivalent to \$105 in one year time (i.e., the decision-maker would be indifferent between the two), or that \$100 in one year is equivalent to ca. \$95.24.

In `SilverDecisions`, discounting can be introduced using variables, e.g. in the following way. In the global scope, we define the variables:

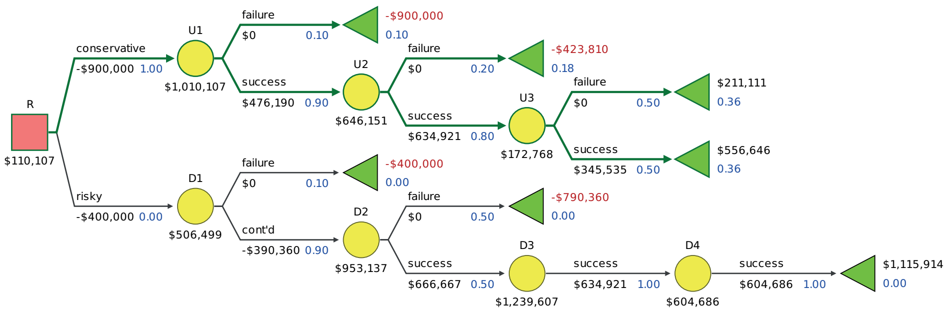


Figure 4.14: The impact of a discount rate on decision.

```
t = 0
rate = 0.05
delta = 1/(1+rate)
```

The two last variables are used only to make changing the discount rate easier and to shorten the formulas. The first variable, `t`, is crucial in handling the discounting; it is used to measure the passage of time to know how strongly to discount the pay-offs obtained in individual nodes. This variable is updated in every node to denote some time has passed. For example, in *'U1'* we have `t=t+1` (one year has passed), and in *'D1'* we have `t=t+0.5` (half a year has passed).

We then use `t` to define payoffs along the edges. For example, in *'U1'* in the lower edge we have the payoff given by the formula `5*10^5*delta^t` (i.e., $500,000 \times \delta^t$). Then, the payoff is reduced by a factor δ^t.

Defining `t` in the global scope, rather than in the root, makes it easier to work with subproblems of the original problem (i.e., with subtrees copied out of the initial tree). The formulas updating `t` will be kept in the subtree, and the definition of $t = 0$ in the global scope will remain. One will only need to update the formulas in the root of the new tree (remove the formula for `t` incremental updating). For example, we may cut the original tree in *'D1'* to represent the situation after the risky alternative has been chosen and the initial payment, $400,000, has been made (see Fig. 4.15). Notice that the pay-offs along corresponding edges have increased by a factor $(1 + 0.05)^{0.5}$, which represents the removal of a half-year discounting.

Using the sensitivity analysis (SA) functionality of `SilverDecisions` (see Section 3.3), we may inspect the impact of the discount rate on the recommendation. We need to move the definition `delta=1/(1+rate)` from the global scope to the root and then run the SA for rate. For example, in Figure 4.16 we present the results for `rate` between 0% and 10% with a step of 1 percentage point. We can see that the conservative strategy is better for a patient decision-maker (low discount rate). Then, for a discount rate of ca. 7%, the risky strategy becomes the recommended one.

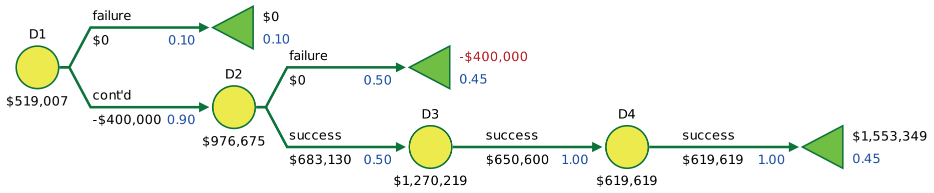
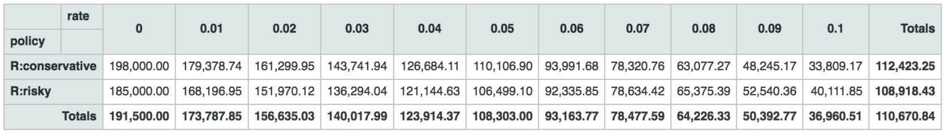


Figure 4.15: A subtree of the original problem with discounting.



rate / policy	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1	Totals
R:conservative	198,000.00	179,378.74	161,299.95	143,741.94	126,684.11	110,106.90	93,991.68	78,320.76	63,077.27	48,245.17	33,809.17	**112,423.25**
R:risky	185,000.00	168,196.95	151,970.12	136,294.04	121,144.63	106,499.10	92,335.85	78,634.42	65,375.39	52,540.36	40,111.85	**108,918.43**
Totals	**191,500.00**	**173,787.85**	**156,635.03**	**140,017.99**	**123,914.37**	**108,303.00**	**93,163.77**	**78,477.59**	**64,226.33**	**50,392.77**	**36,960.51**	**110,670.84**

Figure 4.16: Inspecting the impact of the discount rate on the recommendation.

4.3.2 Hyperbolic discounting and time inconsistency

Empirical observations suggest that the future pay-offs are often discounted differently (rather than using the formula shown above). When a pay-off is postponed, its PV first drops substantially but does not fall strongly with further postponements. To put it differently, the relative impact of the first postponement by one year (from the present moment till one year time) is greater than from one year ahead to two years ahead. This is sometimes represented as hyperbolic discounting: PV $= v_t \times (1 + rt)^{-1}$ (instead of PV $= v_t \times (1 + r)^{-t}$), or quasi-hyperbolic discounting: PV $= v_t \times \beta^{\mathbb{1}_{(0,+\infty)}(t)}(1 + r)^{-t}$, where $0 < \beta \leq 1$.

To see how to use hyperbolic discounting in `SilverDecisions`, consider the following example (not business-oriented, to stress that hyperbolic discounting prevails in mundane situations). A person has just prepared a bottle of a fine alcoholic beverage based on quinces (resembling tincture). It should now be aged for several months to acquire this special taste: the sooner consumed, the less utility the consumer will get. Assume the bottle was prepared at the beginning of January, it can only be consumed at the beginning of a month (say, when the person visits the family house where the bottle is stored), and the utility pay-offs are given in Tab. 4.2. After May, the aging no longer improves the taste.

month:	Jan	Feb	Mar	Apr	May
utility:	2	4	7	10	14

Table 4.2: Pay-offs depend on the time of consumption.

We assume the person discounts the future with hyperbolic function and $r = 0.5$ (we use this very high rate just for illustrative purposes, but in this non-business situation, we may say it results from an extreme impatience).

The structure of the decision problem is quite straightforward (e.g. there is no uncertainty) and is presented in Figure 4.17. In order to introduce discounting, we use the same trick as in the previous subsection: we define the variable `rate=0.5` and `t=0` in the global scope, update `t=t+1` in consecutive decision nodes, and use these variables when defining pay-offs. To simplify things, we also define the vector variable `u=[2,4,7,10,14]` in the global scope. As we may want to analyze subtrees of the original tree, and the `u[...]` is defined using the absolute time (i.e., specific months) and not the relative one (i.e., not the number of months since start), we define it in the root additional variable `[m=1]` to be able to calculate the absolute time in any specific node. With all those preparations, the payoff associated with drinking in any decision node is given by the same formula: `u[m+t]/(1+r*t)`.

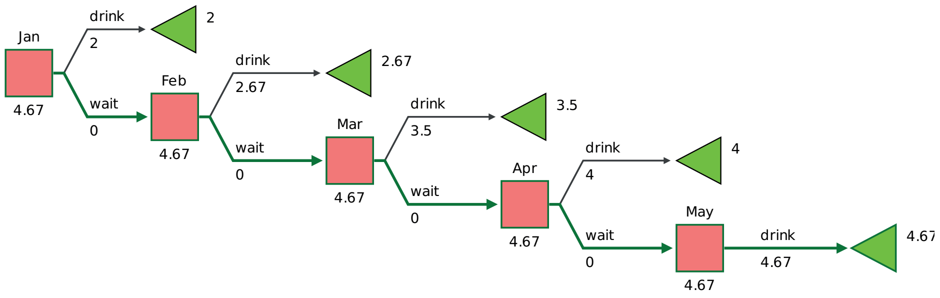


Figure 4.17: When to have a drink – the structure of a decision problem for a time-consistent person.

One can easily verify whether the tree has been set up correctly by changing the global scope definition of `rate` to `rate=0`. Then the pay-offs in individual months should be equal to those in Tab. 4.2.

Even with the high discount rate, it is recommended for a decision-maker to wait only until May and drink the bottle only then. Because the taste does not improve after May and there is a discount, it makes no sense to wait longer. And the possibility of waiting is not even included in the tree's structure. The reader may verify that adding more months will not change the solution.

In the literature on hyperbolic discounting, various types of decision-makers are considered [26]. A time-consistent decision-maker (TC-type) chooses only once and follows this initial decision. TC-type would indeed drink the bottle in May.

