From theORy to application

learning to optimize with Operations Research in an interactive way

Alessandro Bombelli, Bilge Atasoy, Stefano Fazi, Doris Boschma



TU Delft OPEN

Contents

From theORy to application: learning to optimize with Operations Research in an interactive way 8
Contents 18
Introduction 30
Introduction to or 32
What is or and some historical insights 32
Preliminary insights into mathematical modeling 33
Introduction to serious games and gamification 36
Serious games 36
Gamification 36
An overview of serious games in or education 37
Serious games in this book: setup and learning objectives 38
Fundamentals 40
A primer on linear algebra 42
Vectors 42
Matrices 44
Linear systems 45
Introduction to mathematical modeling 48
Sets 48
Parameters 52
Decision variables 52
Objective function 53
Constraints 55
General form of a mathematical model 56
Construction of a mathematical model 57
Special types of constraints 62

Contents (continued)

Big-M notation	62
Either-or constraints	63
K-out-of-N constraints	66
Fixed charge constraints	69
Final remarks	73
Linearization techniques	74
Product of decision variables	74
Product of two binary variables	75
Product of a binary and a continuous decision variable	77
Absolute value	78
Piecewise linear formulations	80
If-else statement	82
Solution Methods	86
The simplex method	88
Graphical representation of an lp and corner points	88
Augmented form of an lp	96
Inequality constraints in the form	97
Equality constraints	99
Inequality constraints in the form	100
Final remarks	102
The simplex method: description of the algorithm	102
Basic and non-basic variables	102
Basic solutions	104
The simplex tableau	106
The simplex algorithm: how to iterate and when to stop	109

Contents (continued)

Examples 119
Additional considerations 129
bb 132
Motivation for bb 132
Problem types 133
The basics of bb 133
Linear relaxation, root node, and tree structure 134
Best bound, best incumbent, and gap optimality 136
A note on functional constraints 139
Fathoming rules 140
Branching, bounding, and separation rules 140
Branching options 141
Bounding and separation rules 143
The bb algorithm in a nutshell 147
Considerations on the algorithmic complexity of the bb algorithm 149
An illustrative example 149
bc 162
Motivation for bc 162
Examples of cutting planes 165
Gomory fractional cuts 165
Cover inequalities 170
Zero-half cuts 174
List of other cutting planes 174
Combining bb and cutting planes for an efficient bc 175
Assignment Problems 176

Contents (continued)

Assignment and scheduling problems 178
Assignment problems 178
The Hungarian algorithm 180
Preliminaries of scheduling 187
smsp 188
pmsp 192
p-median problem 195
Facility location problem 198
Packing problems 202
kps 202
0-1 kp 202
Bounded kp 206
0-1 multiple kp 207
Other variants of the kp 208
bpps 208
One-dimensional bpp 208
Two-dimensional horizontal bpp 210
Other variants of the bpp 217
Networks 220
An introduction to graph theory 222
Definition of a graph 222
Properties of a graph 224
From ``abstract'' graphs to ``concrete'' networks 232
Network problems 238
tp 238

Contents (continued)

General Setting 238
tp: lp mathematical formulation 239
tp: solution with the transportation simplex 241
Maximum flow problem 254
An introduction to cg 258
mcf problem 261
Single-source single-sink variant 261
Multiple-source multiple-sink variant 262
Graph coloring problem 265
sp problem 270
mst problem 272
Routing problems 280
tsp 280
Solution methods for the tsp 283
vrp 284
Widely-used vrp variants 287
vrptw 287
sdvrp 289
Multi-depot vrp 291
pdps 292
Further vrp variants 293
Stochasticity 296
Two-stage stochastic programming 298
Stochastic programming 298
Two-stage recourse problem 299

Contents (continued)

Final words and recommended literature 310
Bibliography 312
List of Acronyms 314
Index 316
Copyright of Figures 318

From theORy to application: learning to optimize with Operations Research in an interactive way

Alessandro Bombelli Bilge Atasoy
Stefano Fazi Doris Boschma

v1.0

This Open Textbook is licensed under a Creative Commons Attribution 4.0 International License, except where otherwise noted. For license details, see http://creativecommons.org/licenses/by/4.0/

The above copyright license, which TU Delft OPEN uses for their original content, does not extend to or include any special permissions granted to us by the rights holders for our use of their content.

Every attempt has been made to ascertain the correct source of images and other potentially copyrighted material and ensure that all materials included in this book have been attributed and used according to their license (see **Colophon** below). If you believe that a portion of the material infringes someone else's copyright, do not hesitate to get in touch with the authors here: a.bombelli@tudelft.nl

Title Open Textbook: **From theORy to application: learning to optimize with Operations Research in an interactive way**
Authors: Alessandro Bombelli, Bilge Atasoy, Stefano Fazi, and Doris Boschma
Publisher: TU Delft OPEN Publishing
Year of publication: 2024
ISBN (softback/paperback): 978-94-6366-851-4
ISBN (E-book): 978-94-6366-850-7
DOI: https://doi.org/10.59490/tb.94
Attribution cover image: Close-up of the hexes defining the game board of *Catan* (previously known as *The Settlers of Catan*), one of the best-selling European board games. Figure by *MorningbirdPhoto*, retrieved here under license CC0

v1.0 [2024-04-12]

Download latest version
The latest version of this book can be downloaded in pdf format here

Source code
The source code of the book, in LaTeX, is available here

Errors? Feedback?
Please report errors or potential improvements here

Colophon
This book was typeset with LaTeX using the `kaobook` class (see here). All figures, unless differently specified, were created using the LaTeX package TikZ. Some icons and figures were retrieved from https://iconoir.com/ and https://pixabay.com/ respectively. We provide an overview of the source websites and copyright for such icons and figures at the end of the book.

The only way to learn mathematics is to do mathematics.

– Paul Halmos

Preface

It is our nature to strive to improve things or rationally try to choose the best option available. For example, when looking for the fastest way to get to our destination or when packing our bag properly so that we do not need a second one. Unconsciously, we apply simple principles of Operations Research (OR). This discipline deals with analytical methods to optimize a large variety of socio-technical problems. Even though OR is quite an intuitive approach to problems, its rigorous models are challenging for students due to their high level of abstraction, especially in fast-paced courses. Education-related literature highlights that a difficult aspect of teaching OR is fostering enthusiasm for developing optimization models (Beliën et al., 2013). Such lack of enthusiasm generally stems from an insufficient understanding of the basic theory and techniques, starting a vicious circle that prevents students from conceptualizing the engineering problem at hand and abstracting it to a mathematical formulation.

A promising approach in engineering education to ease the learning path is **gamification**, i.e., introducing gaming elements in learning processes (Cochran, 2015) and **serious games**, which translate complex and abstract problems into more understandable and engaging games. The primary goal of this book is to combine gamification and serious games with more traditional educational tools, to offer a multi-faceted educational approach to OR education, and to improve the understanding of the topic. This multi-faceted educational tool comprises three pillars as follows:

- this book where several models and solution methods pertaining OR are presented;
- a repository where, for most of the models presented in the book, coded examples and visualizations are provided;
- three board games, one in the form of an online game and two in the form of a print-and-play game, that translate three OR models presented in the book and in the repository into playable versions to enhance students' engagement.

This educational offer is structured in such a way that readers have the freedom to choose the educational tools they find more suitable for them. While there is a strong interlink between the topics treated in this book, the codes in the repository, and the board games, this educational offer is designed in a way that all three can be accessed independently. The variety of complexity in the board games can benefit both more experienced readers and novices.

In terms of content, this book provides an overview of several OR models that have a strong relevance for engineering problems and can be the basis for other extensions. Our selection of the models and methodologies to solve them aligns with much of the teaching portfolio at Delft University of Technology. If readers are interested in more generic and theoretical books, we refer them to Hillier and Lieberman (2015) and Carter et al. (2018).

Open material. This educational project is part of the Open Education Stimulation Fund 2022 (`https://www.tudelft.nl/en/open-science/articles-tu-delft/call-for-proposals-open-education-stimulation-fund-2022`) promoted by Delft University of Technology as part of its Open Science Program (`https://www.tudelft.nl/en/open-science-opbouwportal/about`). As such, the book and the board games are fully open-source and can be freely downloaded here. The repository contains codes written in Python that replicate some of the mathematical models shown in this book using the Python package **pyomo** (`https://www.pyomo.org/`), which is a solver-agnostic package and allows models to be solved both with open-source and commercial solvers. However, some of the provided codes rely on the Python package **gurobipy**, which in turn relies on the commercial solver **Gurobi** (`https://www.gurobi.com/`). With an academic email, a Gurobi license can be obtained that grants access to the full capabilities of the solver.

Because this book mostly targets undergraduate or graduate students, we believe the necessity to rely on a license is not a major problem. This choice stems from the fact that all authors are familiar with such a solver

and regularly use it for their research, which also implies a significant amount of code was already available for this book. To fully comply with an open education paradigm, we aim to fully transition to an open-source setting in future releases of the book.

Who is this book for? The book is mostly designed to support BSc and MSc students from **OR introductory courses** to more advanced. Categorizing the level of OR courses is a challenging task per se, as such a course can be mandatory or elective, a BSc or an MSc course depending on the specific institution and field of study (e.g., Mechanical or Aerospace Engineering, Management Engineering, Computer Science, etc.). More advanced techniques, such as column generation, branch-and-price, and decomposition methods, are not part of this book.

Acknowledgments The contribution of the Delft University of Technology library via the mentioned Open Education Stimulation Fund 2022 is warmly acknowledged. This book is something the three first authors have been planning in the back of their minds for some time, but the grant added momentum (and deadlines) to the initial idea. In particular, we thank Michiel De Jong and Marcell Várkonyi for the tremendous support and the passion they put into open practices and open education in particular. A big thank you to Jacqueline Michielen-van de Riet, who helped us throughout the publication process, and to Kees Moerman who helped with the design of the cover. We would like to thank Federico Marotta, Ken Arroyo Ohori, and Hugo Ledoux for making their *kaobook* LaTeX template (either in its original form or via the *Computational modelling of terrains* book) that serves as the backbone of this book. Some plots generated with the LaTeX package TikZ have been modified from their original versions shared here: we thank Mohammad Namakshenas for sharing the material. We also would like to thank Akshat Kasana, Gracia Bovenberg-Murris, Shaga Eendragt, and Michelle De Smit (the last three from the GameLab, see here) for the help in the development of the serious games. Mihai Constantinov's contribution to the development of some of the codes is highly appreciated. Finally, we would also like to thank some colleagues from the Freight & Logistics Lab (Delft University of Technology) who provided valuable feedback on one of the serious games, as well as the attendees of a workshop during the Education Day 2023 who also played a variant of the same serious game and provided additional feedback.

Book Overview

This book is designed to guide readers through a journey that starts with basic mathematical foundations, continues to the modeling recipe to properly define mathematical models and how to solve them, and finishes with a broad overview of models that reflect real-world operations.

The book is divided into six parts as follows. Part I introduces the concepts of OR and **serious games and gamification** and justifies why such concepts were embedded into this education tool.

Part II provides readers with a **recap on vector and matrix notation**, which is key to the mathematical modeling covered in this book, and with an overview of the ingredients of a mathematical model and of how to set up one.

Part III builds on the previous one and covers the **main solution methods** that can be employed to solve a mathematical model.

Part IV focuses on a first set of mathematical models, i.e., **assignment problems**, whose main goal is to determine how to properly assign items to resources, while Part V focuses on **network problems**, where the goal is to efficiently route resources in a pre-defined environment (the network).

All the models shown in Part IV and Part V are **deterministic** in nature, meaning that all parameters that characterize them are (assumed to be) known with certainty. Because we acknowledge this is not generally the case, we conclude the book with Part VI, where an example of a modeling framework that accounts for **uncertainty** is presented.

The main text of the book adopts a 1.5-column format, utilizing ample white space for side-notes and margin figures where appropriate. This white space also allows readers to add their own notes while navigating through the material.

In the book, we use boxes of different colors to highlight different aspects of the topics covered. The first box type is:

First box type

Light red color and with the lightbulb symbol. It is used to provide additional information that we specifically want to highlight, such as an algorithm or how a constraint can be modeled differently.

the second box type is:

Second box type

Light blue color with the Github symbol. It is used to highlight and provide hyperlinks to our open repository pointing to coded versions of the examples shown in the text.

and the third box type is:

Third box type

Light green color with the gamepad symbol. It is used to highlight and provide hyperlinks to our open repository pointing to playable serious game reinterpretations of some of the mathematical models presented in the text.

About the Authors



Alessandro Bombelli is an assistant professor in the Air Transport & Operations section, part of the Control & Operations Department, at the Aerospace Engineering Faculty at Delft University of Technology. Alessandro co-teaches the MSc courses *Operations Optimisation* and *Airport and Cargo operations*, respectively a profile and elective course for the Sustainable Air Transport MSc program. Alessandro's research emphasis on air cargo operations and logistics closely connects with such courses and the content of this book.



Bilge Atasoy is an associate professor in the Transport Engineering & Logistics section, part of the Maritime & Transport Technology Department, at the Mechanical Engineering Faculty at Delft University of Technology. In relation to the content of this book, Bilge teaches *Quantitative Methods for Logistics*, which is a fundamental course for the MSc programs of Mechanical Engineering (Multi-Machine Engineering track) and Transport, Infrastructure & Logistics (TIL). This course encompasses numerous solution methods and models outlined in the book. Bilge's research closely relates to this area, developing optimization models to achieve adaptive and sustainable transport and logistics systems.



Stefano Fazi is an assistant professor in the Transport and Logistics section, part of the Engineering, Systems, and Services Department, at the Technology, Policy, and Management Faculty of Delft University of Technology. Stefano teaches both BSc and MSc courses revolving around OR models and their applications, especially in the transportation sector. His research line is about the development of algorithms to solve large-scale optimization problems.



Upon completing her MSc degree in Industrial Design Engineering at Delft University of Technology, Doris Boschma transitioned to the faculty of Technology, Policy, and Management. Within this domain, she assumed roles as a designer and project leader at the Gamelab (see here), where she created games for educational and research purposes. Leveraging her knowledge of design, gaming, and diverse learning methodologies, Doris aimed to develop serious games where players could learn, explore, and experience new ideas, worlds, and knowledge.