A naïve decision-maker (N-type) re-evaluates the problem in every decision node. One way to learn about N-type behavior in `SilverDecisions` is to directly represent consecutive subproblems by copying subtrees of the original tree, as shown in Figure 4.18. After copying and pasting, we need to provide two definitions of variables in the roots of the subtrees; for example, in the tree starting in February, we should have:

```
t=0
m=2
```

Redefining `t` guarantees that the future (relative to Feb) pay-offs are correctly discounted. Redefining `m` is needed to read correct values from the `u[...]` vector.

We can now see that the N-type will change her mind in April and drink the bottle then. This results from the relatively small difference between 10 and 14 in absolute pay-offs and a steep decrease of a PV when a pay-off is postponed from the present moment until the future. This changing one's mind (even though no new information was obtained) is dubbed *time inconsistency of decisions* and is extensively studied in economics. As we show, it can also be modeled with `SilverDecisions`.

Finally, the sophisticated decision-maker (S-type) can predict their inconsistent behavior and try to optimize earlier decisions because of anticipated future actions. In our example, the S-type would predict in March own drinking the bottle in April (even though it is optimal from the March perspective to delay until May, see Figure 4.18).

We can solve the problem using S-type eyes by modifying the subtrees proceeding from the last ones: we delete the parts we know will not be chosen, see Figure 4.19. For example, in the March-starting tree, we remove the possibility of waiting in April, as it will not be chosen. This removal leads to the S-type wanting to drink the bottle in March. By learning new actions, we modify earlier trees analogously. Eventually, the bottle will be emptied in March. A complete tree can be found in the file `alcoholMaster.json`.

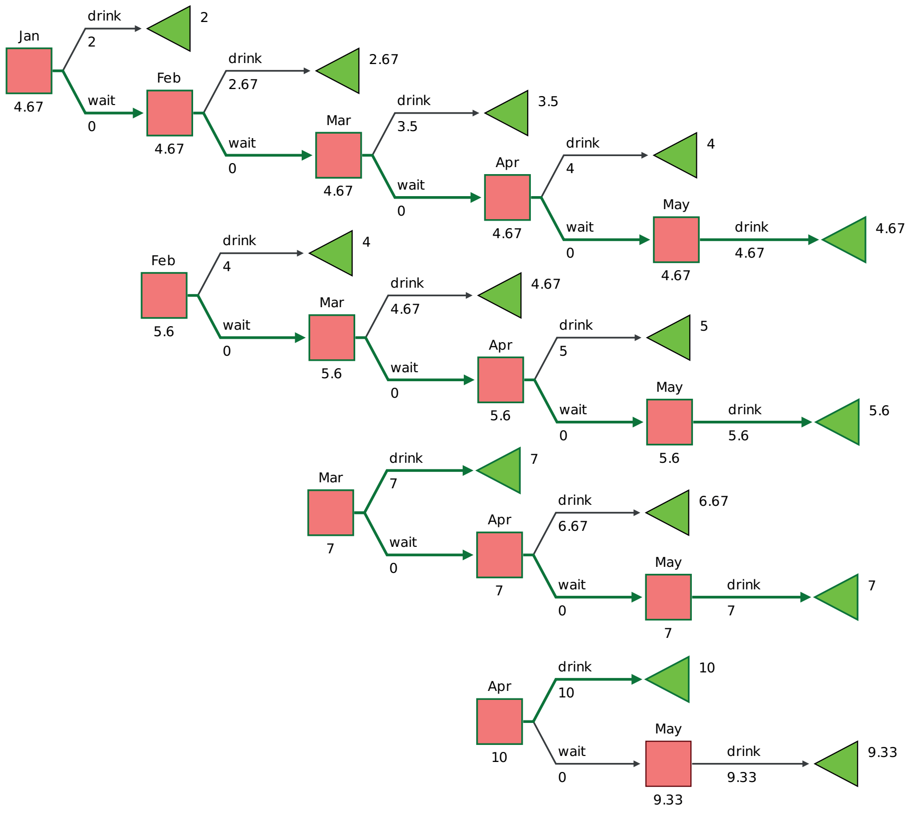


Figure 4.18: When to have a drink – an illustration for a naïve decision-maker.

4.3.3 Discounting in two-criteria problems

As presented in the previous section, `SilverDecisions` supports two-criteria problems. It is not obvious whether both criteria should be discounted at the same rate when facing such problems. In HTA, various arguments are presented, see [17]. One way or another, individual criteria can be discounted using the methods presented in the preceding subsections. We use this opportunity to present that when both criteria are discounted with different rates (but using exponential discounting) then – in specific settings – a time inconsistency of decisions can occur, just as with hyperbolic discounting. We use the following example.

A medical program can be introduced that will immediately cost 103 monetary units and yield one effect unit (say one QALY). The unit of effect is valued at 100 (obviously, the program is not recommended). Now assume that the decision-maker can postpone the introduction (but it can only

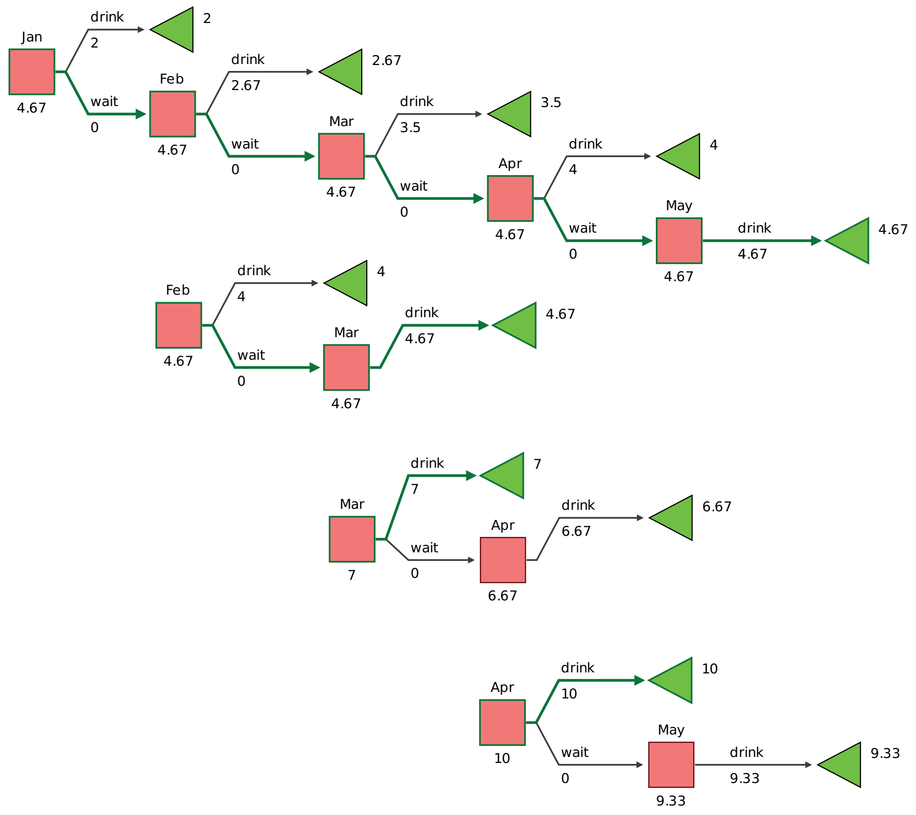


Figure 4.19: When to have a drink – an illustration for a sophisticated decision-maker.

be introduced once) by five years. The future is discounted (exponential discounting) with annual rates of $r_C = 0.1$ for cost and $r_E = 0.05$ for effect.

The above problem is easily represented as in Figure 4.20: the decision-maker can skip the program altogether, introduce it immediately, or wait. In case of waiting, the program can eventually be introduced or skipped. The discounting is done using variables `t` (denoting time and updated in the tree), `rE` and `rC` (representing discounting rates). The variables are used to define pay-offs in edges.

As seen in the tree, it is recommended to wait and introduce the program later. This is confirmed by the league table (reproduced in Tab. 4.3). The remaining options are enumerated explicitly in the league table.

What is interesting is that after waiting for five years the decision-maker faces the problem of comparing two simple strategies: to skip it altogether or introduce it now. It will be recommended to skip (gain of 1 effect valued at 100, at the cost of 103). Hence, the various discount rates can generate time

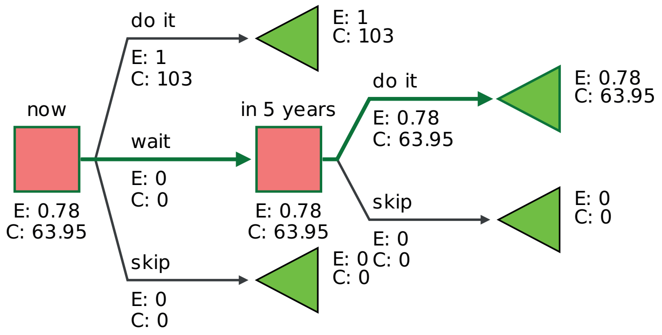


Figure 4.20: When to introduce a given program – a decision tree.

Policy #	Policy	Effect	Cost	Comment
1	wait & skip OR skip	0	0	
2	wait & do	0.784	63.95	ICER=81.62
3	do	1	103	ICER=180.37

Table 4.3: When to introduce a given program – a league table.

inconsistency. The reason for this phenomenon can be traced more easily in CE-plane, see Figure 4.21. Point #1 represents (inter alia) not doing anything (and therefore, it lies in the origin of the CE-plane), and point #3 is an immediate introduction. Delaying the introduction and discounting the delayed consequences represent the delayed introduction towards the origin. If both rates were equal, the point would approach the origin and the segment joining #1 and #3. Then the ICER between #1 and #2 would be equal to the ICER between #2 and #3, so introducing the program now or waiting would be either recommended or not.

Because the cost is discounted stronger, the point for #2 lies below the segment, and the switch from #1 to #2 results in a smaller ICER (still <WTP) than the switch from #2 to #3 (here, >WTP).

The above phenomenon is a variant of the Keeler-Creting paradox and sometimes is given as an argument not to use different discount rates in two-criteria problems.

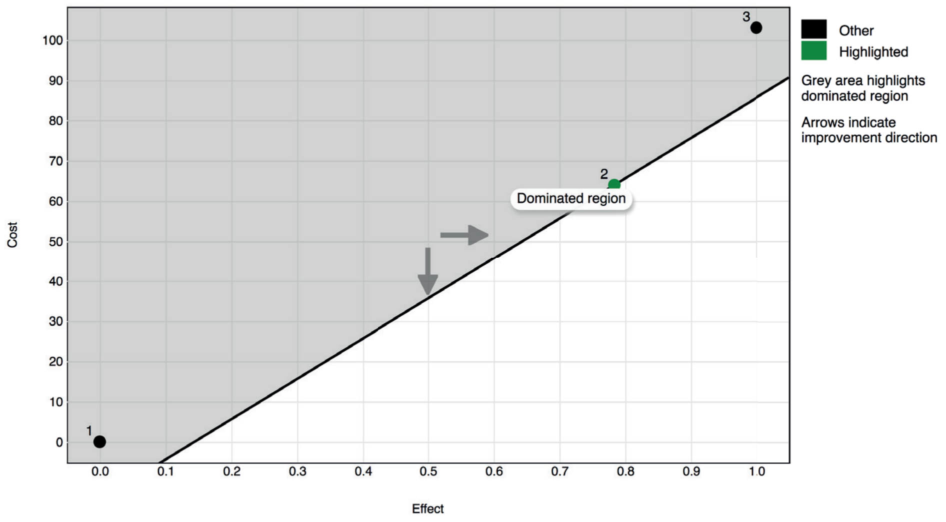


Figure 4.21: When to introduce a given program – a cost-effectiveness plane.

Chapter 5

Case studies

Carmine Spagnuolo
Maria Angela Pellegrino
Alessia Antelmi
Daniel Kaszyński

In the previous chapters, we presented the SilverDecisions software (Chapter 2) and discussed the theoretical background regarding the decision trees (Chapter 3), and more advanced topics like multiple criteria or time-discounting (Chapter 4). This chapter shows five decision cases that could serve the reader as a set of exercises for the previously presented information on the SilverDecisions. The purpose of this chapter is to discuss some practical scenarios that will strengthen the reader's understanding of the decision trees and SilverDecisions.

This chapter is organized as follows. Section 5.1 presents and discusses the case study related to the problem concerning the epidemic control strategies selection; the two scenarios with the corresponding decision trees are proposed. The cases are based on the current Covid-19 pandemic, using simple and more complex decision-making setups. From Section 5.2 onwards, the presented examples are more straightforward applications that are briefly discussed; the remaining models are constructed to give the decision-making context with some prospectus extensions. The reader is welcome to replicate proposed scenarios by themself and come up with a solution.

5.1 Social impact of epidemic control strategies

Preventing and containing contagious diseases not only in the absence of vaccines or drugs, treatments require applying healthcare policies to mitigate the effects of the spread of new viruses or pathogens. These regulation approaches are commonly referred to as Non-Pharmaceutical Interventions (NPIs) [27]. The World Health Organization (WHO) distinguishes three macro-categories of NPIs: i) personal protective measures (PPMs), like using face masks and having regular hand hygiene, ii) environmental measures (EMs), such as surface and object cleaning, and iii) social distancing measures (SDMs), like isolation, quarantine and lockdown measures. Even though their application can help lower the number of infected individuals, each NPI has different effects, resource implications, and ethical considerations [27].

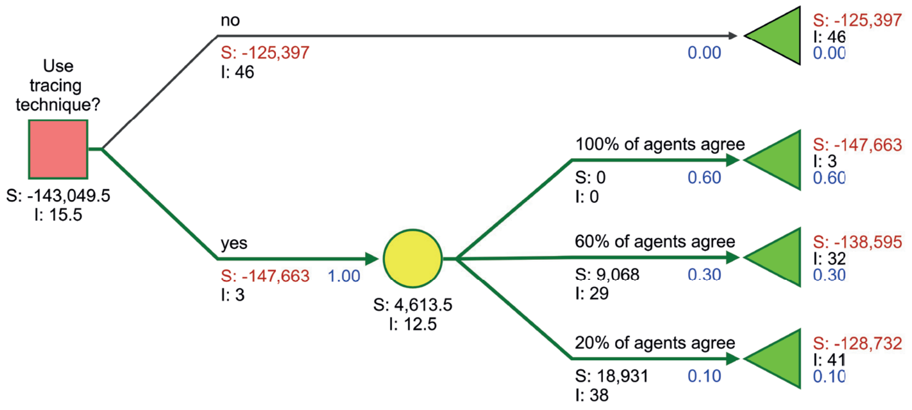


Figure 5.1: Decision tree related to the first case study on whether to invest in a tracing application.

Current development of the recent COVID-19 pandemic highlighted to what extent the increase in human mobility and goods exchange made NPIs, such as lockdown and border closure, more challenging to apply to past cases in history, mainly because of their negative impact on the worldwide economy [11, 3] as well as on the psychological wellness of society [14, 34, 30, 36]. Specifically, each intervention impacts the daily routine of the population both in a direct (for instance, if friends cannot meet in a restaurant) and indirect way (for example, convincing people to wear face masks).

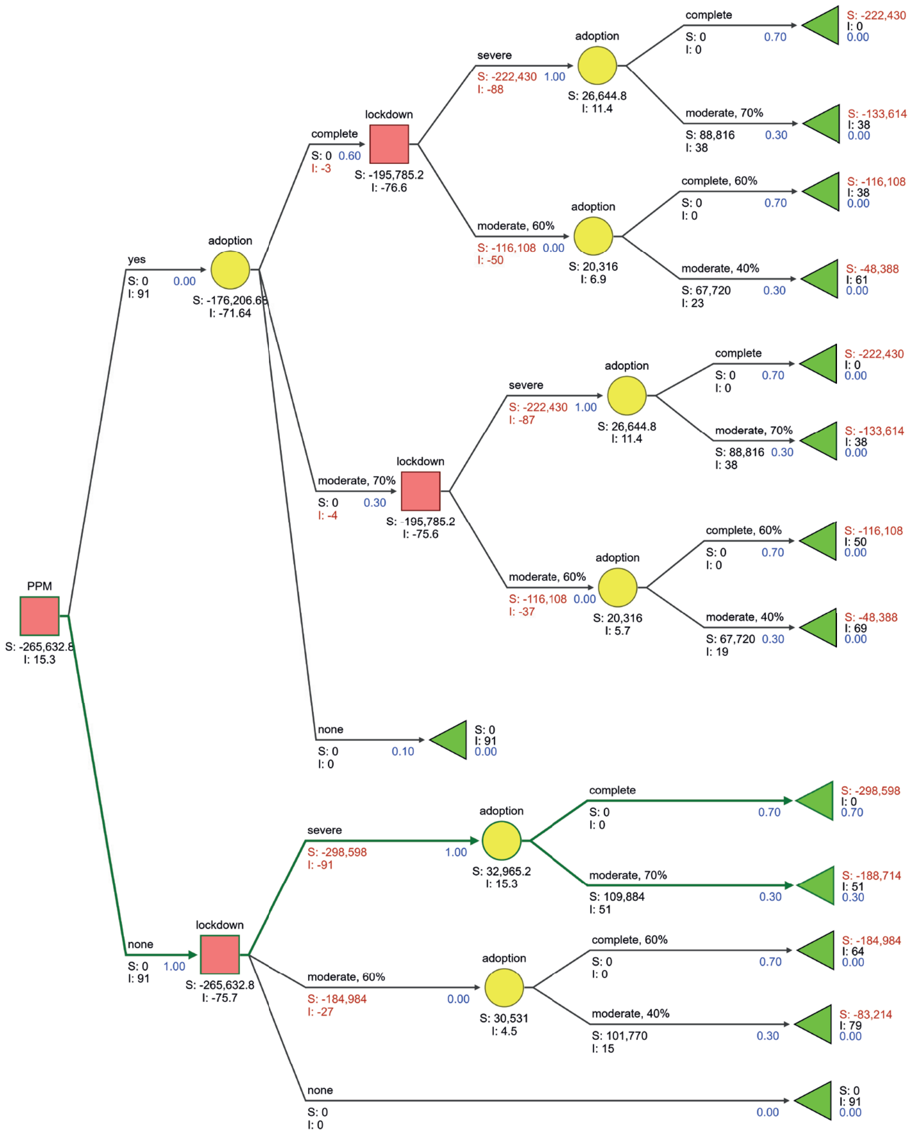


Figure 5.2: PPMs and lockdown use case decision tree.