Contents

Contents **xi**

I Introduction **1**

1 Introduction to Operations Research (OR) **3**
1.1 What is Operations Research (OR) and some historical insights . . . 3
1.2 Preliminary insights into mathematical modeling . . . 4

2 Introduction to serious games and gamification **7**
2.1 Serious games . . . 7
2.2 Gamification . . . 7
2.3 An overview of serious games in OR education . . . 8
2.4 Serious games in this book: setup and learning objectives . . . 9

II Fundamentals **11**

3 A primer on linear algebra **13**
3.1 Vectors . . . 13
3.2 Matrices . . . 15
3.3 Linear systems . . . 16

4 Introduction to mathematical modeling **19**
4.1 Sets . . . 19
4.2 Parameters . . . 23
4.3 Decision variables . . . 23
4.4 Objective function . . . 24
4.5 Constraints . . . 26
4.6 General form of a mathematical model . . . 27
4.7 Construction of a mathematical model . . . 28
4.8 Special types of constraints . . . 33
4.8.1 Big-M notation . . . 33
4.8.2 Either-or constraints . . . 34
4.8.3 K-out-of-N constraints . . . 37
4.8.4 Fixed charge constraints . . . 40
4.9 Final remarks . . . 44

5 Linearization techniques **45**
5.1 Product of decision variables . . . 45
5.1.1 Product of two binary variables . . . 46
5.1.2 Product of a binary and a continuous decision variable . . . 48
5.2 Absolute value . . . 49
5.3 Piecewise linear formulations . . . 51
5.4 If-else statement . . . 53

III Solution Methods **57**

6 The simplex method **59**
6.1 Graphical representation of an LP and corner points . . . 59
6.2 Augmented form of an LP . . . 67
6.2.1 Inequality constraints in the ≤ form . . . 68
6.2.2 Equality constraints . . . 70
6.2.3 Inequality constraints in the ≥ form . . . 71
6.2.4 Final remarks . . . 73
6.3 The simplex method: description of the algorithm . . . 73
6.3.1 Basic and non-basic variables . . . 73
6.3.2 Basic solutions . . . 75
6.3.3 The simplex tableau . . . 77
6.3.4 The simplex algorithm: how to iterate and when to stop . . . 80
6.4 Examples . . . 90
6.5 Additional considerations . . . 100

7 Branch & Bound (BB) **103**
7.1 Motivation for BB . . . 103
7.2 Problem types . . . 104
7.3 The basics of BB . . . 104
7.4 Linear relaxation, root node, and tree structure . . . 105
7.5 Best bound, best incumbent, and gap optimality . . . 107
7.6 A note on functional constraints . . . 110
7.7 Fathoming rules . . . 111
7.8 Branching, bounding, and separation rules . . . 111
7.8.1 Branching options . . . 112
7.8.2 Bounding and separation rules . . . 114
7.9 The BB algorithm in a nutshell . . . 118
7.10 Considerations on the algorithmic complexity of the BB algorithm . . . 120
7.11 An illustrative example . . . 120

8 Branch & Cut (BC) **133**
8.1 Motivation for BC . . . 133
8.2 Examples of cutting planes . . . 136
8.2.1 Gomory fractional cuts . . . 136
8.2.2 Cover inequalities . . . 141
8.2.3 Zero-half cuts . . . 145
8.2.4 List of other cutting planes . . . 145
8.3 Combining BB and cutting planes for an efficient BC . . . 146

IV Assignment Problems **147**

9 Assignment and scheduling problems **149**
9.1 Assignment problems . . . 149
9.1.1 The Hungarian algorithm . . . 151
9.2 Preliminaries of scheduling . . . 158
9.3 Single Machine Scheduling Problem (SMSP) . . . 159
9.4 Parallel Machine Scheduling Problem (PMSP) . . . 163
9.5 p-median problem . . . 166
9.6 Facility location problem . . . 169

10 Packing problems **173**
10.1 KPs . . . 173
10.1.1 0-1 KP . . . 173
10.1.2 Bounded KP . . . 177
10.1.3 0-1 multiple KP . . . 178
10.1.4 Other variants of the KP . . . 179
10.2 BPPs . . . 179
10.2.1 One-dimensional BPP . . . 179
10.2.2 Two-dimensional horizontal BPP . . . 181
10.2.3 Other variants of the BPP . . . 188

V Networks **191**

11 An introduction to graph theory **193**
11.1 Definition of a graph . . . 193
11.2 Properties of a graph . . . 195
11.3 From “abstract" graphs to “concrete" networks . . . 203

12 Network problems **209**
12.1 Transportation Problem (TP) . . . 209
12.1.1 General Setting . . . 209
12.1.2 TP: LP mathematical formulation . . . 210
12.1.3 TP: solution with the transportation simplex . . . 212
12.2 Maximum flow problem . . . 225
12.2.1 An introduction to Column Generation (CG) . . . 229
12.3 Minimum Cost Flow (MCF) problem . . . 232
12.3.1 Single-source single-sink variant . . . 232
12.3.2 Multiple-source multiple-sink variant . . . 233
12.4 Graph coloring problem . . . 236
12.5 Shortest Path (SP) problem . . . 241
12.6 Minimum Spanning Tree (MST) problem . . . 243

13 Routing problems **251**
13.1 Traveling Salesman Problem (TSP) . . . 251
13.2 Solution methods for the TSP . . . 254
13.3 Vehicle Routing Problem (VRP) . . . 255
13.4 Widely-used VRP variants . . . 258
13.4.1 Vehicle Routing Problem with Time Windows (VRPTW) . . . 258
13.4.2 Split Delivery Vehicle Routing Problem (SDVRP) . . . 260
13.4.3 Multi-depot VRP . . . 262
13.4.4 Pickup and Delivery Problem (PDP)s . . . 263
13.5 Further VRP variants . . . 264

VI Stochasticity **267**

14 Two-stage stochastic programming **269**
14.1 Stochastic programming . . . 269
14.2 Two-stage recourse problem . . . 270
14.3 Final words and recommended literature . . . 281

Bibliography **283**

List of Acronyms **285**

Index **287**

Copyright of Figures **289**

List of Figures

1.1 Example of transition from a real-world problem, to a mathematical model and solution, to model revision due to the necessity to include some aspects that were initially omitted. 6

2.1 A typical setting from Dungeons & Dragons. 10

4.1 Example of definition of sets and subsets related to the example provided in Section 4.1. 20
4.2 The iconic lighthouse of Texel. 30
4.3 Feasible regions $\mathcal{F}_1$ (in green) and $\mathcal{F}_2$ (in red) for the problem described in Example 4.5. For the cases $(C_1, C_2) = (1, 1)$ and $(C_1, C_2) = (1, 4)$, the optimal solution is highlighted with a circle: it is respectively $(x_1, x_2) = (2, 4) \rightarrow Z = x_1 + x_2 = 6$ as part of $\mathcal{F}_1$ and $(x_1, x_2) = (5, 2) \rightarrow Z = x_1 + 4x_2 = 13$ as part of $\mathcal{F}_2$. 35
4.4 Feasible regions $\mathcal{F}_1$ (in green), $\mathcal{F}_2$ (in red), and $\mathcal{F}_3$ (in orange) for the problem described in Example 4.6. 37
4.5 A glimpse of the dome of Catania. 42

5.1 Aircraft gated at Amsterdam Schiphol Airport. 47
5.2 Feasible region with the absolute value constraint - convex case. 51
5.3 Feasible region with the absolute value constraint - non-convex case. 51
5.4 Piecewise linear approximation of a non-linear function. 52
5.5 Representation of a customer pair (i, j) served in sequence by a truck. Because of the sequence, a time-precedence constraint $t_j \geq t_i + P_i + T_{ij}$ must hold. 54

6.1 Feasible region for the street food company problem of Example 6.1. 60
6.2 Feasible region for the street food company problem of Example 6.1 with some representative lines defining different objective values Z. 61
6.3 Feasible region for the street food problem of Example 6.1 and objective line if C_1 is increased from 2 to $\frac{15}{4}$. 62
6.4 Feasible region for the street food company problem of Example 6.1 and objective line if C_1 is increased from 2 to 4. 63
6.5 Feasible region for the street food company problem of Example 6.1 if all constraints were in the $\geq$ form. 64
6.6 Example of feasible region for the minimization problem presented in Example 6.2. 65
6.7 Maximization problem of Example 6.3 characterized by the lack of a feasible region. 66
6.8 Corner points for the street food company example. 67
6.9 Feasible region for the street food company case of Example 6.1 if inequality constraint (6.6) is turned into an equality constraint. 70
6.10 Sequence of corner points visited in Example 6.5. 94
6.11 Sequence of corner points visited for Example 6.6. 96
6.12 Corner points and feasible region for the Example 6.7. 98
6.13 Sequence of corner points visited in Example 6.7 (in dark red) and alternative sequence of corner points if x_1 had been chosen in the first iteration (in orange). 100

7.1 Integer feasible solutions for Example 6.4. 103
7.2 Tree structure of BB with binary-only decision variables. Note: in such a model, our goal is to maximize the objective function Z. 106
7.3 Tree structure of BB with integer-only decision variables. Note: in such model, our goal is to maximize the objective function Z. 106

7.4 Example of backtracking strategy. The ordering of the nodes represents the sequence in which they are solved. Note: in our example, for every couple of children nodes, the left one is associated with a rounding down and the right one with a rounding up. In our policy, we always explore the rounded-down node first. . . . 113
7.5 Example of jumptracking strategy. The ordering of the nodes represents the sequence in which they are solved. . . . 113
7.6 Root node and bounds on the objective value of the two children nodes of the variant of the street food company problem of Example 6.1 with increased truck cost. . . . 117
7.7 Root node and solved children nodes for the variant of the street food company problem of Example 6.1 with increased truck cost. Node 1 is colored in green as it resulted in an integer solution. . . . 117
7.8 Complete BB tree for the variant of the street food company problem of Example 6.1 with increased truck cost. Nodes 3 and 4 are colored in red as they are fathomed. . . . 117
7.9 BB decision tree for the build your own pizza problem of Example 7.1: one node explored. . . . 123
7.10 BB decision tree for the build your own pizza problem of Example 7.1: two nodes explored. . . . 126
7.11 BB decision tree for the build your own pizza problem of Example 7.1: three nodes explored. . . . 127
7.12 BB decision tree for the build your own pizza problem of Example 7.1: four nodes explored. . . . 128
7.13 BB decision tree for the build your own pizza problem of Example 7.1: five nodes explored. . . . 129
7.14 BB decision tree for the build your own pizza problem of Example 7.1: six nodes explored. . . . 130
7.15 BB decision tree for the build your own pizza problem of Example 7.1: seven nodes explored. . . . 131

8.1 Example of integer feasible points and LP feasible region. . . . 134
8.2 Example of integer feasible points and LP feasible region reduced to an integer polytope. . . . 135
8.3 Feasible region, integer points (in gray), and optimal solution (in dark orange) of the LP relaxation of (8.10)-(8.13) for the Gomory cutting plane example (before the addition of the cutting plane). 141
8.4 Feasible region, integer points (in gray), and optimal solution (in dark orange) of the LP relaxation of (8.10)-(8.13) for the Gomory cutting plane example (after the addition of the cutting plane, which is depicted in orange). Note that the Gomory cutting plane reduces the feasible region by eliminating the small portion highlighted in red and without cutting off any integer solution. Additionally, the cutting plane passes through integer points $(x_1, x_2) = (3, 3)$ and $(x_1, x_2) = (5, 0)$ and hence defines the missing facet of the integer polytope. . . . 142

9.1 Final solution for the PMSP of Example 9.3. . . . 165
9.2 Location of the customers and facilities for Example 9.4. . . . 167
9.3 Assignment of demand nodes to facilities for different values of p in Example 9.4. The used facilities are highlighted in **green**, the others in **red**. . . . 168
9.4 Objective value (cumulative distance between customers and facilities) as a function of p for Example 9.4. . . . 169
9.5 Location of the customers and facilities for Example 9.5. We also report the index f of each facility to more easily extract its construction cost from Table 9.7. . . . 171
9.6 Assignment of demand nodes to facilities for different values of C_D in Example 9.5. The built facilities are highlighted in **green**, the others in **red**. . . . 172

10.1 The eight outlaws of Example 10.1. . . . 175
10.2 Example of the role of rotation decision variable r_i for an item with original length $L_i = 6$ and width $W_i = 2$, whose left-bottom vertex is placed in the origin $(0, 0)$. If $r_i = 0$, no rotation occurs, resulting in the **green** configuration. If $r_i = 1$, a rotation of $\frac{\pi}{2}$ occurs, resulting in the **red** configuration. . . . 183
10.3 Example of the role of decision variables l_{ij} and b_{ij} in mapping the relative position between items i and j. . . . 184
10.4 The final solution in terms of fields purchased and location of crops to Example 10.2. . . . 187

10.5 Different examples of infeasible, unstable, and stable configurations in a three dimensional BPP. In all three figures, a frontal view of the (x, z) plane is depicted. . . . 189
10.6 Several containers in front of an aircraft. . . . 190

11.1 A graphical representation of G_1. . . . 194
11.2 A graphical representation of G_2. . . . 194
11.3 Examples of objects that closely resemble a graph, but that are not a graph due to failing to satisfy one necessary condition. . . . 194
11.4 Example of undirected graph and its transformation into a directed graph. Note: we assumed that every (v_1, v_2) edge of the undirected version was replaced by both edge (v_1, v_2) and edge (v_2, v_1) in the directed version. . . . 196
11.5 A directed graph G_1 and a subgraph G_2 of G_1. In Figure 11.5b we highlight with light gray the vertices and edges that were removed from G_1 to obtain subgraph G_2. . . . 196
11.6 An undirected graph G_1 and directed graph G_2 used in Example 11.1. . . . 198
11.7 A representation of the famous Königsberg problem. . . . 201
11.8 Example of a directed graph with a closed Hamiltonian walk (**green** arrows) and a cycle (**red** arrows). In **orange** are represented edges of the graph that are neither part of the closed Hamilton walk nor of the cycle. . . . 202
11.9 Example of a DAG. . . . 202
11.10 Graph representation of the airport network from Example 11.4. . . . 204
11.11 Best itinerary for Example 11.4 according to the first (**green** arrows) and second (**red** arrows) traveler. . . . 207

12.1 Generic framework of a TP. . . . 211
12.2 Network representation of the water pipeline of Example 12.1. . . . 227
12.3 Final solution to the maximum flow problem applied to the water pipeline of Example 12.1. . . . 228
12.4 A feasible solution to the variation of the maximum flow problem applied to the water pipeline of Example 12.2. . . . 230
12.5 Solution to the maximum flow problem of Example 12.3 after adding path (S, C, T). . . . 231
12.6 Solution to the maximum flow problem of Example 12.3 after adding path (S, B, C, D, T). The solution matches the one obtained in Example 12.1. . . . 231
12.7 Graph representation $G = (\mathcal{V}, \mathcal{E})$ of the water pipeline network of Example 12.4. . . . 235
12.8 Final solution for Example 12.4. . . . 236
12.9 The Four corners monument at the intersection of Utah (UT), Colorado (CO), New Mexico (NM), and Arizona (AZ). . . . 238
12.10 Graph representation of the 51 states forming the United States of America in Example 12.5. . . . 239
12.11 Final solution to the graph coloring model applied to the 51 states forming the United States of America in Example 12.5.. Four colors are sufficient to have any two neighboring states with different colors. . . . 240
12.12 Graph representation $G = (\mathcal{V}, \mathcal{E})$ of the six state capitals $(\mathcal{V})$ and their connections $(\mathcal{E})$ of Example 12.6. . . . 243
12.13 Examples of spanning trees for graphs with $|\mathcal{V}| = 2$, $|\mathcal{V}| = 3$, and $|\mathcal{V}| = 4$ vertices. . . . 245
12.14 Example of a wrong mathematical modeling approach to compute a MST. . . . 245
12.15 Graph representation $G = (\mathcal{V}, \mathcal{E})$ of the six state capitals $(\mathcal{V})$ and their connections $(\mathcal{E})$ for Example 12.7. . . . 247
12.16 Resulting MST connecting the 6 state capitals. . . . 248

13.1 An example network for the TSP. . . . 253
13.2 Solution to Example 13.1 . . . 254
13.3 Solution to Example 13.2 . . . 258

List of Tables

3.1 First row operation to update an equation of Example 3.1. 17
3.2 Second row operation to update an equation of Example 3.1. 17