Choosing the correct control policy to adopt is a burden that governments bear as their decisions have repercussions also when the epidemic is under control. Choosing the best intervention or combination of interventions to apply remains challenging, considering the numerous and closely tied aspects to examine. Further, each NPI intervention may eventually translate into social damage and economic costs. However, estimating the impact of healthcare policies is not trivial. For instance, evaluating how much a given intervention costs in dollars requires knowing production costs (e.g., face masks or sanitization products manufactured), application costs (e.g., closing a specific shop category, like cinema or restaurants), and hospitalization costs (e.g., drug treatments or intensive care units).

This section presents two case studies, discussing two plausible scenarios policymakers may have faced during the last pandemic. Each scenario relates to a specific set of NPIs and analyzes which combination of the chosen interventions apply. Each intervention is evaluated through an agent-based simulation exploiting the model proposed by Antelmi et al. in [1, 2], considering the reduction in the percentage of infected $\mathcal{I}$ and to what extent its application damages the daily life of the population. To overcome the modeling difficulties of considering the economic costs of an NPI intervention, we will use the data-driven approach defined by the same authors. Generally, NPIs and specifically SDMs have a primary objective of reducing contact among people. As a consequence, the impact on society of those interventions translates, in practice, into the number of possible contacts lost. To evaluate each intervention's impact, we determined for each agent the number of agents it was unable to meet because of the adopted NPIs. We then defined the overall social impact of the intervention $\mathcal{S}$ as the sum of the whole population. This definition of social impact represents a domain-agnostic notion of cost and allows us to apply the same methodology to different use cases. In both case studies, we will design a two-criteria decision tree as our final goal is to minimize both percentage of infected and the social impact. The mobility pattern used in the simulation is described by the data set `TownS-1k` introduced by Kim et al. in [21].

5.1.1 Investing in a tracing application

Let us suppose that the government of Wonderland is facing a global pandemic. Currently, 91% of the 1000 agents living in that place are infected. The government decides to apply PPMs, isolation, and quarantine measures. After a week, policymakers should decide whether to invest in a tracing application that lets the agents know if and when they have been in contact with other infected agents.

At this point, the policymakers of Wonderland can evaluate two scenarios. First, they could decide not to invest in the tracing application. This choice translates into having an overall social impact $\mathcal{S}$ equal to $-125,397$ and still having the 46% infected $\mathcal{I}$ after five months from the pandemic's beginning. In the second scenario, the policymakers can suggest investing in the development of the application. However, the benefits of applying the tracing measure are related to the actual percentage of agents using the application. Being a measure imposed by the government, all agents will use the application with high probability. However, policymakers cannot take this behavior for granted. For this reason, they could model with 0.6 probability of this scenario. In this case, the social impact $\mathcal{S}$ paid by the population is $-147,663$, and the epidemic hits zero infected.

The tree models other two plausible scenarios, considering 60% and 20% of the population adopting the application. These scenarios consider that the majority of the agents or only a minor portion of the population agree to adopt the tracing measures. Based on the same rationale, the probability that 60% and 20% of the agents use the application is 0.3 and 0.1, respectively. As stated before, the decreasing probabilities model that policymakers expect a considerable number of agents to adopt the application. When 60% of the agents use the tracing application, the final value of the social impact $\mathcal{S}$ is $-138,595$, while having 32% of infected agents. In the last case, the social impact $\mathcal{S}$ equals $-128,732$, and the final percentage of infected is 41%.

Figure 5.1 shows the decision tree. In this case, the best solution is investing in the tracing application, regardless of the final number of actual users.

5.1.2 The impact of lockdown policies

Let us suppose that the government of Wonderland wants to evaluate whether applying only PPMs or lockdown measures, or a combination of those to better face the pandemic.

The policymakers of Wonderland can evaluate different scenarios, which are described and summarized in Table 5.1. For each scenario, the table lists: *(i)* a brief description of the scenario, *(ii)* which policy has been applied and to what extent (in terms of the percentage of citizens using PPMs and locations to close), *(iii)* the final value of the social impact $\mathcal{S}$, and *(iv)* the final number of infected $\mathcal{I}$.

The values $\mathcal{S}$ and $\mathcal{I}$ let policymakers evaluate different aspects as reported in the decision tree graphically represented in Figure 5.2. As in the previous decision tree, each decision node represents whether the government should apply a specific intervention and how strict the healthcare policy should be. Each chance node models policymakers' uncertainty about how many people will agree to adopt the intervention. Hence, policymakers can consider a scenario in which all agents adopt the measure and a scenario in which only part of the population adopts it. The decision applied as government regulation can hypothesize that all agents will respect the

Table 5.1: PPMs and lockdown case study parameters and final values for the social impact $\mathcal{S}$ and the percentage of infected $\mathcal{I}$.

Description	**NPIs applied**		$\mathcal{S}$	$\mathcal{I}$
	PPMs	**Lockdown**		
In these scenarios, everyone	100%	100%	−222,430	0
uses PPMs, while the percent-age	100%	70%	−133,614	38
of locations to close	100%	60%	−116,108	38
ranges from 100% to 40%.	100%	40%	−48,388	61
In these scenarios, only 70% of	70%	100%	−222,430	0
the agents use PPMs, while the	70%	70%	−133,614	38
percentage of locations to close	70%	60%	−116,108	50
spans from 100% to 40%.	70%	40%	−48,388	69
In these scenarios, no one uses	0%	100%	−298,598	0
PPMs, while the percentage	0%	70%	−188,714	51
of locations to close	0%	60%	−184,984	64
ranges from 100% to 40%.	0%	40%	−83,214	79
No intervention.	0%	0%	0	91

intervention with a higher probability. The overall social impact and the final percentage of infected are based on this consideration. When not all agents behave according to the measures, the final social impact evaluated decreases with respect to the baseline as some agents still will be free to move and meet other agents in the simulation. As a consequence, if, on the one hand, policymakers may have back some of the estimated social impacts, on the other hand, they will observe the final number of infected increasing.

In this case, the best solution is to apply severe lockdown policies to reduce to zero the percentage of infected, even though paying the highest social impact.

5.2 Banana trees

The second case study concerns a decision problem undertaken by a banana grower. The decision-maker in this scenario is a Brazilian farmer who established a banana plantation. For many years, the grower followed one proven strategy: exporting all of the crops. This strategy has been generating \$ 10,000 in net profit. A year ago, to increase his income, the grower decided to sell all the obtained crops to an external company that produces banana nectars. Considering the obtained income and operating costs and greater market recognition, the grower began to consider changing the commercialization strategy for the results of his work.

This year, the farmer plans to maximize the profit obtained from his crops by choosing the most convenient harvest time. The grower can take one of several strategies. The first strategy assumes that the still unripe fruit will be harvested and exported to European countries – during long sea transport, the fruit will ripen. This is a safe decision option as all bananas will be sold, and the farmer will not suffer any losses from spoiled fruits. If this decision is chosen, the grower will get the lowest profit from the sale (\$ 10,000, the baseline profit level in this case study).

The grower can also harvest moderately ripe fruit, but there is a 10% chance that a quarter of the fruit will rot. The rest of the fruit will be exported to nearby countries for nectars. Since the company purchasing fruit for nectars has problems with production this year, there is a 30% chance that it will not buy any fruit from the farmer (if it does, the profit increases by 20%, up to \$ 12,000). There is also a 10% chance that a competing company from a neighbouring country that purchases fruit will propose a higher purchase price, but only for a large amount of fruit, i.e., if nothing rots (in this case, the profit increases by 60%).

The grower is also considering buying a nectar-making machine – the farmer could export produced nectar, obtaining a higher income than he would get for selling the fruit to the factory (the profit increases by as much as 80% compared to the baseline income \$18,000).

The grower could also wait almost until the end of the season and designate the entire harvest for drying. Dried bananas are the most expensive form a grower can sell (profit increases by 50% compared to the baseline price), but there are two obstacles. Only four drying machines can be rented in the area where the grower lives. There is a 40% chance that there will be no rental machine left, and the grower will have to sell bananas for little money to a nectar-producing company (the low price is caused by the fact that very ripe bananas are only an optional nectar addition, profit drops by 50%, $5,000). The second problem is that there is a 20% chance that a third of the fruit will be spoiled and not survive the harvest.

The description of the problem may seem confusing and not entirely clear after reading it once. In reality, however, the number of possible economic decisions is almost always extensive, and the variants of the outcomes may often be very differentiated (as in the presented example – both selling fruit for export or renting fruit drying machines). However, this case study aims to show the translation of practical decision-making problems into the decision trees using the `SilverDecisions`. Below we present only a part of the decision tree; see Figure 5.3.

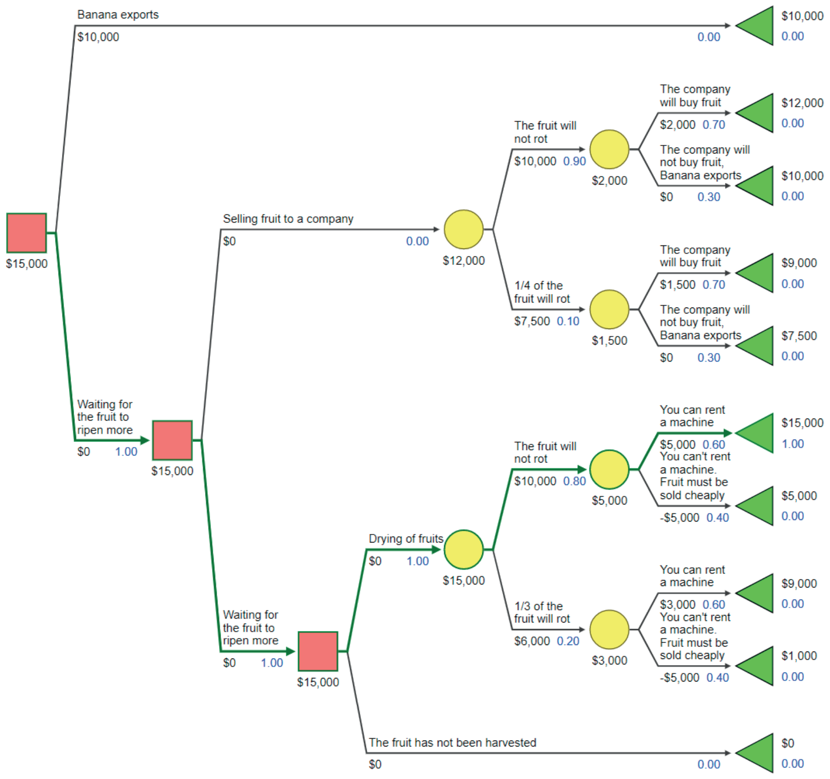


Figure 5.3: Banana trees – unfinished decision tree

In this task, we ask the reader to analyze the description of the decision task and complete the missing part of the tree. When refilling, it is worth paying attention to how the path indicating the optimal solution changes, i.e., the one that will bring the greatest profit. Additionally, we encourage you to perform a sensitivity analysis before and after completing the tree on the *standard_price* parameter. We propose the following parameter values:

min = 8,000, max = 12,000, step = 10,000.

With the decision tree indicated above, the best solution is to rent a banana drying machine. Note that for the full decision tree (filled with the missing branches), the export nectar strategy will be the most profitable option. Sample solution; see Figure 5.4.

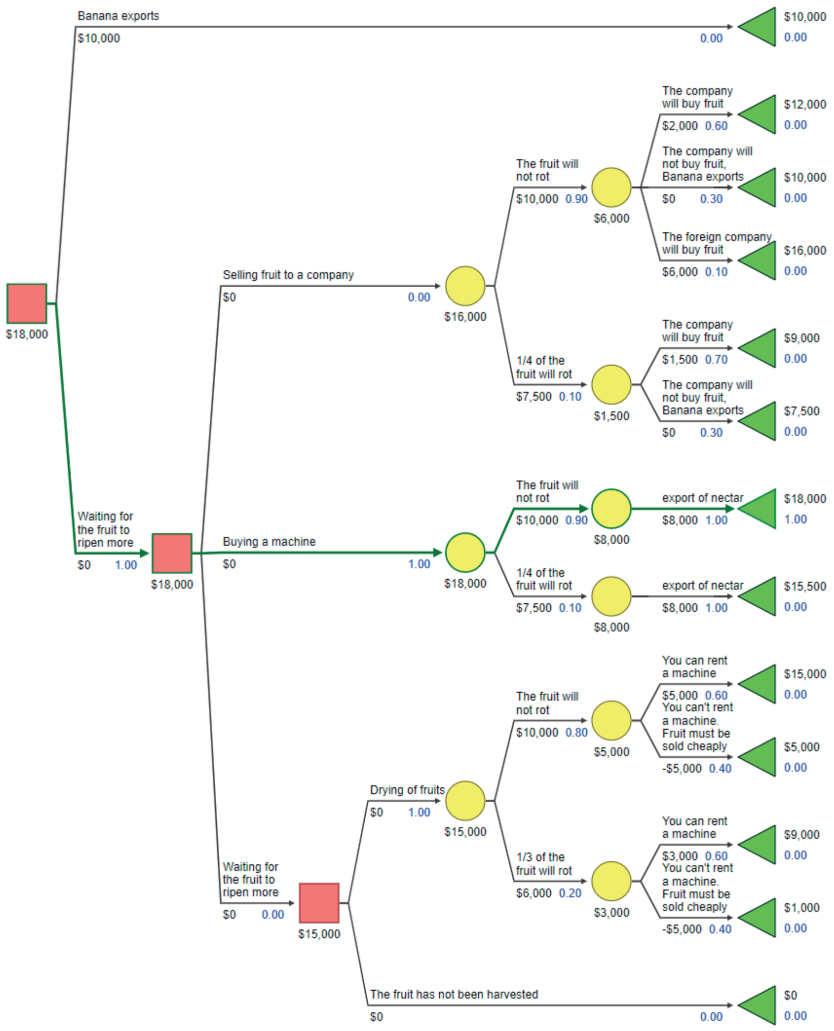


Figure 5.4: Banana trees – full solution of the decision tree

5.3 Machine rental

In the next case study, the company that produces decorated pencils for sketching, which rents production machines from the beginning, will be considered. Unfortunately, the company is struggling with considerable financial problems, and if at the end of the month the company's balance is below $35,000, it will have to file for bankruptcy. Currently, the company has a balance equal to $50,000, and the management board has to decide whether to change the company's current model. Currently, the company rents machines, but the costs are not fixed (e.g., depending on exchange rates): in the optimistic scenario (30% chance), the fee is equal to $8,000. In the pessimistic scenario, it equals $12,000. Based on the costs, the solution may be to buy a machine worth $22,000 (the price is already reduced by the possible costs of integrating the new equipment into the production line structure). Configuration of the device after purchase is a one-time operation. Configuration correctness levels can be divided into three groups. Correct configuration (40% chance) will bring $20,000 revenue. The average configuration and the poor one will bring $15,000 and $5,000, respectively, equal probability.

Task: Enter the values given in the description on the initial tree diagram. See Figure 5.5.

In this case, the decision tree will maximize the profits achieved – the solution with the lowest level of loss will be chosen. See Figure 5.6.

5.4 Transport company

In this case study, we show an example of a transport company that plans to introduce a new system for optimizing drivers' routes. This decision is due to the constant increases in fuel and car parts prices. The introduction of the new system will ensure better routing of delivery vehicles, translating into greater efficiency. The first implementation stage will be to change the current environment in which all the company's systems are placed: it costs $50,000. If the company does not decide to introduce changes, it will lose an average of $15,000 a month due to additional fuel and car repairs. In addition, if the change in the technological environment is unsuccessful (15% chance), the company will spend $10,000 restoring old copies of the software.

Otherwise, it will be possible to implement a new optimization system that costs $20,000. A properly functioning system will bring a profit of $200,000 because the company will be able to provide more services at

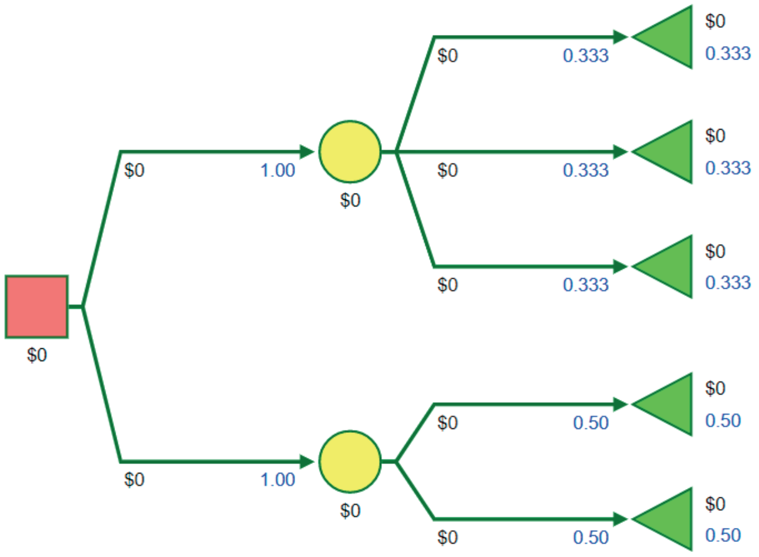


Figure 5.5: Machine rental – blank decision tree

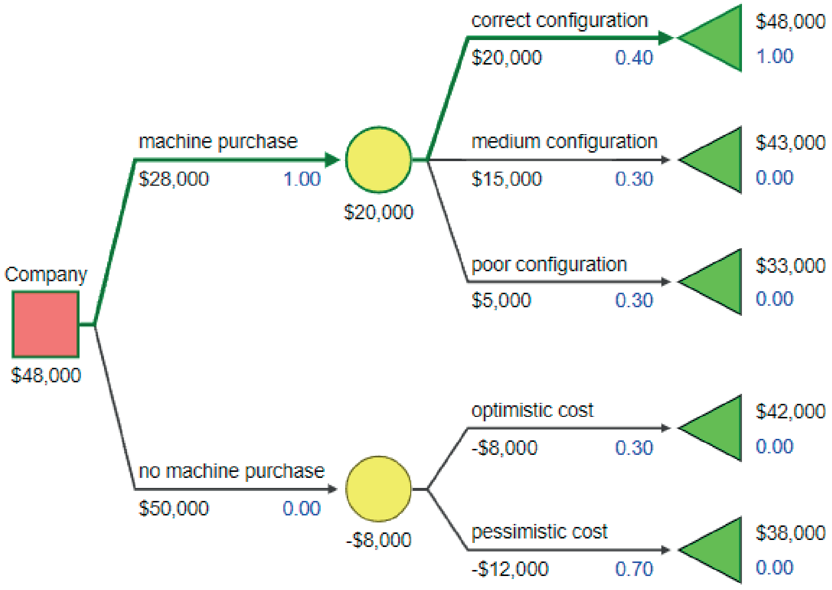


Figure 5.6: Machine rental – full decision tree

lower costs of their implementation. An incorrect operating system (20% chance) will generate the previously mentioned $10,000 costs to restore the old environment.