6.1 Initial simplex tableau for the street food company problem of Example 6.1. 78
6.2 Initial simplex tableau for the street food company problem of Example 6.1 with the row and column that must undergo changes highlighted in red. 84
6.3 Updated row of the exiting basic variable for the street food company problem of Example 6.1 after the first iteration. 85
6.4 Updated objective row for the street food company problem of Example 6.1 after the first iteration. 85
6.5 Updated (x_3) row for the street food company problem of Example 6.1 after the first iteration. . 86
6.6 Updated (x_5) row for the street food company problem of Example 6.1 after the first iteration. . 87
6.7 Simplex tableau for the street food company problem of Example 6.1 after the first iteration. . . 87
6.8 Initial simplex tableau for Example 6.4. 90
6.9 Simplex tableau for Example 6.4 after the first iteration. 91
6.10 Simplex tableau for Example 6.4 after the second iteration. 91
6.11 Initial simplex tableau for Example 6.5 before correct initialization. 92
6.12 Initial simplex tableau for Example 6.5 after correct initialization. 92
6.13 Simplex tableau for Example 6.5 after the first iteration. 93
6.14 Simplex tableau for Example 6.5 after the second iteration. 93
6.15 Initial simplex tableau for Example 6.6 after correct initialization. 95
6.16 Simplex tableau for Example 6.6 after the first iteration. 95
6.17 Simplex tableau for Example 6.6 after the second iteration. 95
6.18 Simplex tableau for Example 6.6 after the third iteration. 96
6.19 Initial simplex tableau for Example 6.7 after correct initialization. 99
6.20 Simplex tableau for Example 6.7 after the first iteration. 99
6.21 Simplex tableau for Example 6.7 after the second iteration. 99

7.1 Optimal simplex tableau for the full linear relaxation (root node) of the variant of the street food company problem of Example 6.1 with increased truck cost. 115
7.2 Menu with additional toppings and prices of Example 7.1. 121
7.3 Menu with additional toppings and prices of Example 7.1, plus satisfaction S_t and binary decision variable x_t for every topping t. 121

8.1 Optimal tableau of the LP relaxation of (8.10)-(8.13) before the addition of a Gomory cut. . . . 137
8.2 Optimal tableau of the LP relaxation of (8.10)-(8.13) after the addition of a Gomory cut. Because x_5 features a negative value, the addition of the cut makes the current fractional solution infeasible. 138
8.3 Optimal tableau of the LP relaxation of (8.10)-(8.13) after the addition of a Gomory cut and after the dual simplex method has been applied to restore the feasibility of the solution. 139

9.1 Notation for the assignment problem. 150
9.2 Notation for the SMSP. 160
9.3 Notation for the PMSP. 163
9.4 Data pertaining to the six topics of Example 9.3. 164
9.5 Notation for the p-median problem. 166
9.6 Notation for the facility location problem. 170
9.7 Construction cost C_f for each facility $f \in \mathcal{F}$ of Example 9.5. 171

10.1 Notation for the 0-1 KP. . . . 174
10.2 Data pertaining to the eight outlaws of Example 10.1. . . . 175
10.3 Solution to Example 10.1 using the sorting heuristic based on B_o values. . . . 177
10.4 Notation for the bounded KP. . . . 177
10.5 Notation for the 0-1 multiple KP. . . . 178
10.6 Notation for the one-dimensional BPP. . . . 180
10.7 Notation for the two-dimensional horizontal BPP. . . . 184
10.8 Data pertaining to the 16 crops of Example 10.2. . . . 186

12.1 Example of a TP table. . . . 213
12.2 TP table with a dummy destination. . . . 214
12.3 TP table with an initial (infeasible) solution generated with the North-West corner rule. . . . 215
12.4 TP table filled in with the Vogel's method: initial setup. . . . 217
12.5 TP table filled in with the Vogel's method: situation after setting $(S_2, D_5) = 50$ and removing column D_5. . . . 217
12.6 TP table filled in with the Vogel's method: situation after setting $(S_2, D_4) = 10$ and removing column D_4. . . . 218
12.7 TP table filled in with the Vogel's method: situation after setting $(S_2, D_2) = 20$ and removing row S_2. . . . 218
12.8 TP table filled in with the Vogel's method: situation after setting $(S_1, D_1) = 30$ and removing column D_1. . . . 218
12.9 TP table filled in with the Vogel's method: situation after setting $(S_1, D_3) = 20$ and removing row S_1. . . . 218
12.10 TP table with an initial (feasible) solution generated with the Vogel's method. . . . 218
12.11 TP table with the revised problem where cells with flow variables feature a cost coefficient $C'_{sd} = C_{sd} - u_s - v_d = 0$. . . . 221
12.12 TP table with the revised problem after having made $x_{3,5}$ basic and $x_{1,1}$ non-basic, but before having updated the u_s and v_d coefficients. . . . 223
12.13 TP table with the revised problem after having made $x_{3,5}$ basic and $x_1, 1$ non-basic and having updated the u_s and v_d coefficients. . . . 224
12.14 Notation for the maximum flow problem. . . . 226
12.15 Notation for the single-source single-sink MCF problem. . . . 233
12.16 Notation for the multiple-source multiple-sink MCF problem. . . . 234
12.17 Notation for the graph coloring problem. . . . 237
12.18 Notation for the SP problem. . . . 242
12.19 Notation for the MST problem. . . . 246

13.1 Notation for the TSP. . . . 252
13.2 Distance matrix for Example 13.1 . . . 253
13.3 Notation for the CVRP. . . . 255
13.4 Additional notation for the VRPTW . . . 259

14.1 Input data for Example 14.1. . . . 274
14.2 Different solutions for Example 14.1. *Stoc. Sol.* stands for the solution from the two-stage stochastic model. *Det. Sol.* stands for the solution from the deterministic counterpart using the weighted average demand. *Det. Sol.* $s = *$ stands for the outcome of the deterministic solution in case of realized demand from scenario $*$. . . . 276
14.3 WS solution to Example 14.1. . . . 277
14.4 Data pertaining to the four outlaws in the first stage of Example 14.2. . . . 279
14.5 Data pertaining to the two outlaws for each scenario $s \in \mathcal{S}$ in the second stage of Example 14.2. 279
14.6 WS solution to Example 14.2. . . . 280

1 List of figures using images or logos from various sources. For each figure, we provide a brief description, the reference website, and the specified copyright. 289

Part I

Introduction

Introduction to Operations Research (OR) 1

The conclusions of most good operations research studies are obvious.

Robert E. Machol

1.1 What is Operations Research (OR) and some historical insights 3
1.2 Preliminary insights into mathematical modeling . . 4

1.1 What is Operations Research (OR) and some historical insights

OR[1] can be summarized as the development of (advanced) quantitative methods, mostly mathematical models, to assist decision-makers in understanding, analyzing, and improving the performance of a system under scrutiny. The quantitative methods can vary significantly in the mathematical approach, complexity, and scalability, among other features. In this book, we will focus on specific types of mathematical models that are linear (or that can be linearized).

1: In some domains, e.g., Management Engineering and Management Sciences, the terms Operations Research and Management Science are used interchangeably. We will employ Operations Research in the context of this book.

Many systems can be modeled and analyzed using OR tools. In principle, any system with causes/effects components can be translated into a mathematical model that captures, in a simplified way, all the dynamics and relationships of the original system. Such a model can then be analyzed and (attempted to be) solved with a suitable solution technique. Solving it entails improving in the best possible way a set of Key Performance Indicator (KPI)s deployed to map the effect of some changes in the system to the final performance. Examples of KPIs can be productivity, profit, overall traveled distance, costs, number of dissatisfied customers, and overall accrued delays, just to name a few.

A common trait of several mathematical models is their complexity and sometimes the absence of non-trivial solutions. Such systems are usually characterized by different, generally contrasting, requirements that make it extremely hard to improve while satisfying all the requirements.

To anticipate some OR jargon that we will use extensively in the context of this book we define:

1. **objective function**: a function that collects all KPIs that we use in our decision-making process to assess the “quality" of a solution. Given the list of KPIs provided above, it follows that each KPI of interest has a specific direction to follow. For example, KPIs such as productivity or profits should increase, whileKPIs such as costs or overall accrued delays should decrease. For some systems, a mix of KPIs could be employed so that some of them should increase and others should decrease;
2. **constraints**: the set of different, and generally contrasting, requirements characterizing the problem at hand. Such constraints bound the flexibility of the set of actions we can take to improve the system. They can take the form of a monetary budget, a limited number of workers to employ, a limited fleet of trucks to operate, a minimum

wage workers should get, or a maximum quantity of tonnes of crop that can be harvested, for example.

Among the several domains where OR can contribute, transportation systems of any kind (road, rail, air networks) have been researched extensively with OR techniques. Historically, similarly to other technological advancements (e.g., drone technology), military interests boosted the rise of OR, which became a recognized term and discipline during the advent of World War II. In particular, the necessity to deploy materials, supplies, weapons, and devices of any kind at an unprecedented scale forced the military to recruit scientists in order to improve logistics performances and outperform the enemy. During the 1930s, the term *operational research* was coined by British scientists (with *operations research* being the United States equivalent term, which we are using in the book).

After the end of World War II, the analytical methods developed during the war were not forgotten, and they were redirected to other domains. In addition, advancements in computer technology have made it possible to scale up algorithms and address problems of larger size and complexity. In the 1950s, OR became so widespread as a research topic that academics that would label themselves as mathematicians, engineers, etc., would redefine themselves *operations researchers* and would found the first societies specifically devoted to OR. With a similar parallelism to the different terms used in the United Kingdom and the United States, similar societies were founded in the two countries: the *Operational Research Society* in Britain, the *Operations Research Society of America* and *The Institute of Management Science* in the United States.

In the latter case, the overlap between the two societies was so evident that they eventually merged into the INstitute For Operations Research and Management Science (INFORMS) (`https://www.informs.org/About-INFORMS`), which is nowadays the largest international association for professionals in operations research, analytics, management science, economics, behavioral science, statistics, Artificial Intelligence (AI), data science, applied mathematics, and other relevant fields, with over 12,500 members worldwide. Given the list of topics covered by INFORMS it should be noted that, especially in recent times, OR is not a standalone technique, but encompasses several fields and that it can be used in conjunction with related domains. Such synergy of techniques has flourished in the last decades and extended OR techniques to new domains, e.g., when combining classic OR tools with behavioral sciences to better embed human behavior in an optimization framework, or has made it possible to address larger scale problems, e.g., when combining AI with OR.

1.2 Preliminary insights into mathematical modeling

The core of OR is to build **mathematical models** that are an **idealized and simplified version** of the real-life problem under scrutiny. A mathematical model represents the bridge between a real-life problem and a proper algorithm capable of "understanding" and solving such a model. With such logic, an OR practitioner is basically translating something

non-numerical (the real-life problem) into something numerical (the mathematical model) that a proper algorithm can solve. In OR, the term **optimality** is key. Optimality means that an algorithm has solved a specific mathematical model in *the best possible way*. This expression means that we found a specific set of actions to be taken such that our objective cannot get any better without violating at least one of the constraints of the mathematical model. We call this set of actions an **optimal solution**.

Notwithstanding, when we talk about optimization problems, we are not necessarily interested in a solution that is optimal. In fact, for some complex problems, it may not even be feasible to find the optimum in a reasonable time frame. Some may require years of continuous computation to be solved. Hence, it is up to the decision-maker to judge the required level of **solution quality** and assess if a good enough and fast solution is better or worse than an optimal one depending on many factors, resolution time being the most important. A good understanding of the original problem is of course key. We can argue that any solution that improves current operations should be, at least, labeled as acceptable.

Partially related to solution quality is **model complexity**. It is up to the decision-maker to decide how much complexity to add to the model and what to simplify. In principle, a mathematical model that captures more features of the real-world problem should provide a more realistic solution, but it might also be harder to solve. We also would like to stress that "more complicated" does not necessarily mean "closer to the real-world problem". There could be a very complicated mathematical model that does not address some crucial features of the associated real-world problem, while it contains some other irrelevant features. Even in OR practice, the Occam's razor rule[2] still applies. Namely, given a real-world problem, we are looking for the simplest mathematical model that enables us to capture all the required elements of the system.

2: We refer readers to this Wikipedia page for a more exhaustive description of the concept.

An overly complex model might be impossible to solve or, if solved, its solution might be very cumbersome to analyze and hence implement. Conversely, a very simplistic model might yield an optimal solution very quickly. Still, the lack of necessary detail would make such a solution of little to no use in the context of the original problem. Deciding where to draw the line is an acquired skill that OR researchers learn with time and many trial-and-error feedback loops. In principle, the best model is the one that strikes a balance between capturing enough of the complexity of the original problem so that it yields meaningful results while being interpretable and computationally tractable. We should never forget that OR was born to assist decision-makers in making better decisions in the context of real-life problems, and the most mathematically pristine model is of little use if the only application is a computer simulation.

Related to the previous point, we also want to stress that a good OR model can also help in achieving a better understanding, compatibly with its assumptions, of the real-world problem and in exploring solutions and scenarios that human intuition might struggle to see. Of course, we should never forget that every mathematical model is a simplification, in some form, of the original problem, and hence a solution to a mathematical model might not be 100% transferable to the original problem.

To illustrate the preceding point, let us examine Figure 1.1. The left segment depicts Milan's (Italy) subway network, comprising five lines.

The middle portion illustrates a graph (we will see more about graphs in Chapter 11), a fundamental mathematical construct that underpins a model representing the actual subway system. This mathematical model could, for instance, optimize each line's hourly frequency to meet passenger demand effectively while keeping operational costs in check. The output of the mathematical model is a solution that is well-suited for the mathematical model itself. The efficacy of such a solution for the real-world problem the mathematical model approximates depends on many factors, primarily on how strong the assumptions and simplifications introduced into the mathematical model are. This is visually represented by the right portion of Figure 1.1. If the modeler realizes some of the simplifications make the solution not applicable to the original problem, then the mathematical model must be revised. In the end, mathematical models are devised to solve real-world problems. Hence they should be tractable mathematically while capturing enough complexity of the real-world problem itself.

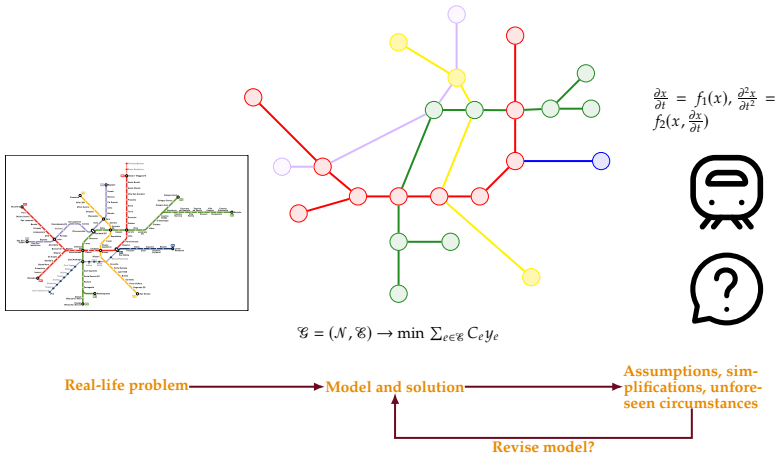


Figure 1.1: Example of transition from a real-world problem, to a mathematical model and solution, to model revision due to the necessity to include some aspects that were initially omitted.

Introduction to serious games and gamification

2

2.1 Serious games 7
2.2 Gamification 7
2.3 An overview of serious games in OR education . . 8
2.4 Serious games in this book: setup and learning objectives 9

> It's not whether you win or lose, it's how you play the game.
>
> Grantland Rice

In this chapter, we explain more in detail the concepts of serious games and gamification, respectively in Section 2.1 and Section 2.2, that were introduced in the preface. Then, we provide some examples of serious games, gamification, or a combination of the two applied to OR education in Section 2.3. Finally, we introduce the setup and main learning objectives of our own serious games that complement this book in Section 2.4.

2.1 Serious games

A serious game is a game developed with a purpose other than just entertainment. The purpose of a serious game can be anything, from education or exploration to persuasion or informing to assessment or data collection. **A serious game is often developed for a specific situation or environment and, as such, requires a good understanding of the addressed situation by the developer**.

Children play; it is one of the main ways they learn (Essame, 2020). They experiment, experience, and create new connections. When children play, they can freely and safely do these things. There is a safe circle. If they fail, fall, or find out their current approach does not yield the expected outcome, they can start over without any consequences. What is important is that playing offers the player control and the ability to choose and try. An overarching goal of serious games is to provide that very same sense of control (Undiyaundeye, 2013).

A key feature of serious games is to encourage users to adopt a "trial and error" approach that is generally harder to implement with more traditional learning mechanisms. In most games there is a "best" strategy to win or achieve the final objective. Notwithstanding, the goal of a serious game is to help learners understand the intricacies of the underlying topic by playing, and possibly failing multiple times in doing so.

2.2 Gamification

Gamification is the application of game design elements and principles in non-game contexts to enhance engagement, motivation, and behavior. **It involves incorporating game-like features such as points, levels, badges, challenges, and leaderboards into activities that are typically not considered games**. The goal of gamification is to make tasks or processes more enjoyable, interactive, and rewarding, thereby encouraging

participation, learning, and achievement. While the concept of gamification might, and in principle does, overlap with the concept of serious games, the two concepts differ in terms of depth and goals. Gamification does not necessarily entail the development of a full game to be played, but rather introduces game-like elements in traditional teaching methods. Serious games, as the name suggests, are full-blown games instead (*ELB learning* 2024). **While adding gamification elements to an activity does not imply the design of a serious game, the design of a serious game generally involves by definition gamification elements such as points and levels**. We will expand on this in Section 2.4.

A final consideration about gamification addresses its ability to engage and motivate users. By tapping into the psychological mechanisms that drive intrinsic motivation and enjoyment in games, gamification seeks to motivate individuals to complete tasks, achieve goals, and adopt desired behaviors. It is commonly used in various fields such as education, marketing, health and wellness, employee training, and customer engagement to promote desired outcomes and create more engaging experiences. When focusing on education, gamification elements are not necessarily designed to transfer knowledge or replace traditional teaching methods. They are rather employed to engage students and increase their curiosity so that the “more traditional" teaching methods are more effective and knowledge retention is boosted.

2.3 An overview of serious games in OR education

Serious games, when applied to OR, serve as powerful tools for both education and decision-making. These games aim at simulating real-world scenarios by adding simplifications so that the game remains understandable and playable, yet with enough complexity to well approximate the original problem. They allow users to engage with complex OR concepts in an interactive and immersive environment. By combining entertainment with learning, serious games enhance comprehension and retention of OR principles, making abstract concepts more tangible and accessible. While serious game do not fully replace the theory behind, they make it more tangible and interpretable when studied or reviewed after playing the game. Board games inherently involve players adhering to a set of rules that dictate how the game unfolds. Interestingly, these rules can be translated into constraints within OR models. **Successfully navigating the game requires players to either satisfy these constraints or adapt their strategy accordingly**. This concept underscores a fundamental aspect of OR, where OR problems are framed within a structured framework akin to the rules of a board game, emphasizing the importance of strategic decision-making and problem-solving.

To this avail, serious games can simulate supply chain optimization, inventory management, or production scheduling, providing users with hands-on experience in solving OR problems. Furthermore, serious games can be used as decision support systems, enabling stakeholders to explore different strategies and scenarios, evaluate their consequences, and make informed decisions. Overall, serious games offer a dynamic and

engaging approach to learning and applying OR techniques, fostering deeper understanding and effective problem-solving skills.

A few examples of OR-related serious games are:

- **The beergame** (*The Beergame* 2024): a role-play supply chain simulation that immerses students in typical supply chain challenges. Participants enact a four-stage supply chain where the objective is to produce and deliver units of beer. Beginning at the factory, participants navigate through three subsequent stages, each responsible for delivering beer units until they reach the customer at the downstream end of the chain. A key feature of this game is to introduce players to the concept of the **bullwhip effect**[1];
- **You've got freight!** (*You've got Freight* 2024): a cutting-edge serious game on synchromodality designed for educational institutions and companies aiming to enhance collaboration. Through the game, participants gain insights into running a sustainable business, cutting carbon dioxide emissions, ensuring reliable delivery, fostering intentional collaboration, and cutting costs effectively for their companies;
- **The burrito optimization game** (*Burrito Optimization Game* 2024): a serious game designed to teach participants about supply chain management and optimization. Participants take on the role of managing a burrito restaurant chain. They are tasked with making decisions related to inventory management, production scheduling, and distribution logistics to meet customer demand while minimizing costs and maximizing profits. Through this interactive experience, participants learn about key concepts such as demand forecasting, inventory control, production planning, and transportation logistics in a fun and engaging way.

1: The bullwhip effect happens in supply chains when orders placed with suppliers vary a lot more than actual sales to customers. This causes a big increase in demand variation as you go further back in the supply chain. Essentially, it means that small changes in what customers want can cause big swings in inventory levels the further you go up the chain. For more information on the bullwhip effect we refer interested readers to this Wikipedia page.

2.4 Serious games in this book: setup and learning objectives

Our aim with the games you can play throughout this book is to help students in their education. The games are designed to aid in understanding various theories and allow students to explore these theories in practice. Some people learn best while reading, others while listening or looking at pictures, and some learn through experimentation. By including the games, we offer you an opportunity to experiment with optimization theories. We aimed to take the abstract concept of optimization and the steps it involves and put it into "practice."

Each serious game we provide entails different levels of increasing complexity, hence embedding a powerful gamification element. In fact, the first level is generally quite easy and solvable "by hand" without the help of an OR setting, let alone a code. The more challenging levels pose additional challenges because of an increase in the size of the problem and of the spectrum of decisions that can be taken. The goal is not to have students haplessly struggle to find a solution, but rather to witness first-hand how scalability in OR problems is an issue and why algorithms are paramount. Notwithstanding, playing these complicated levels is still crucial, as players must anyway understand the rules and the setting,

which entails understanding the underlying principles of the associated OR model. **Ultimately, the effectiveness of a code in modeling and solving an OR problem hinges on the modeler's comprehension of the problem and their accurate translation of it into code**.

If you do not feel like playing or trying the serious games as you believe they do not work for you, that is okay. The games are offered as an additional fun tool with a pedagogical effect and they support the book and the coded examples, but they are not designed as mandatory tools. All of them are designed to be played either singularly or as a group. In the latter scenario, they facilitate collaborative decision-making, team-building, and the sharing of knowledge and contrasting ideas to collectively arrive at the optimal solution-an inherent principle of OR.

In this book, serious games are presented within a fantasy setting, chosen deliberately to illustrate fundamental theory elements in a distinct context. By concentrating on these core elements, we demonstrate their applicability across various scenarios. While the games are interconnected within the same fantasy setting, each one functions independently, allowing readers to select and engage with them individually. The games are accompanied by an optional story. Players have the choice to engage with the story or not, as the rules are presented independently. Sci-fi fans or tabletop Role-Playing Game (RPG) lovers[2] might feel compelled to go through the full story, accompanying each game, but otherwise players can directly read the rules and play the game.

After playing the game, take some time to answer the included questions either individually or with your peers. These questions are designed to help you reflect on your gameplay experience and relate it back to theoretical concepts and real-world applications. Following the debrief, you will find information on the game's design and its connection to the underlying theory.

While the game environment allows for experimentation and failure, it is important to approach it with seriousness. Despite its fantasy setting, the problem-solving techniques you employ mirror those used in real-world scenarios. Therefore, treat the task of finding solutions seriously. Do not settle for just any solution; strive to uncover the optimal one, using each attempt as a learning opportunity to refine your approach.

In essence, serious games provide a risk-free space for exploration and experimentation. Your performance in these games will not affect your grades or have real-world consequences. They are designed to offer a unique perspective on understanding theoretical optimization concepts. Each game is self-contained, but they all contribute to a unified fantasy setting. While the accompanying story is optional, it adds context to the gameplay. However, the debrief session is essential. It prompts reflection on your experience and learning, helping you solidify your understanding of the material.

2: A tabletop RPG, also known as a pen-and-paper RPG, involves participants describing their characters' actions verbally or through gestures. Character actions are determined by their traits, and outcomes are determined by a set system of rules, often involving dice rolling. Players have the freedom to improvise within the framework of the rules, influencing the game's direction and outcome. One of the most famous tabletop RPGs of all times is *Dungeons & Dragons* (Figure 2.1): we refer readers to this Wikipedia page for more info.



Figure 2.1: A typical setting from Dungeons & Dragons.

Part II

Fundamentals

A primer on linear algebra | 3

3.1 Vectors 13
3.2 Matrices 15
3.3 Linear systems 16

Dear algebra, please stop asking us to find your X: she's never coming back and don't ask Y.

Anonymous

Linear algebra is one of the foundations of mathematical modeling, as many mathematical models (at least, most of the ones treated in this book) are entirely or substantially linear in nature. In this chapter, we provide a brief overview of **vectors** in Section 3.1 and **matrices** in Section 3.2, i.e., the two main ingredients of linear algebra. Their combination in the context of linear systems is described in Section 3.3. We also partially contextualize vectors and matrices in relationship to mathematical modeling, so that this chapter serves as a prerequisite to Chapter 4.

3.1 Vectors

Let us start by defining vectors. An n-dimensional vector comprises a sequence of n numbers or scalars. In practice, such a sequence of elements could be vertically or horizontally stacked. In the rest of this book, we assume that vectors are $(n, 1)$ vertical operators, i.e., columns. The n represents the number of elements (or rows of the vector), and the 1, albeit redundant, represents that a vector can be considered a single column. **Hence, when defining a vector as n-dimensional, we refer to the number of elements it contains. Conversely, every vector is a one-dimensional structure, meaning that the n elements it contains are arranged along a single direction (a column, in our convention). Hence, defining a vector an n- or one-dimensional entity are, while sounding confusing, both correct interpretations depending on the intended meaning.** [1] Vectors can represent various entities, such as items or properties. There are two fundamental natures of vectors. First, they can be filled with known information, constituting a collection of **parameters** or **coefficients**. Alternatively, vectors can consist of unknown terms, termed **variables**.

1: As a matter of fact, many programming languages such as Python, consider vectors as one-dimensional entities, generally called *one-dimensional arrays*. Along this single dimension, they can contain any number n of elements. We refer readers to this Wikipedia page for more info.

Consider a collection of 5 items. Each item has a certain known quality, that we call "weight" (a parameter). We can decide to give an identification letter to this measure, say W. In order to identify the weight for each item, we need a so-called **index**. It is customary to choose an identification letter for indexes. Common letters are i, j, or k. However, there is no fixed rule, as we can choose any name for anything. A rule for giving the name to vectors is to select a letter that represents to some extent the quantity of interest (for example, its initial). We will discuss more about this in Chapter 4.

In our case, we choose i as the index because it is a customary choice and, in addition, it is also the initial letter for "item". The expression W_i then

represents the weight of a generic item i. If the weights are unknown, we can write:

$$\overrightarrow{W} = \begin{pmatrix} W_1 \\ W_2 \\ W_3 \\ W_4 \\ W_5 \end{pmatrix} \tag{3.1}$$

where the expression $\overrightarrow{W}$ highlights that we are dealing with a vector. Knowing the weights, we could replace each W_i with its actual value as shown in (3.2).

$$\overrightarrow{W} = \begin{pmatrix} 3 \\ 7 \\ 2 \\ 0.1 \\ 4.5 \end{pmatrix} \tag{3.2}$$

If a $(1, n)$ row vector is required, then it is sufficient to use the **transposition** operator $\overrightarrow{W}^T$ which translates a column vector into a row one (and vice versa). Several operations are possible with vectors:

- accessing a specific element[2]. If we are interested in the weight of item 2, we can access $\overrightarrow{W}(2) = 7$;
- summation of all elements. It can be expressed in several ways, such as $\sum_{i=1}^{5} W_i = 16.6$ or $\sum_{i=1}^{|\overrightarrow{W}|} W_i = 16.6$. In the first case, we explicitly specify the maximum number of elements (i.e., 5), while the second case is more generic (and mathematically more pristine) because it defines as the final index of the summation the cardinality[3] of vector $\overrightarrow{W}$, i.e., $|\overrightarrow{W}|$;
- scalar multiplication. Given a vector $\overrightarrow{W}$ and a scalar α, we can generate a new vector $\overrightarrow{V} = \alpha\overrightarrow{W}$ where each element V_j is given by $V_j = \alpha W_j$. In our example, if $\alpha = \frac{1}{\rho}$ (with ρ being the density) $\overrightarrow{V} = \frac{1}{\rho}\overrightarrow{W}$ would return the volume of each item;
- vector addition. Given two vectors $\overrightarrow{X} = (x_1, x_2, \cdots, x_n)^T$ and $\overrightarrow{Y} = (y_1, y_2, \cdots, y_n)^T$ (both n-dimensional), we define their element-by-element summation as $\overrightarrow{Z} = \overrightarrow{X} + \overrightarrow{Y} = (z_1, z_2, \cdots, z_n)^T$, where each element is expressed as $Z_i = X_i + Y_i$. **Note: element-by-element vector addition is only possible if the two vectors contain the same number of elements, i.e., if $|\overrightarrow{X}| = |\overrightarrow{Y}|$;**
- scalar product, also known as **dot product**. Given two vectors $\overrightarrow{X}$ and $\overrightarrow{Y}$, their dot product is a scalar value obtained by summing all the element-by-element products $X_i Y_i$. We can express the dot product as $\sum_{i=1}^{|\overrightarrow{X}|} X_i Y_i$ **or, equivalently, $\overrightarrow{X}^T \cdot \overrightarrow{Y}$ where $\cdot$ is the symbol of the dot product**. Note that, assuming vectors are defined as column vectors, we need the first vector to be a row vector to obtain a scalar.

2: For programming languages that start their enumeration from 0, such as Python, accessing the n-th element of a vector requires specifying index $n - 1$. For example, the second element of array $\overrightarrow{W}$ is $\overrightarrow{W}(1)$

3: The cardinality of a set of elements is the number of elements part of the set.

In fact $(1,n)\times(n,1)=(1,1)$, where the final dimension $(1,1)$ implies a scalar value. Let us consider a course where the final grade is the weighted average of 3 partial exams. The 3 grades we obtained are stored in $\overrightarrow{G}=(8,8.5,8)^T$ and the 3 weights are $\overrightarrow{W}=(0.2,0.3,0.5)^T$. We can compute our final grade using the dot product operator as $\overrightarrow{W}^T\cdot\overrightarrow{G}=(0.2,0.3,0.5)\cdot(8,8.5,8)^T=0.2\times8+0.3\times8.5+0.5\times8=8.15$.