Prepare an alternative solution in case of a malfunctioning system. Suggest a branch that implements the system repair scenario. For a sample solution, see Figure 5.8.

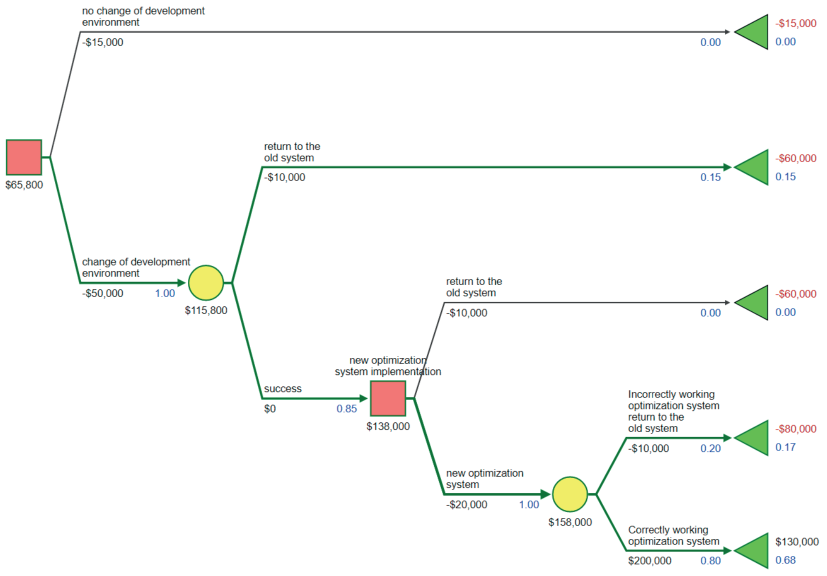


Figure 5.7: Transport company – unfinished decision tree

5.5 Sale of cosmetics

In the last case study, a facial care company has a chance to launch and sell a luxury product or a popular product. For each decision option, the probabilities of good, average and mediocre sales and the financial effects of these outcomes were determined based on forecasts and statistical data analyses.

For a luxury product, the probability of a good sale (with revenue of $12,000) is 40%, of an average sale (with revenue of $65,000) – 30%, and of a mediocre sale (with revenue of $12,000) – 30%. Similarly, for a popular product, the probability of good sales is 50% (revenue of $105,000), of average sales – 40% (revenue of $55,000) and mediocre sales – 10% (revenue of only $20,000).

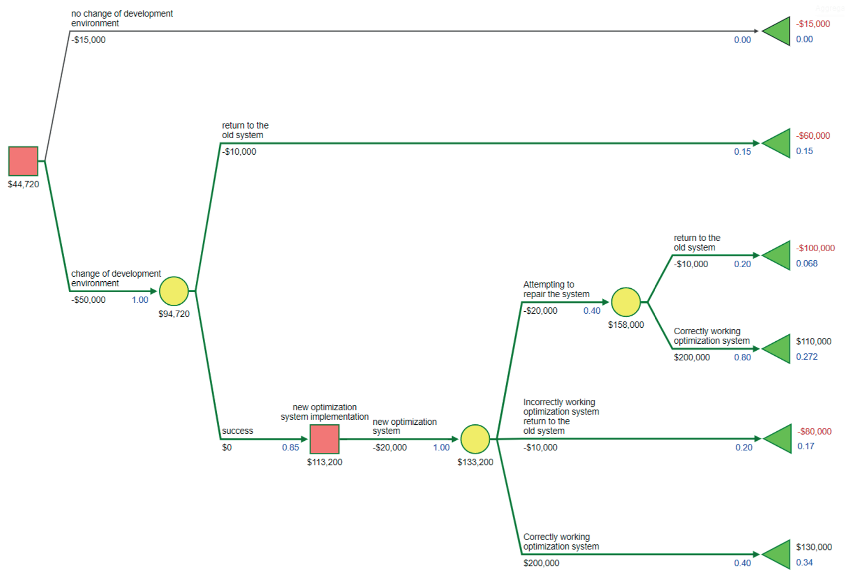


Figure 5.8: Transport company – extended decision tree

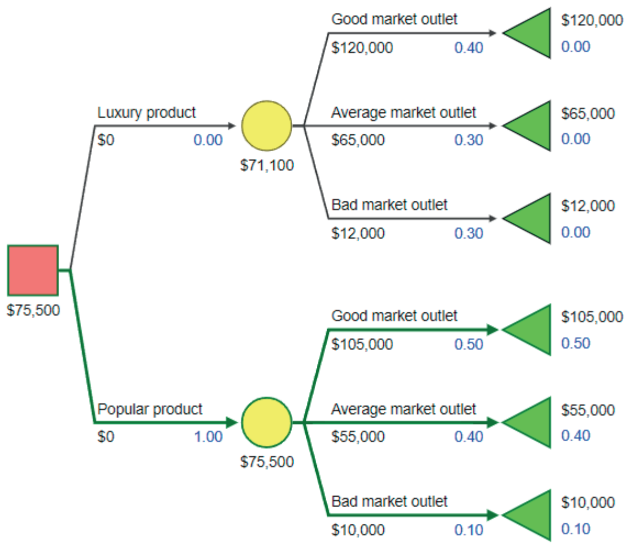


Figure 5.9: Sale of cosmetics – full decision tree

The goal is to build a tree and evaluate which decision options are more profitable for the company. Then suggest other probabilities that make it more beneficial to introduce a luxury products. For a sample solution, see Figure 5.9.

Chapter 6

Concluding remarks

Bogumił Kamiński
Michał Jakubczyk
Przemysław Szufel

In this book, we have presented the theory and applications of decision trees as a tool that can be used for modeling sequential decision-making problems. We have presented, in detail, how one can build decision trees using the `SilverDecisions` software. In particular, we discussed advanced topics of working with functions and variables, which are important when a user builds a large decision tree and wants to perform its sensitivity analysis. Next, we have given rigorous mathematical foundations of the decision tree model and presented all statistical concepts that the user needs to know to perform and interpret sensitivity analysis of decision trees correctly. In what follows, important practical topics of working with decision trees are discussed: computation of value of information, solving multiple-criteria problems, and various options how discounting can be taken into account when working with decision trees. The book concludes with three example case studies meant to show users how `SilverDecisions` can be used to solve practical problems. The book contains two appendices that can serve users as reference material when using `SilverDecisions` software.

The `SilverDecisions` software is available for free using a browser at http://silverdecisions.pl/ website. In the past, the software has been used to support decision-making in many application areas. Sequential decision-making is naturally encountered in managerial problems involving, e.g., the valuation of real options. Similarly, `SilverDecisions` users have reportedly found it applicable in modeling complex legal cases or processes in engineering. The features related to modeling multiple-criteria decisions are often used to model problems from medicine (especially for health technology assessment) and environmental economics. An interested user can find additional example decision trees in the Gallery section of its documentation (https://github.com/SilverDecisions/SilverDecisions/wiki/Gallery).

We hope that using this book; you will be able to analyze complex sequential decision problems with uncertainty using the decision tree approach. We have described the software and mathematical foundations of decision trees in detail as well and have provided several examples of practical applications.

`SilverDecisions` has been developed with the JavaScript language and runs with the most modern web browser. This approach makes it possible to integrate `SilverDecisions` with other software (e.g., via embedding or linking). In this book, we have also shown that some features of the JavaScript interface are exposed to users of the software. This allows it to extend the presentation/reporting possibilities of decision trees created with this product.

Since `SilverDecisions` is developed as open-source software, interested users can get involved in its development. You can find a guide on how to get started in the developer's section of the manual (https://github.com/SilverDecisions/SilverDecisions/wiki/Developer's-guide).

Appendix A

Random distributions available in SilverDecisions

Przemysław Szufel

Below you will find the complete list of random functions available for use in `SilverDecisions` along with their parameters. The functions presented in this list can be used in variable definitions as well as directly for pay-off and probability values. Please note that every time you press the '↻ *Recompute*' button those functions return different values and your tree is updated.