3.2 Matrices

Let us extend the definition of a vector, by adding an extra dimension to it. For vectors, as seen in Section 3.1, one dimension can change (the length n of the vector, i.e., the number of elements it contains) while the other is fixed to 1[4]. If we allow both dimensions of a vector to be greater than 1, then we obtain a two-dimensional array of numbers, called a matrix. For example, matrix

4: This further corroborates the fact that vectors are one-dimensional structures.

$$A=\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix} \tag{3.3}$$

is an (n,m) matrix. A is defined as a square matrix if its dimensions are equal ($m=n$). The individual elements of A are denoted as a_{ij} and are referred to as its elements. **Any matrix can be used to extract vectors by "slicing" specific rows or columns, as ultimately, a matrix is essentially a collection of vectors**. For example, $A(2,)$ represents a horizontal slice where we retrieve all the elements of the second row of the A matrix[5]. Hence, it is a row vector. $A(,2)$ represents the column vector obtained by isolating the second column of A instead.

5: For programming languages starting the enumeration from 0, as stated in a previous sidenote, $A(2,)$ retrieves the third row of the two-dimensional array A.

Similar to vectors, several important mathematical operations for matrices include:

- scalar multiplication. Scalar multiplication of a matrix A and a real number α yields a new matrix $B=\alpha A$ that has the same dimensions as the original matrix A. Its elements b_{ij} are given by $b_{ij}=\alpha a_{ij}$;
- addition of two matrices. Element-by-element addition of two matrices A and B yields a new matrix $C=A+B$, whose elements c_{ij} are given by $c_{ij}=a_{ij}+b_{ij}$. **Note: only matrices with the same dimensions can be added element-by-element to create a new matrix**;
- multiplication of matrices, AB. It is possible only if matrix A has the same amount of columns as the rows of B. This is necessary as element (i,j) of the new matrix is determined by the dot product $A(i,)\cdot B(,j)$. It also follows that the resulting matrix $C=AB$, if $A=(n,k)$ and $B=(k,m)$, will be of size (n,m). We can define each element C_{ij} as $C_{ij}=\sum_{p=1}^{k}a_{ip}b_{pj}$. For example:

$$\underbrace{\begin{pmatrix}1 & 2\\ 0 & -3\\ 3 & 1\end{pmatrix}}_{(3,2)}\underbrace{\begin{pmatrix}2 & 6 & -3\\ 1 & 4 & 0\end{pmatrix}}_{2,3}=\underbrace{\begin{pmatrix}4 & 14 & -3\\ -3 & -12 & 0\\ 7 & 22 & -9\end{pmatrix}}_{(3,3)} \tag{3.4}$$

is the product of a $(3,2)$ and a $(2,3)$ matrices, and hence yields a $(3,3)$ matrix;

- **transposition** of a matrix A, i.e., A^T, entails swapping the rows with the columns of A. Thin means that the transpose of an (n,m) matrix is an (m,n) matrix. For example:

$$\underbrace{\begin{pmatrix}2 & 4 & -1\\ -3 & 0 & 4\end{pmatrix}^T}_{(2,3)}=\underbrace{\begin{pmatrix}2 & -3\\ 4 & 0\\ -1 & 4\end{pmatrix}}_{(3,2)} \tag{3.5}$$

where the transpose operator transforms a $(2,3)$ matrix into a $(3,2)$ matrix in (3.5);

- **inversion** of a matrix A. A square (n,n) matrix (if invertible) is associated with its inverse matrix A^{-1} such that the following relationship[6] holds: $A^{-1}A = \mathbb{I}_n$, where with $\mathbb{I}_n$ we mean the **identity matrix**[7] of size (n,n).

6: We refer readers to this Wikipedia page for some guidelines on how to perform matrix inversion.

7: An identity matrix is a square matrix with all 1s along the main diagonal and 0s everywhere else.

3.3 Linear systems

A linear system is, in its most general form, a combination of two vectors and a matrix as:

$$A\vec{x} = \vec{b} \tag{3.6}$$

where $\vec{x}$ is the column vector containing some unknown variables, A is the coefficient matrix, and $\vec{b}$ the right-hand side vector containing some constants. To isolate and solve vector $\vec{x}$, A must be square and invertible. If these two conditions are met (we assume A is an (n,n) invertible matrix), we can modify (3.6) as:

$$A^{-1}A\vec{x} = A^{-1}\vec{b} \rightarrow \mathbb{I}_n\vec{x} = A^{-1}\vec{b} \rightarrow \vec{x} = A^{-1}\vec{b} \tag{3.7}$$

where the unknown vector $\vec{x}$ is now expressed as a product of a known matrix and vector. Hence, we can use the inversion operator introduced in Section 3.2 to solve a linear system pending that matrix A is invertible[8]. We elaborate on this in Example 3.1.

8: Note that if matrix A is non-invertible, it suggests that the equations comprising the linear system are not linearly independent, implying that the system typically has no solution.

Example 3.1 Let us consider the following linear system of 3 equations in 3 unknowns x_1, x_2, and x_3:

$$x1 + 2x_2 = 5 \tag{3.8}$$
$$2x1 + 3x_3 = 12 \tag{3.9}$$
$$x_2 + 4x_3 = 9 \tag{3.10}$$

which can be translated into a matrix and vector form as:

$$\begin{pmatrix} 1 & 2 & 0 \\ 2 & 0 & 3 \\ 0 & 1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5 \\ 12 \\ 9 \end{pmatrix} \tag{3.11}$$

After computing the inverse of the coefficient matrix:

$$\begin{pmatrix} 1 & 2 & 0 \\ 2 & 0 & 3 \\ 0 & 1 & 4 \end{pmatrix}^{-1} = \frac{1}{19} \begin{pmatrix} 3 & 8 & -6 \\ 8 & -4 & 3 \\ -2 & 1 & 4 \end{pmatrix} \tag{3.12}$$

we can determine our unknowns as:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \frac{1}{19} \begin{pmatrix} 3 & 8 & -6 \\ 8 & -4 & 3 \\ -2 & 1 & 4 \end{pmatrix} \begin{pmatrix} 5 \\ 12 \\ 9 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix} \tag{3.13}$$

Matrix inversion is not the sole method for solving a linear system. In a linear system, like the one shown above, the solution does not change if we replace an equation with a linear combination of some of the original equations. **If we manage to "isolate" an unknown (e.g., x_1) in one equation through a series of row operations, we can directly determine its value and then substitute it into the other equations to deduce the values of the remaining unknowns**.

For example, we could replace (3.8) with a linear combination of itself minus twice times (3.10), as shown in Table 3.1.

x_1	$2x_2$		=	5	-
	$2x_2$	$8x_3$	=	18	=
x_1		-$8x_3$	=	-13	

Table 3.1: First row operation to update an equation of Example 3.1.

By eliminating x_2, we added x_3 to the updated version of (3.8). Hence, we could replace it with a linear combination of itself plus $\frac{8}{3}$ times (3.9) as shown in Table 3.2.

x_1	-$8x_3$	=	-13	+
$\frac{16}{3}x_1$	$8x_3$	=	32	=
$\frac{19}{3}x_1$		=	19	

Table 3.2: Second row operation to update an equation of Example 3.1.

Hence, we managed to successfully isolate x_1 in the revised equation so that $\frac{19}{3}x_1 = 19 \rightarrow x_1 = 3$ as already highlighted by the previous solution obtained with matrix inversion. The revised linear system is:

$$\frac{19}{3}x1 = 19 \tag{3.14}$$

$$2x1 + 3x_3 = 12 \tag{3.15}$$

$$x_2 + 4x_3 = 9 \tag{3.16}$$

We can now substitute $x_1 = 3$ (as obtained from (3.14)) in (3.15) to get $x_3 = 2$, which is in turn substituted in (3.16) to obtain $x_2 = 1$.

Note that we obtained the same solution that was already available to us, but using linear combinations of the original equations instead of matrix inversion. This key concept of row operations will be one of the cornerstones of Chapter 6.

Introduction to mathematical modeling

4

4.1 Sets 19
4.2 Parameters 23
4.3 Decision variables 23
4.4 Objective function 24
4.5 Constraints 26
4.6 General form of a mathematical model 27
4.7 Construction of a mathematical model 28
4.8 Special types of constraints 33
4.8.1 Big-M notation 33
4.8.2 Either-or constraints . . . 34
4.8.3 K-out-of-N constraints . . 37
4.8.4 Fixed charge constraints . 40
4.9 Final remarks 44

In general, analytical and numerical methods are used for quantitative problems that do not entail a high level of complexity or that have particular properties, easy to describe with formulas. Such problems may allow for several assumptions. In many cases, calculations and graphical representations are required.

Typically, any quantitative problem is defined by three major elements: sets, parameters, and variables. Parameters and variables are then intertwined in a two-fold manner: they are combined to define the specific objective of the problem at hand and a set of constraints that bound the problem. It is the first task of the modeler to spot in the description of a problem all these elements, especially before dealing with complex techniques, such as mathematical modeling. We define some rules regarding our notation in the **Notation: sets, parameters, and decision variables** box.

Notation: sets, parameters, and decision variables

In the rest of the book, unless differently specified, we will use a calligraphic style for sets (e.g., $\mathcal{S}$), upper-case letters for parameters (e.g., S), and lower-case letters for decision variables (e.g., s).

We describe sets, parameters, and decision variables in Section 4.1, Section 4.2, and Section 4.3 respectively. Then, in Section 4.4 and Section 4.5 we discuss how these elements can be combined to form an objective function and constraints. We combine all these elements in Section 4.6, where we describe the general form of a mathematical model and how it changes if the model under scrutiny is linear. In Section 4.7 we combine all the information provided and describe some examples of seminal models. In Section 4.8 we explain some special types of constraints, while in Section 4.9 we provide some final recommendations.

4.1 Sets

In a mathematical model, sets are, generally speaking, sequences of **distinct elements** (usually defined by a unique index) that play a role in the model itself. For example, if we are considering a problem where some trucks shall start/end their journey from/to some depots while servicing some customers during their trip, we will have to consider three sets, namely the set of trucks, the set of depots, and the set of customers. It is advised, although not mandatory, to use meaningful letters when defining sets. Using the aforementioned example, a good choice is to rely on the initials of the elements of each set and use $\mathcal{T}$ for the set of trucks, $\mathcal{D}$ for the set of depots, and $\mathcal{C}$ for the set of customers. In case more than one set starts with the same letter, a different choice has, of course, to be made.

Elements in a set are generally ordered. Another good policy is to use for the generic index the equivalent lower-case letter that was used to define the set. Using an enumeration that starts with 1, and assuming that our fleet of trucks encompasses 4 trucks, we can express our set of trucks as $\mathcal{T} = \{1, 2, 3, 4\}$. We can refer to a specific truck in the set using index t. For example, the expression $\forall t \in \mathcal{T}$ implies we are considering every truck t in our set $\mathcal{T}$. Let us complete our example defining 2 depots as part of set $\mathcal{D}$ and 9 customers as part of set $\mathcal{C}$. **Since both depots and customers define geographical locations that trucks can visit, we can combine them in a new set** $\mathcal{N}$, indexed by i or j (we will anticipate in the **Note: set indexing practices** box why two distinct letters are needed in this case. We will extensively rely on the same notation in Chapter 13), representing the nodes of our problem. To avoid duplicate indices, we could define indices so that depot indices precede customer indices. Namely, $\mathcal{D} = \{1, 2\}$ and $\mathcal{C} = \{3, \cdots, 11\}$, so that $\mathcal{N} = \{1, 2, \cdots, 10, 11\}$.

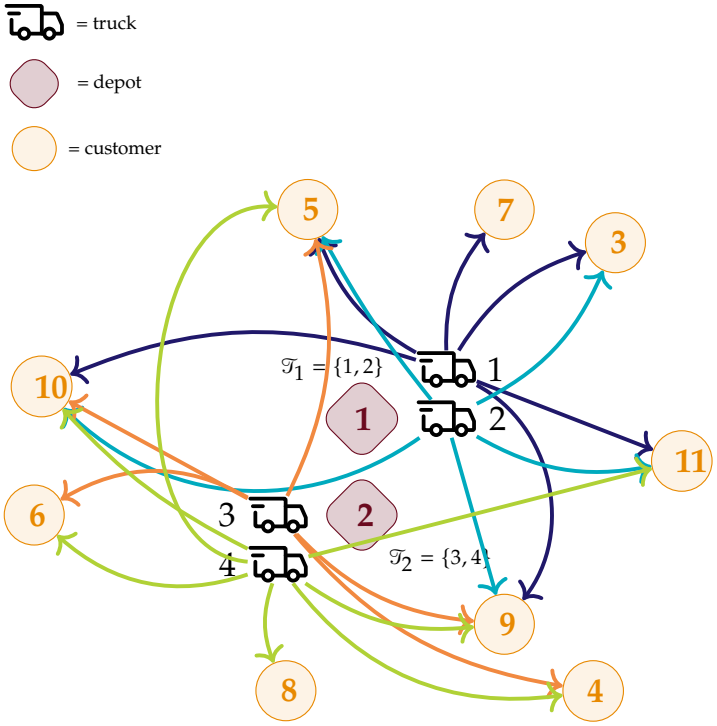


Figure 4.1: Example of definition of sets and subsets related to the example provided in Section 4.1.

Furthermore, we can use a combination of calligraphic letters and lower-case letters to define subsets. For example, we could define $\mathcal{T}_d$ as the subset of trucks that start/end their trip in depot d. If we assume that trucks 1 and 2 are linked to depot 1, while trucks 3 and 4 are linked to depot 2, we can define $\mathcal{T}_1 = \{1, 2\}$ and $\mathcal{T}_2 = \{3, 4\}$. Similarly, we can assume that each customer can only be served by a subset of trucks and define $\mathcal{T}_c$ as the subset of trucks that can visit customer c. For example, defining $\mathcal{T}_3 = \{1, 2\}$ we imply that customer 3 can only be visited by either truck 1 or truck 2. For the sake of completeness, let us define the remaining subsets as $\mathcal{T}_4 = \{3, 4\}$, $\mathcal{T}_5 = \{1, 2, 3, 4\}$, $\mathcal{T}_6 = \{3, 4\}$, $\mathcal{T}_7 = \{1\}$, $\mathcal{T}_8 = \{4\}$, $\mathcal{T}_9 = \{1, 2, 3, 4\}$, $\mathcal{T}_{10} = \{1, 2, 3, 4\}$, $\mathcal{T}_{11} = \{1, 2, 4\}$. Our definition implies that customers 7 and 8 are very picky, as they require to be serviced by a specific truck, while customers 5, 9, and 10 are very flexible as they can be serviced by any truck $t \in \mathcal{T}$. In Figure 4.1 we provide a visual representation of the example with trucks, depots,

and customers that we described above. To limit the text in the figure, we provide a more visual representation of subsets $\mathcal{T}_c$ by connecting each truck with the customers that can be serviced by it via a colored arc. As such, the origins of arcs pointing to customer c represent the trucks that can visit such node, i.e., $\mathcal{T}_c$.

Additionally, we provide some recommendations on indexing practices in the **Note: set indexing practices** box. Let us conclude this section by addressing the earlier point made in the box addressing the notation of customer set $\mathcal{C}$, where we could not designate a "first" customer since the customer with the smallest index was labeled *customer 3*. However, if a modeler prefers to assign increasing indices starting from 1 to customers, our notation can still be utilized within the model. Then, during the post-processing phase, the solution can be adjusted by subtracting 2 from every customer index, transforming $3 \rightarrow 1$, $4 \rightarrow 2$, and so forth, up to $11 \rightarrow 9$. Therefore, if our solution indicates that truck 1 should serve customer 3, it implies that this truck should attend to the first customer in our adjusted list.