The random functions are also the foundation tool for probabilistic sensitivity analysis – see Section 3.3.

- `random()` – returns an uniformly distributed random value in the range $[0, 1)$.
  This is a standard math.js function that can be also used in `SilverDecisions`. This function takes no parameters.
- `Uniform(a,b)` – returns an uniformly distributed random value
  - `a` – lower bound
  - `b` – upper bound, $b >= a$
- `Exponential(beta)` – returns an exponentially distributed random value
  - `beta` – shape parameter, $\beta > 0$, $\lambda = 1/\beta$.
    For exponential distribution β is equal to the mean.
- `Normal(mean,std)` – returns a normally distributed random value
  - `mean` – mean value
  - `std` – standard deviation, $\sigma >= 0$

- `Pareto(alpha, minimum)` – returns a Pareto distributed random value
  - `alpha` shape parameter, $\alpha > 0$
  - `minimum` scale parameter, $minimum > 0$
- `Loglogistic(alpha,beta)` – returns a Log-logistic distributed random value
  - `alpha` shape parameter, $\alpha > 0$
  - `beta` scale parameter, $\beta > 0$
- `Weibull(alpha,beta)` – returns a Weibull distributed random value
  - `alpha` shape parameter, $\alpha > 0$
  - `beta` scale parameter, $\beta > 0$
- `Erlang(k,beta)` – returns an Erlang distributed random value
  - k shape parameter, k = 1,2,3,...
  - `beta` scale parameter, $\beta > 0$ $\beta = 1/\lambda$. For Erlang distribution β is equal to the mean value.
- `Triangular(a,b,m)` – returns a Triangular distributed random value
  - `a` left side
  - `b` right side, $b >= a$
  - `m` triangle peak (mode), $a <= m <= b$
- `Trapezoidal(a,b,c,d)` – returns a Trapezoidal distributed random value
  - `a` left side
  - `b` right side, `b >= a`
  - `c` left trapezoid peak (mode)
  - `d` right trapezoid peak (mode), $a <= c <= d <= b$
- `LogNormal(mean,std)` – returns a Log-normally distributed random value
  - `mean` – mean value
  - `std` – standard deviation, $\sigma >= 0$
- `Bernoulli(p)` – returns a Bernoulli distributed random value (0 or 1)
  - `p` success probability
- `Binomial(n,p)` – returns a Binomial distributed random value $(0, 1, ..., n)$
  - `n` number of trials
  - `p` success probability in each trial
- `Geometric(p)` – returns any Geometric distributed random value
  - `p` success probability
- `Poisson(lambda)` – returns a Poisson distributed random value
  - `lambda` mean value

- `Gamma(alpha,beta)` – returns a Gamma distributed random value
  - `alpha` shape parameter, $\alpha > 0$
  - `beta` rate parameter, $\beta > 0$

Appendix B

SilverDecisions quick user reference

Przemysław Szufel

In this section, we present a comprehensive list of basic SilverDecisions functionalities. We start by presenting a list of mouse and keyboard shortcuts supported by the application. Next, we present an overview of the options that are displayed on the left side panel of the application. Finally, we discuss the buttons available at the menu bar.

Mouse actions

left-click	node/edge selection
left-click	context menu (adding/manipulating nodes)
double-click	context menu (adding/manipulating nodes)

Keyboard

Del	delete selected nodes
Ctrl-C	copy selected nodes
Ctrl-X	cut selected nodes
Ctrl-V	paste copied nodes as a subtree of a selected node
Ctrl-Z	undo (revoke the last tree modification)
Ctrl-Y	redo (revoke the undo operation)
Ctrl-Alt-D	add new Decision subnode of a selected node
Ctrl-Alt-C	add new Chance subnode of a selected node
Ctrl-Alt-T	add new Terminal subnode of a selected node

⌨ Ctrl-Alt-D	inject new Decision node into a selected edge
⌨ Ctrl-Alt-C	inject new Chance node into a selected edge

Note that when you click the "About" icon in the top right corner, you can get the basic information on your SilverDecision application, including a list of important keyboard shortcuts.

Actions in the left panel

Layout

Horizontal margin	set margin from the left of the canvas
Vertical margin	set margin from the top of the canvas
Node size	set the node scaling
Edge slant (max)	set the maximum slant for plotting the sloping part of the edge
Width	horizontal separation of tree node (shown only for *'TREE'* or *'CLUSTER'* layout)
Height	vertical separation of tree nodes (shown only for *'TREE'* or *'CLUSTER'* layout)

Details

Title	title of the diagram the title can consist of several line (you can use ⌨ Enter)
Description	description for the diagram it is displayed right under the title in a smaller font size the description can consist of several line (you can use ⌨ Enter)

Decision Node

This action will be shown in the left panel when a decision node is selected.

Label	name of the decision node (name can span multiple lines)
Connections	labels and payoffs of specified edges (shown only if the decision node has sub-nodes)

Chance Node

This action will be shown in the left panel when a chance node is selected.

Label	type a name of the event (span multiple lines by pressing enter)
Connections	labels, payoffs and probabilities of specified edges (shown only if the chance node has sub-nodes)

Terminal Node

This action will be shown in the left panel when a terminal node is selected.

Label	type the endpoint name (span multiple lines by pressing enter)

Edge

This action will be shown in the left panel when an edge within a decision tree is selected.

Label	provide a decision name or an outcome label (span multiple lines by pressing ⌨ Enter)
Payoff	type the value of the specified action/outcome
Probability	type the probability of the specified outcome (shown only for chance nodes)

Note that arithmetic expressions are allowed as probability and payoff inputs. The `#` parameter is a defaulted input for probabilities. For more information, please read the description below.

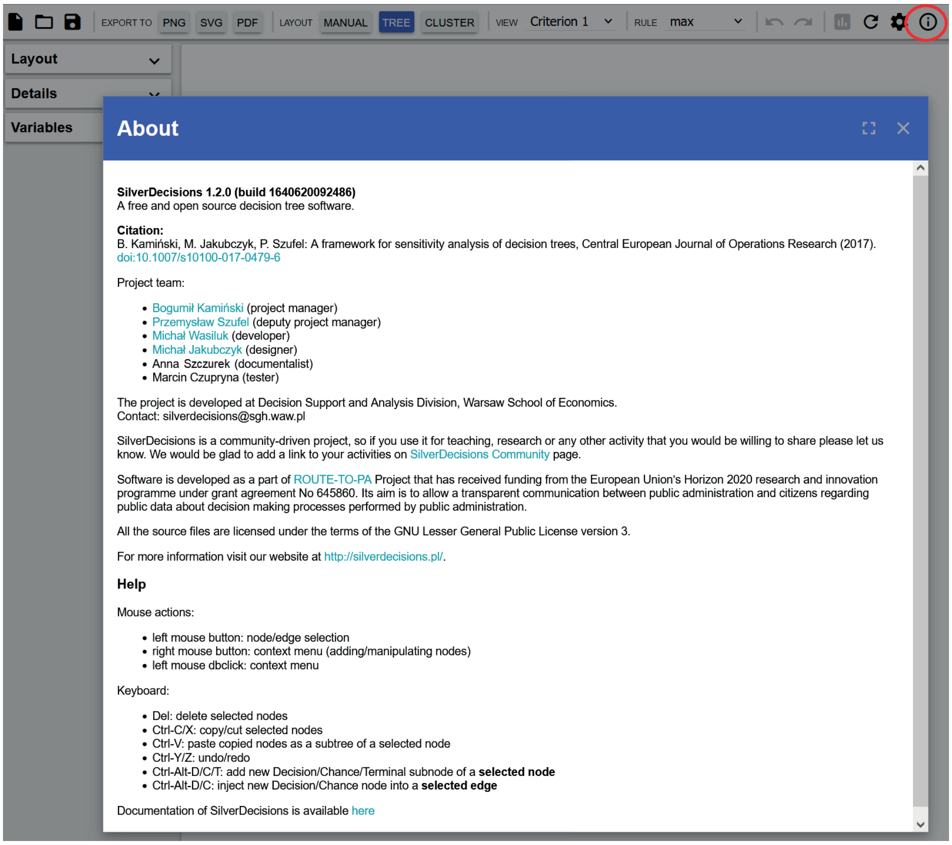
Floating text

This action will be shown in the left panel when a floating text is selected.

Text	any text can be entered into this dialog. Note that text can be made to span multiple lines by pressing ⌨ Enter

Toolbar actions

New diagram	clears canvas and starts an empty new diagram
Open existing diagram	loads existing diagram from disk
Save current diagram	saves current diagram to disk in JSON format
PNG	saves current diagram in PNG format
SVG	saves current diagram in SVG format
PDF	saves current diagram in PDF format
MANUAL	allow manual positioning of tree nodes
TREE	default, automatic layout of tree nodes aligned to the left
CLUSTER	automatic layout of tree nodes aligned to the right
RULE	decision-making criterion can be selected from the following options: `max` – expected value maximization rule; `maxi-min` – rule based on the worst-case scenario, pessimistic approach: in each chance node the lowest payoff is chosen; `maxi-max` – rule based on the best possible scenario, optimistic approach: in each chance node the highest payoff is chosen; `min` – expected value minimization rule; `mini-max` – rule based on the worst-case scenario for cost type of payoffs, pessimistic approach: in each chance node the highest payoff is chosen; `mini-min` – rule based on the best possible scenario for cost type of payoffs, optimistic approach: in each chance node the lowest payoff is chosen.
Undo	undo last action
Redo	redo action
Settings	open diagram settings dialog box
About	open dialog box containing concise information about SilverDecisions



About

SilverDecisions 1.2.0 (build 1640620092486)
A free and open source decision tree software.