Note: set indexing practices

Note that, when it comes to notation or other OR practices, we provide some guidelines in this book, not hard constraints. For example, considering the example of Figure 4.1, one could still define the indexing of the depots as $\mathcal{D} = \{1, 2\}$ and the indexing of the customers as $\mathcal{C} = \{1, \cdots, 9\}$. This approach works well for visualization purposes and provides a more direct mapping between a customer and the node associated with it (with our notation customer 3 is the first one, etc.) but it comes with at least two caveats:

- first, one could still differentiate between subsets $\mathcal{T}_d$ and $\mathcal{T}_c$ if written in this form. On the other hand, a term such as $\mathcal{T}_1$ (where we replaced the generic letter with an actual index) would create ambiguity, as that would represent both the trucks stationed in depot 1 ($\mathcal{T}_1$ if $d = 1$) and the trucks that can visit customer 1 ($\mathcal{T}_1$ if $c = 1$). **A way to circumvent this is to define the subsets as $\mathcal{T}_d^D$ and $\mathcal{T}_c^C$. The capital letters D and C are not indices but letters that explicitly define that the first subset refers to depots and the second to customers**;
- second, as we shall see later in this book, a decision variable in these problems can be generally written as x_{ij}^t. This decision variable is unitary if truck t moves from node i to node j in the network and 0 otherwise. With the original notation that we propose in the text, we have that $i \neq j$ for every pair of nodes in the network, hence every x_{ij}^t properly defines a movement from a node to a different one. With the notation suggested above, we would define the decision variable between depot 1 and customer 1 (two distinct nodes) as $x_{1,1}^t$. This looks like a decision variable from a node to itself, which might be cumbersome and does not reflect the nature of the decision variable. **Note that decision variables can still feature the same value for indices related to different sets. For example, $x_{1,3}^1$ defines, if unitary, that truck 1 travels from depot 1 to customer 3. Because of the way x_{ij}^t is defined (indices of nodes as subscripts and of trucks as superscripts), there is no ambiguity**.

When approaching mathematical modeling as beginners, familiarizing with good notation practices entails quite a steep learning curve!

4.2 Parameters

In a mathematical model, **parameters are known values defining specific characteristics of the problem at hand**. They can represent a distance, a cost, a revenue, a weight, a maximum capacity, among other things. Referring back to the example with trucks and customers, we could define D_{ij} as the distance between nodes $i, j \in \mathcal{N}$. Related to distance, R_t could represent the maximum range that truck $t \in \mathcal{T}$ can cover before having to return to the depot. Additionally, D_c could be the demand for goods (e.g., the overall weight) required by customer $c \in \mathcal{C}$. Note that we have two parameters that look quite similar: D_{ij} and D_c. It should be noted that the first one depends on two indices, being a distance, while the second one depends on one index as it is a customer-specific parameter. Hence, the chances of confusion should be slight. If readers are still not comfortable with the choice, then we could change the letter for the customer demand and use Q_c, or use DEM_c to make an even more explicit reference to the parameter being represented. As previously hinted at, there are good practices and standards when it comes to notation, but no golden rules. **A crucial recommendation is that when constructing a mathematical model, the notation should streamline the task for the modeler while remaining comprehensible to other users**.

4.3 Decision variables

In a mathematical model, decision variables are **unknown elements defining the problem under scrutiny. The ultimate goal of the modeler is to determine the optimal values for each decision variable and hence obtain the optimal solution**[1].

1: We will discuss and motivate the concept of optimality (already introduced in Chapter 1) in detail in Chapter 6 and Chapter 7. For now, we can picture the optimal solution as the set of actions and decisions that ensure our problem is addressed in the best way possible.

We can identify two main types of decision variables:

- **continuous** decision variables. They represent quantities that can take any (even fractional) value within the range where they are defined. For instance, determining the optimal arrival time at the airport to ensure reaching the gate on time while minimizing unnecessary waiting periods. Our solution could be 2.7 hours before the intended departure, hence a real number. Note that allowing decision variables to take fractional values does not mean they are allowed to take any value. We would all agree that arriving at the airport -1.5 hours before departure does not sound efficient, but arriving 20 hours before departure does not sound efficient either. Hence, continuous decision variables generally have a smallest allowed value (**lower bound**) and a maximum one (**upper bound**);
- **integer** decision variables. Sometimes, the decision we must take cannot be fractional. This is, for example, the case when an airline needs to decide how many aircraft of a specific type to order. The choice is bounded to be an integer number $\{0, 1, 2, \cdots\}$. A special type of integer decision variables are **binary** decision variables. As the name suggests, they only offer two options. Unless differently specified, the two options are 0 or 1. Binary decision variables are usually associated with a decision that takes the form of a **choice**: shall I purchase this product or not, shall I go from A to B or not?

Across all decision variables discussed in this book, a consistent characteristic is their **non-negativity**[2]. This aligns with the majority of real-world decisions, which typically involve positive values.

2: Models with negative decision variables can be created and solved. We refer interested readers to Hillier and Lieberman, 2015

Notation-wise, it is quite custom to choose x to designate a single set of decision variables for problems where only one set of decision variables exists, regardless of the actual type of decision. Routing problems feature binary decisions related to the possibility of going from a certain location i to another location j or not. In such problems, it is quite standard to use binary decision variable $x_{ij} \in \{0, 1\}$ to represent such an option. When a model features several distinct sets of decisions, it is good practice to choose meaningful letters. For example, a decision variable related to when to arrive at a location i can be defined t_i to capture that it is a decision about time.

4.4 Objective function

The objective function is the main KPI of a mathematical model. It is a function of the decision variables (not necessarily all of them) and of the parameters, which we can define in general terms as

$$Z = f(\overrightarrow{P}, \overrightarrow{x}) \tag{4.1}$$

where $\overrightarrow{P}$ represents the vector containing all parameters and $\overrightarrow{x}$ the vector containing all decision variables. $f(\cdot)$ represents a generic function, as the objective function could combine decision variables in different ways. In this book, we deal with models that are **linear or linearizable** because they can be efficiently solved with specific solution algorithms that will be described in Chapter 6, Chapter 7, and Chapter 8.

Because of this restriction, an objective function such as $Z = 3x_1 + 4x_2x_3$, displaying the product of two decision variables, will not be treated in this book. If we restrict our domain to linear objectives, then we can rewrite (4.1) as

$$Z = \overrightarrow{C}^T \cdot \overrightarrow{x} = \sum_{i=1}^{|\overrightarrow{x}|} C_i x_i \tag{4.2}$$

where the generic function $f(\cdot)$ has been replaced by a summation (linear operator). We provide two equivalent forms for the linear version of the objective function. The first one is based on the dot product we discussed in Chapter 3, while the second explicitly displays the summation. **In addition, note that we replaced vector $\overrightarrow{P}$ with vector $\overrightarrow{C}$ in (4.2). Because of the linear nature of the objective, now every decision variable can appear there multiplied by its specific coefficient**. Hence, for every x_i there exists a specific C_i. Conversely, in the non-linear example we showed, in term $4x_2x_3$ the coefficient 4 is associated with neither x_2 nor x_3, but with their product.

It is worth noting that (4.2) does not necessitate the inclusion of every decision variable in the objective. Depending on the specific model, some decision variables may be absent from the objective function altogether.

In scenarios where only a subset of decision variables is present in the objective, one might wonder about the role of the remaining variables. **In such cases, it is anticipated that these variables will appear in the constraints that define the model**.

Finally, we should remember that the objective function represents our main KPI and, as such, our goal is to steer such an objective towards the right direction. For objectives mapping "favorable" KPIs, it is in our interest the see them grow as much as possible. Some examples are profit, users' satisfaction, etc. For objectives mapping "unfavorable" KPIs, it is in our interest the see them reduce as much as possible. Some examples are costs, make-span, discomfort, etc. In the former case we deal with **maximization problems** while in the latter case we deal with **minimization problems**. In a mathematical model, we explicitly commit to one of the two cases by adding in front of the objective function the keywords max or min, respectively. For a maximization problem, we will then write

$$\max \; Z = \sum_{i=1}^{|\vec{x}|} C_i x_i \tag{4.3}$$

while a minimization problem would feature the following objective

$$\min \; Z = \sum_{i=1}^{|\vec{x}|} C_i x_i \tag{4.4}$$

In the rest of the book, when dealing with the general setting of a mathematical model, we will assume a maximization problem for the sake of simplicity. Some additional notes are:

- each mathematical model features an objective that, for practical purposes, can be unambiguously labeled as a maximization or minimization objective. **Mathematically speaking, we can always convert a** max **into a** min **problem (or vice versa) without hindering the final solution** as follows:

$$\max \; Z = \sum_{i=1}^{|\vec{x}|} C_i x_i \rightarrow \min \; -Z = -\sum_{i=1}^{|\vec{x}|} C_i x_i \tag{4.5}$$

  where (4.5) delineates how maximizing a certain objective function is equivalent to minimizing the opposite of such an objective (we will take more about this in Section 6.4, and more specifically in Example 6.7);
- in general, the letter Z is dropped from the definition of the objective function (albeit we shall see it can be useful in the context of the simplex method in Chapter 6). Hence, a general maximization problem is characterized by an objective expressed as $\max \; \sum_{i=1}^{|\vec{x}|} C_i x_i$
- while (4.3) generally captures the essence of the objective function, it is advisable to decompose the objective into its constituent terms tailored to the specific problem at hand. This can be achieved by

leveraging the sets, parameters, and decision variables outlined in Section 4.1, Section 4.2, and Section 4.3. For instance, **if the objective function represents profit, it typically consists of multiple terms, including a "positive" term representing revenue and a "negative" term accounting for costs, each possibly comprising further sub-terms**.

4.5 Constraints

Properly defining an objective function is of paramount importance for the correct implementation and solution of a mathematical model[3], but it is one side of the story. The other side is represented by the correct understanding and implementation of the constraints characterizing our problem.

3: While so far we talked about "generic" mathematical models, the models we address in this book are optimization models because it is our goal to look for the combination of decision variables that maximize or minimize our objective while satisfying all the constraints. We will formalize this concept in Chapter 6. In practice, we are looking for the best (optimal) solution given the inputs shaping the model.

Consider budget constraints that prevent unrealistic profit expectations by restricting investments beyond available funds. Similarly, time constraints mandate accounting for travel duration between points A and B, precluding instantaneous teleportation.

Moreover, constraints may encompass detailed specifications of the problem. For instance, a logistics firm may mandate single-truck visits per customer, while another company allows multiple trucks to serve the same customer by splitting orders for efficiency. The two different needs will be captured by different expressions for some constraints despite, for example, the objective of both companies being to maximize profit.

Another important constraint set addressed the need to ensure that each decision variable satisfies its own nature, as described in Section 4.3. When dealing with continuous variables, we assume that every value they take within the bounds we provided is feasible. When dealing with integer variables, we need to ensure the solution we come up with satisfies all integrality requirements. We will tackle this last issue in Chapter 7.

In analogy to what we did for the objective function in Section 4.4, we can start with two general definitions for the constraints characterizing a mathematical model as follows:

$$g(\overrightarrow{P}, \overrightarrow{x}) \leq \overrightarrow{b_{in}} \tag{4.6}$$

$$h(\overrightarrow{P}, \overrightarrow{x}) = \overrightarrow{b_{eq}} \tag{4.7}$$

4: Note that we use the $\leq$ form for the sake of simplicity, but we intend that both inequality constraints in the $\leq$ and $\geq$ form are captured.

where (4.6)[4] and (4.7) highlight, respectively, a set of inequality and equality constraints. $g(\cdot)$ and $h(\cdot)$ represent generic functions, so that constraints such as $2x_1 + 3x_2x_3 \geq 6$ or $x_2 + x_4x_5 = 6$ are allowed.

In the context of linear problems, we can replace (4.6) and (4.7) with their linear counterparts

$$A_{in}\,\overrightarrow{x} \leq \overrightarrow{b_{in}} \tag{4.8}$$
$$A_{eq}\,\overrightarrow{x} = \overrightarrow{b_{eq}} \tag{4.9}$$

where A_{in} and A_{eq} are matrices[5] with as many columns as the number of decision variables and as many rows as how many inequality (resp. equality) constraints are needed.

Finally, let us split our vector of decision variables $\overrightarrow{x}$ into 3 mutually exclusive subsets, namely $\overrightarrow{x_c}$, $\overrightarrow{x_i}$, and $\overrightarrow{x_b}$ (representing respectively continuous, integer, and binary decision variables). In our model, we will have to ensure that

$$\overrightarrow{x_c} \in \mathbb{R}_0 \tag{4.10}$$
$$\overrightarrow{x_i} \in \mathbb{N}_0 \tag{4.11}$$
$$\overrightarrow{x_b} \in \{0, 1\} \tag{4.12}$$

where $\mathbb{R}_0$ and $\mathbb{N}_0$ represent the set of non-negative real and integer numbers respectively.

5: In some other references, readers might stumble upon a single expression in the form $A\overrightarrow{x} \leq \overrightarrow{b}$ representing the whole set of constraints. Because this is a general form, of course, some details of the actual model are lost anyway. We preferred to keep separate the two blocks describing inequality and equality constraints instead.

4.6 General form of a mathematical model

We can now condense the knowledge acquired in Section 4.4 and Section 4.5[6] to fully define a mathematical model as follows:

$$\max \; Z = \sum_{i=1}^{|\overrightarrow{x}|} C_i x_i \tag{4.13}$$

s.t.:[7]

$$A_{in}\,\overrightarrow{x} \leq \overrightarrow{b_{in}} \tag{4.14}$$
$$A_{eq}\,\overrightarrow{x} = \overrightarrow{b_{eq}} \tag{4.15}$$
$$\overrightarrow{x_c} \in \mathbb{R}_0 \tag{4.16}$$
$$\overrightarrow{x_i} \in \mathbb{N}_0 \tag{4.17}$$
$$\overrightarrow{x_b} \in \{0, 1\} \tag{4.18}$$

where (4.13) defines the objective, (4.14) and (4.15) the inequality and equality constraints, respectively. Finally, (4.16)-(4.18) ensure that decision variables are of the proper type. **Note that this formulation is very generic. Hence, some models might only feature continuous or integer or binary decision variables, making some of (4.16)-(4.18) redundant. Additionally, some models might only feature inequality or equality constraints.**

6: Note that, given the general setting, we still do not fully leverage the notion of sets and the suggested notation defined earlier in the chapter. We will fully adopt the suggested notation when presenting specific problems where sets, parameters, and decision variables will be explicitly presented.

7: With s.t. we mean *subject to*.

In the rest of the book, for the sake of simplicity, we will drop the vector symbol $\overrightarrow{(\cdot)}$ from above the decision variable vector $\overrightarrow{x}$ which will become just x. Similarly, with C we mean the coefficient vector $\overrightarrow{C}$. Because we are dropping the vector symbol, we will also drop the dot product symbol $\cdot$ and represent a generic objective function of an LP as $C^T x$ if needed. Furthermore, for continuous variables we will be using interchangeably the expressions $\in \mathbb{R}_0$ and ≥ 0 as they both represent the set of non-negative real numbers.

4.7 Construction of a mathematical model

In this section, we experiment with the knowledge acquired and translate a few practical problems into mathematical models. Typically, the following steps should be undertaken:

1. understand the practical problem[8];
2. identify sets, parameters, and decision variables;
3. build the objective function;
4. build the constraints;
5. assemble the full mathematical model.

8: Otherwise, we might formulate a correct mathematical model, which however does not represent the original practical problem. On a similar note, albeit this requires more experience, it is also important to assess which simplifications are allowed and which are not when translating a practical problem into a model (the latter is anyway a simplification of reality).

We showcase this systematic approach in Example 4.1-Example 4.3.

Example 4.1 *A company produces 2 types of fertilizer: L-fert and H-fert. Both fertilizers need 3 raw materials to be produced: A, B, and C. For the coming month, the company has got 1,500 tons of A, 1,200 tons of B, and 500 tons of C. To produce 1 ton of L-fert, 2 tons of A, 1 ton of B, and 1 ton of C are needed. To produce 1 ton of H-fert, 1 ton of A, 1 ton of B, and no raw material C are needed. For each ton sold of L-fert the company earns 50€ whereas for each ton of H-fert sold 15€. Build a mathematical model for this problem with profit maximization as the objective.*

Let us focus on the sets first. We have fertilizers and raw materials. We define $\mathcal{F}$ as the set of fertilizers (which will be indexed by f) and $\mathcal{M}$ as the set of materials (which will be indexed by m). Because working with numbers is easier, we write $\mathcal{F} = \{1, 2\}$ and $\mathcal{M} = \{1, 2, 3\}$ implying that L-fert $\rightarrow$ 1 and H-fert $\rightarrow$ 2 in $\mathcal{F}$ and A $\rightarrow$ 1, B $\rightarrow$ 2, and C $\rightarrow$ 3 in $\mathcal{M}$.

In terms of parameters, we have three main types of parameters. First, we have a maximum supply per material, which we define S_m. Hence, $S_1 = 1,500$, $S_2 = 1,200$, and $S_3 = 500$. Second, we have a specific quantity of each material m needed for the production of 1 ton of fertilizer f. We can define this parameter Q_{mf}. **Note that it features two indices m and f because it is a parameter associated with a specific material-fertilizer combination**. In our case, we can write $Q_{1,1} = 2$, $Q_{2,1} = 2$, $Q_{3,1} = 1$, $Q_{1,2} = 1$, $Q_{2,2} = 1$, and $Q_{3,2} = 0$. Finally, we have the revenue associated with 1 ton sold of a specific fertilizer, which we label R_f. In our case, $R_1 = 50$ and $R_2 = 15$.

We now proceed with the definition of the decision variables. We need to decide how many tonnes to produce for both fertilizers, hence we can define x_1 and x_2 as, respectively, the tonnes produced for fertilizer 1 (L-fert) and 2 (H-fert). The general form of this set of decision variables is $x_f \; \forall f \in \mathcal{F}$. Because we are not given any constraint regarding how much

to produce, we assume that fractional values are allowed and hence will treat x_1 and x_2 as continuous decision variables.

The exercise states that the objective is to maximize the profit. Hence an adequate objective function is

$$\max \sum_{f \in \mathcal{F}} R_f x_f \tag{4.19}$$

where we leverage the conciseness of the introduced mathematical notation. With $\sum_{f \in \mathcal{F}}$ we imply that the objective is the summation of all the terms $R_f x_f$ part of our model. **Note that the expression is unchanged if we deal with 2 fertilizers like in this case ($|\mathcal{F}| = 2$), or 2,000. This compact representation offers inherent advantages**. Each term in our objective maps revenue generated from the sale of fertilizer f. Notably, we exclude any incurred costs from consideration. Thus, in this context, profit equates to revenue.

Lastly, constraints must be addressed. We are constrained by the limited supply of each of the 3 materials, meaning we cannot indefinitely produce both fertilizers. Therefore, our production of L-fert and H-fert can vary, provided the combination complies with the available supply of materials. As a consequence, it feels right to write one constraint per material m. The left-hand side will map how much of that material is used to produce both fertilizers and should be less or equal to the right-hand side containing the available supply S_m. In compact form, we can write

$$\sum_{f \in \mathcal{F}} Q_{mf} x_f \leq S_m \qquad \forall m \in \mathcal{M} \tag{4.20}$$

where the term $\forall m \in \mathcal{M}$ defines that we need to write such a constraint for every material. Given a specific material $m \in \mathcal{M}$, $\sum_{f \in \mathcal{F}} Q_{mf} x_f$ defines the overall amount (in tonnes) of the material that we need to produce all the fertilizers and such amount should be less or equal to the available supply S_m. For material A, for example, we would write $2x_1 + x_2 \leq 1,500$. This means that 750 tonnes of fertilizer $f = 1$, 1,500 tonnes of fertilizer $f = 2$, or any other combination that does not exceed the available 1,500 tonnes of material $m = 1$ can be produced.

We can now assemble all the pieces of the puzzle to obtain our full model

$$\max \sum_{f \in \mathcal{F}} R_f x_f \tag{4.21}$$

s.t.:

$$\sum_{f \in \mathcal{F}} Q_{mf} x_f \leq S_m \qquad \forall m \in \mathcal{M} \tag{4.22}$$

$$x_f \in \mathbb{R}_0 \qquad \forall f \in \mathcal{F} \tag{4.23}$$

Because of the limited number of decision variables and constraints, for the sake of completeness, we also provide the extended mathematical formulation

$$\max \quad 50x_1 + 15x_2 \tag{4.24}$$

s.t.:

$$2x_1 + x_2 \leq 1,500 \tag{4.25}$$
$$x_1 + x_2 \leq 1,200 \tag{4.26}$$
$$x_1 \leq 500 \tag{4.27}$$
$$x_1, x_2 \in \mathbb{R}_0 \tag{4.28}$$

Example 4.2 *A traveler is preparing for a weekend getaway to the picturesque island of Texel in the Netherlands (see Figure 4.2). Alongside a larger backpack filled with essentials like clothes and toiletries, they have a smaller backpack with a strict weight limit of 3 kg. As an avid reader, they face the dilemma of selecting which of the following six books to pack: "The Hitchhiker's Guide to the Galaxy" by Douglas Adams, "Ten Little Indians" by Agatha Christie, "Nineteen Eighty-Four" by George Orwell, "Too Many Cooks" by Rex Stout, "Neuromancer" by William Gibson, and "The Name of the Rose" by Umberto Eco. These books weigh 1.1, 0.8, 0.6, 0.9, 1.0, and 1.2 kg, respectively. Additionally, the traveler has assigned each book a sentimental value based on their enjoyment, rating them as 4.8, 4.6, 4.9, 4.2, 4.7, and 4.8, respectively. However, they have come to the realization that they cannot carry all six books in their small backpack. Thus, our task is to assist them by formulating a mathematical model to maximize the total value of the books they carry.*



Figure 4.2: The iconic lighthouse of Texel.

We have a set of books $\mathcal{B} = \{1, 2, 3, 4, 5, 6\}$ indexed by b where the ordering follows the same order in which books were listed (hence, *"The Hitchhiker's Guide to the Galaxy"* is book 1, etc.). The larger backpack is irrelevant to our model, so we only need to consider the smaller backpack. While we could define a set solely for the backpack in question, it is not essential. **It is worth noting that our objective is to determine which books to pack in the backpack, not in which backpack to place them**.

Each book b comes with two parameters, namely its weight and value. We could label them, respectively, W_b and V_b. In addition, the backpack has a maximum weight capacity of C. **Because there is only one backpack, we do not need to assign any index to this parameter**.

Finally, our only decision variable regards whether to pack book b in the backpack or not. Given it is a choice, a binary decision variable seems the appropriate decision variable type: $x_b \in \{0, 1\} \forall b \in \mathcal{B}$ will take a unitary value if book b travels with the traveler in the backpack to Texel and a zero value otherwise.

We now combine parameters and decision variables to determine the objective value and the constraints. To maximize the carried value, we can write max $\sum_{b \in \mathcal{B}} V_b x_b$ which correctly adds the value V_b of book b if such a book is packed ($x_b = 1$). Having one backpack, we need a single

constraint in the form $\sum_{b\in\mathcal{B}} W_b x_b \leq C$. We can now assemble the full model as:

$$\max \sum_{b\in\mathcal{B}} V_b x_b \tag{4.29}$$

s.t.:

$$\sum_{b\in\mathcal{B}} W_b x_b \leq C \tag{4.30}$$

$$x_b \in \{0,1\} \qquad \forall b \in \mathcal{B} \tag{4.31}$$

Note that a good policy is to ensure the objective function and the constraints are consistent unit-wise. For example, let us consider (4.30), the left-hand side is a summation of weights in kilograms (each x_b is adimensional), which is consistent with the right-hand side. For the sake of completeness, we expand this model as well:

$$\max \; 4.8x_1 + 4,6x_2 + 4.9x_3 + 4.2x_4 + 4.7x_5 + 4.8x_6 \tag{4.32}$$

s.t.:

$$1.1x_1 + 0.8x_2 + 0.6x_3 + 0.9x_4 + 1.0x_5 + 1.2x_6 \leq 3 \tag{4.33}$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \in \{0,1\} \tag{4.34}$$

Because Example 4.2 focused on books, Example 4.3 will feature a plot twist[9].

9: See this Wikipedia page for a proper definition and different types of plot twists in books and movies.

Example 4.3 *Just before leaving for Texel, our traveler has acquired two additional small backpacks with capacities of 2.2 and 2.3 kg, respectively. They have opted to discard the original 3 kg backpack in favor of using these two. Given their preference for categorizing books, they have decided not to mix science fiction books ("The Hitchhiker's Guide to the Galaxy", "Nineteen Eighty-Four", and "Neuromancer") and mystery books ("Ten Little Indians", "Too Many Cooks", and "The Name of the Rose") in the same backpack. Our revised task is to adjust the model from Example 4.2 to accommodate these new conditions while maintaining the same objective.*

We first need to make some adjustments to our sets. We could split set $\mathcal{B}$ into the two subsets $\mathcal{B}_s$ and $\mathcal{B}_m$ containing, respectively, science fiction and mystery books. Inheriting the indexing from Example 4.2, we write $\mathcal{B}_s = \{1,3,5\}$ and $\mathcal{B}_m = \{2,4,6\}$. Note that the two subsets are mutually exclusive: $\mathcal{B}_s \cap \mathcal{B}_m = \emptyset$. Because a new requirement mandates that no books of different genres should be placed in the same backpack, we could define a set with all the incompatible pairs: $\mathcal{B}_{inc} = \{(1,2),(1,4),(1,6),(3,2),(3,4),(3,6),(5,2),(5,4),(5,6)\}$. We assembled $\mathcal{B}_{inc}$ by constructing book pairs (b_1, b_2) with $b_1 \in \mathcal{B}_s$ and $b_2 \in \mathcal{B}_m$. **We do not need to write the reciprocal pairs, because saying that books**

1 and 2 are incompatible is equivalent to saying that books 2 and 1 are incompatible. Because now we have two backpacks, we define set $\mathcal{K} = \{1, 2\}$ (we chose $\mathcal{K}$ instead of the more natural $\mathcal{B}$ that would have reflected the initial letter of *backpack* as set $\mathcal{B}$ was already defined).

We must also update our capacity parameter, now denoted as C_k, to accommodate the presence of two backpacks with different weight capacities. Furthermore, our decision variables require adjustment to reflect the increased complexity. We must now decide not only which books to carry but also in which backpack to place them. We define $x_{bk} \in \{0, 1\} \forall b \in \mathcal{B}, k \in \mathcal{K}$ a binary decision variable that is unitary if the place book b in backpack k and zero otherwise. Note that now we have 12 decision variables ($|\mathcal{B}| \times |\mathcal{K}| = 6 \times 2 = 12$) instead of the 6 from Example 4.2 because the choice of which backpack to select increases the number of options.

We directly provide the mathematical formulation and then discuss the variations with respect to Example 4.2. The model is as follows:

$$\max \sum_{b \in \mathcal{B}} \sum_{k \in \mathcal{K}} V_b x_{bk} \tag{4.35}$$

s.t.:

$$\sum_{k \in \mathcal{K}} x_{bk} \leq 1 \qquad \forall b \in \mathcal{B} \tag{4.36}$$

$$\sum_{b \in \mathcal{B}} W_b x_b \leq C_k \qquad \forall k \in \mathcal{K} \tag{4.37}$$

$$x_{b_1 k} + x_{b_2 k} \leq 1 \qquad \forall (b_1, b_2) \in \mathcal{B}_{inc}, k \in \mathcal{K} \tag{4.38}$$

$$x_{bk} \in \{0, 1\} \qquad \forall b \in \mathcal{B}, k \in \mathcal{K} \tag{4.39}$$

(4.35) features a double summation instead of the single summation of (4.29). This is correct because now we must consider that books must be assigned to a specific backpack. Moreover, considering the decision variable x_{bk}, if we were to solely consider the summation $\sum_{b \in \mathcal{B}}$, we would encounter difficulties in determining the appropriate value for k. This should prompt a cautionary note.

Constraint set (4.36) presents a new constraint essential for ensuring that each book is allocated to at most one backpack. **Although in practice, we would never breach this constraint due to the impossibility of duplicating a book, it is crucial to explicitly inform the model of this restriction to prevent inadvertent assignment of the same book to multiple backpacks**. Constraint (4.37) replicates the constraint in (4.30), but now it is formulated for each backpack individually to ensure their respective capacities are not exceeded.

Additionally, (4.38) introduces a novel constraint to prevent incompatible books from coexisting in the same backpack. For every pair of incompatible books ($\forall (b_1, b_2) \in \mathcal{B}_{inc}$) and backpack, the constraint ensures that at most one book from the pair is assigned to that backpack. The use of the $\leq$ inequality allows for none of the books to be assigned to the backpack if it is beneficial for the solution. **In general, constraint (4.38)**

is a very important constraint in many models and is a variant of the either-or (see Section 4.8.2) constraint, as it forces a maximum of one choice between two incompatible options. Finally, (4.39) defines the binary nature of the decision variables and states that they are defined for every book and backpack combination.

In this instance, we opt not to present the extended formulation due to its extensive nature, encompassing nearly twenty constraints. The succinct representation of the mathematical model, although requiring some familiarity, demonstrates its brevity and effectiveness in delineating all the pertinent sets, subsets, and parameters, a task that the extended formulation would struggle to achieve.

4.8 Special types of constraints

4.8.1 Big-M notation

When constructing a mathematical model, there are instances where we need to enable a certain decision variable to be greater than zero only if another variable is activated, or activate only one constraint among a set of two. To accomplish this, we utilize an auxiliary decision variable that serves as a trigger for the decision-making process. This decision variable, typically binary, is coupled with a "sufficiently large" constant[10] generally denoted as M symbolically.

10: The term big-M denotes that this constant should be large enough to ensure the model achieves its goal. Notwithstanding, big-M should not be excessively large for algorithmic reasons related to the solution methods explained in Chapter 6. While choosing a larger-than-necessary big-M still serves its mathematical purposes, it generally makes the algorithm slower.

Example 4.4 Consider a scenario where we must choose between enforcing the constraint $x_1 \leq 4$ and $x_1 \geq 7$. This situation might arise in a production process where, depending on certain investments governed by other decision variables, we can produce a maximum of either 4 or a minimum of 7 items (with x_1 representing this decision variable). However, directly adding both constraints would lead to infeasibility, as the two constraints are in conflict with each other. To address this, we employ the big-M formulation, albeit at the expense of introducing an additional decision variable.