Citation:
B. Kamiński, M. Jakubczyk, P. Szufel: A framework for sensitivity analysis of decision trees, Central European Journal of Operations Research (2017).
doi:10.1007/s10100-017-0479-6

Project team:

- Bogumił Kamiński (project manager)
- Przemysław Szufel (deputy project manager)
- Michał Wasiluk (developer)
- Michał Jakubczyk (designer)
- Anna Szczurek (documentalist)
- Marcin Czupryna (tester)

The project is developed at Decision Support and Analysis Division, Warsaw School of Economics.
Contact: silverdecisions@sgh.waw.pl

SilverDecisions is a community-driven project, so if you use it for teaching, research or any other activity that you would be willing to share please let us know. We would be glad to add a link to your activities on SilverDecisions Community page.

Software is developed as a part of ROUTE-TO-PA Project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645860. Its aim is to allow a transparent communication between public administration and citizens regarding public data about decision making processes performed by public administration.

All the source files are licensed under the terms of the GNU Lesser General Public License version 3.

For more information visit our website at http://silverdecisions.pl/.

Help

Mouse actions:

- left mouse button: node/edge selection
- right mouse button: context menu (adding/manipulating nodes)
- left mouse dbclick: context menu

Keyboard:

- Del: delete selected nodes
- Ctrl-C/X: copy/cut selected nodes
- Ctrl-V: paste copied nodes as a subtree of a selected node
- Ctrl-Y/Z: undo/redo
- Ctrl-Alt-D/C/T: add new Decision/Chance/Terminal subnode of a **selected node**
- Ctrl-Alt-D/C: inject new Decision/Chance node into a **selected edge**

Documentation of SilverDecisions is available here

Figure B.1: *'About'* info contains basic information on keyboard shortcuts.

Bibliography

[1] A. Antelmi, G. Cordasco, V. Scarano, and C. Spagnuolo. Modeling and evaluating epidemic control strategies with high-order temporal networks. *IEEE Access*, pages 1–1, 2021.

[2] A. Antelmi, G. Cordasco, C. Spagnuolo, and V. Scarano. A design-methodology for epidemic dynamics via time-varying hypergraphs. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 61–69, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems.

[3] A. Atalan. Is the lockdown important to prevent the covid-19 pandemic? effects on psychology, environment and economy-perspective. *Annals of Medicine and Surgery*, 56:38 – 42, 2020.

[4] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[5] G. Boriani, C. Kennergren, K. Tarakji, D. Wright, F. Ahmed, J. McComb, A. Goette, T. Blum, M. Biffi, M. Green, J. Shore, P. Carion, and B. Wilkoff. Cost-effectiveness analyses of an absorbable antibacterial envelope for use in patients at increased risk of cardiac implantable electronic device infection in germany, italy, and england. *Value in Health*, 24(7):930–938, 2021.

[6] J. Busemeyer. Dynamic decision making. In N. J. Smelser and P. B. Baltes, editors, *International Encyclopedia of the Social & Behavioral Sciences*, pages 3903–3908. Pergamon, Oxford, 2001.

[7] A. C. Davison and D. V. Hinkley. *Bootstrap Methods and their Application.* Cambridge University Press, 1997.

[8] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester. *A Modern Introduction to Probability and Statistics.* Springer, 2005.

[9] A. Diederich. Sequential decision making. In N. J. Smelser and P. B. Baltes, editors, *International Encyclopedia of the Social & Behavioral Sciences*, pages 13917–13922. Pergamon, Oxford, 2001.

[10] R. Edlin, C. McCabe, C. Hulme, P. Hall, and J. Wright. *Cost Effectiveness Modelling for Health Technology Assessment. A Practical Course.* Adis, 2015.

[11] H. Fang, L. Wang, and Y. Yang. Human mobility restrictions and the spread of the novel coronavirus (2019-ncov) in china. Working Paper 26906, National Bureau of Economic Research, 2020.

[12] M. Gold, J. Siegel, L. Russell, and M. Weinstein. *Cost-Effectiveness in Health and Medicine.* Oxford University Press, 1996.

[13] P. Goodwin and G. Wright. *Decision Analysis for Management Judgment.* John Wiley & Sons Ltd, 4 edition, 2009.

[14] M. R. Gualano, G. Lo Moro, G. Voglino, F. Bert, and R. Siliquini. Effects of covid-19 lockdown on mental health and sleep disturbances in italy. *International Journal of Environmental Research and Public Health*, 17(13):4779, 2020.

[15] R. Hespos and P. Strassmann. Stochastic decision trees for the analysis of investment decisions. *Management Science*, 11(10):B244–B259, 1965.

[16] B. Illowsky and S. Dean. *Introductory Statistics.* OpenStax, 2013.

[17] J. John, F. Koerber, and M. Schad. Differential discounting in the economic evaluation of healthcare programs. *Cost Effectiveness and Resource Allocation*, 17, 2019.

[18] B. Kamiński, M. Jakubczyk, and P. Szufel. A framework for sensitivity analysis of decision trees. *Central European Journal of Operations Research*, 26:135–159, 2018.

[19] B. Kamiński and JuliaData contributors. *DataFrames.jl: In-memory tabular data in Julia*, 2021.

[20] B. Kamiński, P. Prałat, and F. Théberge. *Mining Complex Networks.* CRC Press, 2022.

[21] J.-S. Kim, H. Jin, H. Kavak, O. C. Rouly, A. Crooks, D. Pfoser, C. Wenk, and A. Züfle. Location-based social network data generation based on patterns of life. In *2020 21st IEEE International Conference on Mobile Data Management (MDM)*, pages 158–167, 2020.

[22] D. P. Kroese, T. Taimre, and Z. I. Botev. *Handbook of Monte Carlo Methods*. Wiley, 2011.

[23] A. M. Law, W. D. Kelton, and W. D. Kelton. *Simulation modeling and analysis*, volume 3. Mcgraw-hill New York, 2007.

[24] P. Lázaro, L. de Isla, G. Watts, R. Alonso, R. Norman, O. Muñiz, F. Fuentes, N. Mata, J. López-Miranda, J. González-Juanatey, J. Díaz-Díaz, A. Blasco, and P. Mata. Cost-effectiveness of a cascade screening program for the early detection of familial hypercholesterolemia. *Journal of clinical lipidology*, 11(1):260–271, 2017.

[25] J. Magee. Decision trees for decision making. harvard business review, 1964.

[26] T. O'Donoghue and M. Rabin. Doing it now or later. *The American Economic Review*, 89:103–124, 1999.

[27] W. W. H. Organization. Non-pharmaceutical public health measures for mitigating the risk and impact of epidemic and pandemic influenza, 2019.

[28] J. Quinn and JuliaData contributors. *CSV.jl: Utility Library for Working with CSV and Other Delimited Files in the Julia Programming Language*, 2021.

[29] V. Retèl, D. Byng, S. Linn, K. Jóźwiak, H. Koffijberg, E. Rutgers, F. Cardoso, M. Piccart, C. Poncet, L. Veer, and W. van Harten. Cost-effectiveness analysis of the 70-gene signature compared with clinical assessment in breast cancer based on a randomised controlled trial. *European Journal of Cancer*, 137:193–203, 2020.

[30] R. Rossi, V. Socci, D. Talevi, S. Mensi, C. Niolu, F. Pacitti, A. Di Marco, A. Rossi, A. Siracusano, and G. Di Lorenzo. Covid-19 pandemic and lockdown measures impact on mental health among the general population in italy. *Frontiers in Psychiatry*, 11:790, 2020.

[31] W. F. Samuelson and S. G. Marks. *Managerial Economics*. John Wiley & Sons, Inc., 5 edition, 2006.

[32] A. N. Shiryaev. *Probability.* Springer, 1996.

[33] N. T. Thomopoulos. *Essentials of Monte Carlo Simulation.* Springer, 2013.

[34] C. Wang, R. Pan, X. Wan, Y. Tan, L. Xu, R. S. McIntyre, F. N. Choo, B. Tran, R. Ho, V. K. Sharma, and C. Ho. A longitudinal study on the mental health of general population during the covid-19 epidemic in china. *Brain, Behavior, and Immunity*, 87:40 – 48, 2020.

[35] D. Waters. *Quantitative Methods for Business.* Financial Times/ Prentice Hall, 5 edition, 2011.

[36] L. Webb. Covid-19 lockdown: A perfect storm for older people's mental health. *Journal of Psychiatric and Mental Health Nursing*, 2020.

[37] D. B. West. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[38] H. P. Williams. *Model building in mathematical programming.* John Wiley & Sons, 2013.