Let us showcase the formulation first and then analyze what it achieves. The formulation is:

$$x_1 \leq 4 + M(1 - y) \tag{4.40}$$
$$x_1 \geq 7 - My \tag{4.41}$$
$$x_1 \in \mathbb{R}_0 \tag{4.42}$$
$$y \in \{0, 1\} \tag{4.43}$$

where $y \in \{0, 1\}$ is the additional decision variable. Because the model must assign a value to y, we can explore the two scenarios. If $y = 0$ we obtain:

$$x_1 \leq 4 + M \simeq \infty \tag{4.44}$$
$$x_1 \geq 7 \tag{4.45}$$
$$x_1 \in \mathbb{R}_0 \tag{4.46}$$
$$\tag{4.47}$$

whereas if $y = 1$:

$$x_1 \leq 4 \tag{4.48}$$
$$x_1 \geq 7 - M \simeq -\infty \tag{4.49}$$
$$x_1 \in \mathbb{R}_0 \tag{4.50}$$
$$\tag{4.51}$$

Hence, in the first case, we leverage M and the choice $y = 0$ to de-activate (4.44) and use (4.45) as the active constraint[11]. Note that choosing the right value for M is key. $M \simeq \infty$ would do the job as it imposes $x_1 \leq \infty$, but we mentioned it is wise to choose, if possible, the smallest value that achieves the intended goal. In the second case, we leverage M and the choice $y = 1$ to de-activate (4.49) and use (4.48) as the active constraint. Note that in (4.41) we used $-My$ so that, if $y = 1$, (4.49) yields $x_1 \geq -\infty$ which is always verified (**for us, a constraint that is "always verified" is equivalent to de-activting such a constraint**).

11: In other references, the active constraint might be defined as the *binding constraint*, with the de-activated one defined as the *non-binding constraint* or *redundant constraint*.

We shed some light on the sign of big-Ms according to the specific constraint in the **A note on the sign of big-M in inequality constraints** box.

A note on the sign of big-M in inequality constraints

Generally speaking, M is used with a plus sign in $\leq$ inequality constraints and with a minus sign in $\geq$ constraints. With such a choice, we set the right-hand side of an $\leq$ constraint to ∞ so that the constraint is always satisfied, while we set the right-hand side of an $\geq$ constraint to $-\infty$ so that the constraint is always satisfied as well. **Again, in practical terms, we do not need ∞ or $-\infty$, but the smallest M that achieves the intended goal**.

Example 4.4 was mainly focused on the introduction of the concept of the big-M formulation and its main role in activating/de-activating constraints. In the rest of this section and of the book, we will appreciate even more its widespread use, starting with the either-or constraint type presented in Section 4.8.2 (which was also the type of constraint we used for our introductory example above).

4.8.2 Either-or constraints

An **either-or** constraint[12], as hinted at with Example 4.4 in Section 4.8.1, is a constraint type where one alternative out of two options available, as the name suggests, must be picked. *Alternative* can mean a single

12: In the literature, either-or constraints might be referred to as bi-linear formulations.

constraint, as in Example 4.4, or a set of constraints that should be active altogether, with the other set being de-activated. The activation and de-activation are carried out by the additional binary variable y we discussed above. We consider now an example characterized by a set of constraints representing each alternative.

Example 4.5 Let us consider a mathematical model based on two decision variables x_1 and x_2 whose objective is the minimization of their linear combination: $\min Z = C_1x_1 + C_2x_2$ (where C_1 and C_2 are pre-defined coefficients). Additionally, let us assume that the problem can be defined in one of the following two feasible regions: $\mathcal{F}_1$ where $2 \leq x_1 \leq 4$ and $4 \leq x_2 \leq 6$ and $\mathcal{F}_2$ where $5 \leq x_1 \leq 8$ and $2 \leq x_2 \leq 3$. Depending on the actual values of C_1 and C_2, choosing $\mathcal{F}_1$ and $\mathcal{F}_2$ as the active feasible region is recommended given the objective specified above. For example, if $(C_1, C_2) = (1, 1)$ then choosing $\mathcal{F}_1$ is the better choice with $(x_1, x_2) = (2, 4) \rightarrow Z = x_1 + x_2 = 6$ being the optimal solution and objective. Conversely, if $(C_1, C_2) = (1, 4)$ then choosing $\mathcal{F}_2$ is the better choice with $(x_1, x_2) = (5, 2) \rightarrow Z = x_1 + 4x_2 = 13$ being the optimal solution and objective respectively. We display the situation with the two preferred feasible regions and optimal solutions according to the (C_1, C_2) choice in Figure 4.3. For interested readers, the two solutions can be computed graphically as explained in Section 6.1.

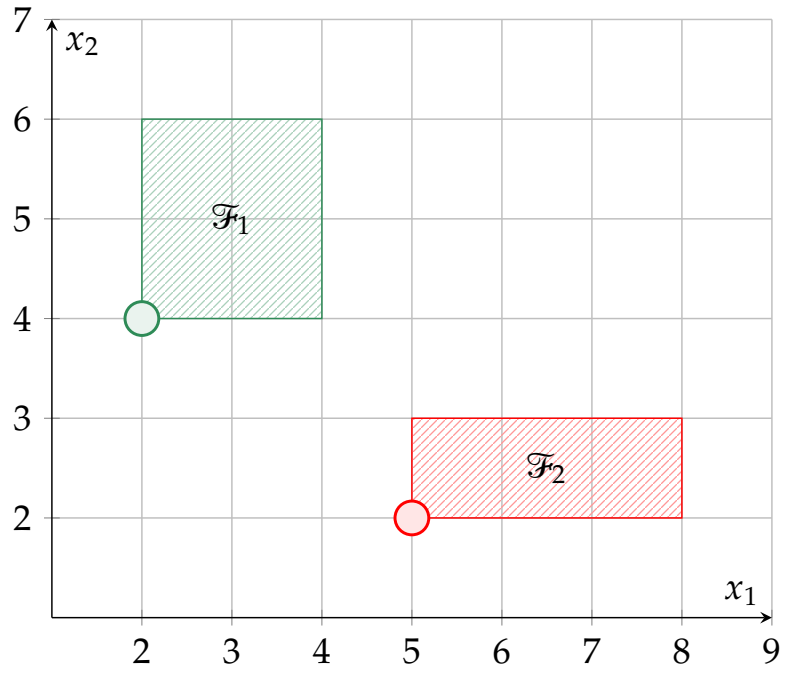


Figure 4.3: Feasible regions $\mathcal{F}_1$ (in green) and $\mathcal{F}_2$ (in red) for the problem described in Example 4.5. For the cases $(C_1, C_2) = (1, 1)$ and $(C_1, C_2) = (1, 4)$, the optimal solution is highlighted with a circle: it is respectively $(x_1, x_2) = (2, 4) \rightarrow Z = x_1 + x_2 = 6$ as part of $\mathcal{F}_1$ and $(x_1, x_2) = (5, 2) \rightarrow Z = x_1 + 4x_2 = 13$ as part of $\mathcal{F}_2$.

Our objective is to refine our mathematical model to automatically select the optimal feasible region while de-activating the alternate one, based on the specific coefficients (C_1, C_2) chosen. We achieve such an objective by leveraging the properties of the either-or constraint as follows:

$$\min \; C_1x_1 + C_2x_2 \tag{4.52}$$

s.t.:

$$x_1 \geq 2 - M_1(1 - y) \tag{4.53}$$
$$x_1 \leq 4 + M_2(1 - y) \tag{4.54}$$
$$x_2 \geq 4 - M_3(1 - y) \tag{4.55}$$
$$x_2 \leq 6 + M_4(1 - y) \tag{4.56}$$
$$x_1 \geq 5 - M_5 y \tag{4.57}$$
$$x_1 \leq 8 + M_6 y \tag{4.58}$$
$$x_2 \geq 2 - M_7 y \tag{4.59}$$
$$x_2 \leq 3 + M_8 y \tag{4.60}$$
$$x_1, x_2 \in \mathbb{R}_0 \tag{4.61}$$

If $y = 1$, we select $\mathcal{F}_1$ as the active feasible region by activating (4.53)-(4.56) and de-activating (4.57)-(4.60). If $y = 0$, we select $\mathcal{F}_2$ as the active feasible region by activating (4.57)-(4.60) and de-activating (4.53)-(4.56).

Readers will observe that we have precisely labeled the 8 Ms with distinct indices M_1 through M_8. This choice serves to underscore our earlier assertion that each M need not be set to an exceedingly large value, but rather to the smallest value necessary to de-activate the associated constraint. One primary consideration is that each M only influences the solution when it is multiplied by a non-zero value. For instance, in (4.54), M_2 is significant only if $y = 0$. If $y = 0$, the objective is to activate $\mathcal{F}_2$, implying the necessity to satisfy the condition $x_1 \leq 8$ (as in (4.58)). By assigning $M_2 = 4$, (4.54) transforms into $x_1 \leq 4 + 4(1 - y) = 8$ when $y = 0$, thus harmonizing with (4.58). Any value smaller than 4 for M_2 would lead to failure in activating $\mathcal{F}_2$ accurately. For instance, setting $M_2 = 3$ yields $x_1 \leq 7$ from (4.54), resulting in a vertical truncation of $\mathcal{F}_2$. Conversely, any value exceeding 4 for M_2 would serve the purpose, yet an excessively large number (e.g., $M_2 = 1,000,000$) might adversely affect the algorithm's performance in seeking the optimal solution within the feasible region (refer to Chapter 6)[13].

13: We refer interested readers to *OR in an OB World* (2024) for a thorough explanation of the algorithmic role of big-M in mathematical modeling. We also encourage readers to go through Chapters 6-7 before consulting the suggested reference.

For the sake of completeness, we report here the "smallest" values for the eight big-Ms in (4.53)-(4.60): $M_1 = 0$, $M_2 = 4$, $M_3 = 2$, $M_4 = 0$, $M_5 = 3$, $M_6 = 0$, $M_7 = 0$, and $M_8 = 3$. **Readers might notice that four values were set to zero. This deliberate decision stems from the recognition that no big-M value is necessary for these instances**. Our attention is drawn specifically to Equation 4.53, and we encourage readers to validate the other three scenarios. When $y = 1$, the model activates $\mathcal{F}_1$, necessitating $x_1 \geq 2$. Conversely, when $y = 0$, $\mathcal{F}_2$ is activated, requiring $x_1 \geq 5$. Notably, the latter inequality imposes a more stringent condition than $x_1 \geq 2$. Consequently, in $x_1 \geq 2 - M_1$, there is no need to further diminish the original right-hand side of 2 to avoid conflicting with $x_1 \geq 5$. This permits us to set $M_1 = 0$.

In conclusion, we showed how either-or decisions in a mathematical model can be described with this specific constraint type that entails the addition of a binary decision variable. Such a variable acts as a switch deciding which condition to activate and which to de-activate. We also showed that a condition could be a single constraint or a set of constraints that should hold altogether. Finally, we showed that the big-M coefficients we need for the de-activation do not need to be set

to ∞, but can be properly bounded by analyzing the properties of the considered model.

4.8.3 K-out-of-N constraints

For the next special constraint type, i.e., the ***K*-out-of-*N***, we directly dive into Example 4.6.

Example 4.6 Let us consider the same situation as in Example 4.5, but let us add a third candidate feasible region $\mathcal{F}_3$ where $\frac{5}{2} \leq x_1 \leq 4$ and $\frac{5}{2} \leq x_2 \leq \frac{7}{2}$. We display the new scenario (without the optimal solutions for different (C_1, C_2) values) in Figure 4.4.

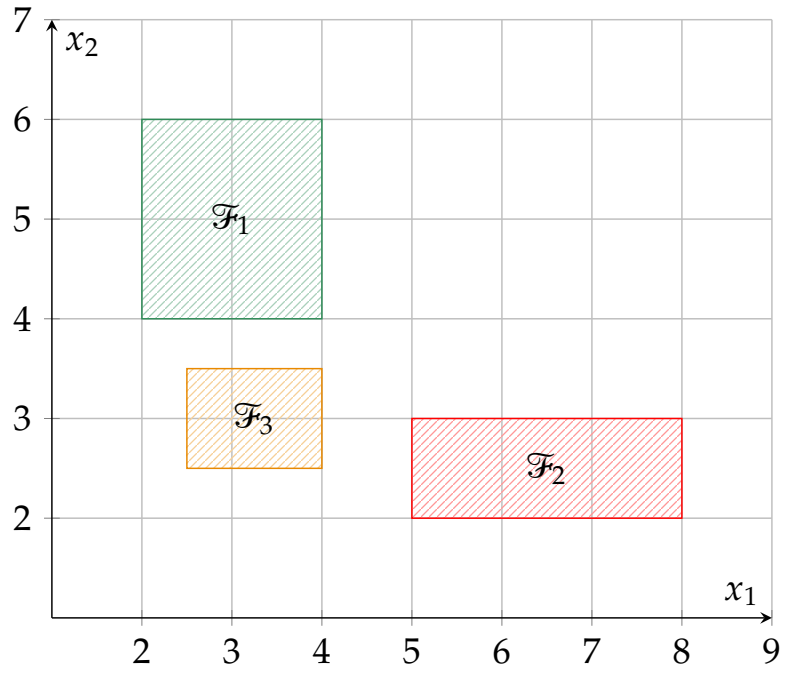


Figure 4.4: Feasible regions $\mathcal{F}_1$ (in green), $\mathcal{F}_2$ (in red), and $\mathcal{F}_3$ (in orange) for the problem described in Example 4.6.

Our goal remains the same. Given a specific (C_1, C_2) pair of coefficients in (4.52), we want our mathematical model to discern and select the suitable feasible region that minimizes the objective function. Alas, despite the knowledge acquired in Section 4.8.2, we realize an unforeseen complication is hindering our planned modeling approach. Because now we have three options to choose from and only one to activate, a single binary variable y is not enough. When two options are available, we can employ y as a switch that activates the option (i.e., the constraints) where the big-M terms disappear and de-activates the option where the big-M terms are triggered instead.

Luckily, we can get inspiration from the name of this constraint type, i.e., *K-out-of-N*, and realize we are in a situation where we would like to activate one constraint set ($K = 1$) out of the three available ($N = 3$). Hence, we need three switches (binary decision variables) instead of one so that the model can turn them on and off in the best way possible according to the specific (C_1, C_2) values in this case. We can label the three variables y_1, y_2, and y_3 and impose that if a certain variable is set to 1, then the associated constraint set is active. Then, because we require exactly one constraint set out of the three to be active in our problem

(K-out-of-$N \rightarrow$ 1-out-of-3), we need constraint $y_1 + y_2 + y_3 = 1$ to impose such a requirement.

We can model our problem as follows:

$$\min \; C_1 x_1 + C_2 x_2 \tag{4.62}$$

s.t.:

$$x_1 \geq 2 - M(1 - y_1) \tag{4.63}$$
$$x_1 \leq 4 + M(1 - y_1) \tag{4.64}$$
$$x_2 \geq 4 - M(1 - y_1) \tag{4.65}$$
$$x_2 \leq 6 + M(1 - y_1) \tag{4.66}$$
$$x_1 \geq 5 - M(1 - y_2) \tag{4.67}$$
$$x_1 \leq 8 + M(1 - y_2) \tag{4.68}$$
$$x_2 \geq 2 - M(1 - y_2) \tag{4.69}$$
$$x_2 \leq 3 + M(1 - y_2) \tag{4.70}$$
$$x_1 \geq \frac{5}{2} - M(1 - y_3) \tag{4.71}$$
$$x_1 \leq 4 + M(1 - y_3) \tag{4.72}$$
$$x_2 \geq \frac{5}{2} - M(1 - y_3) \tag{4.73}$$
$$x_2 \leq \frac{7}{2} + M(1 - y_3) \tag{4.74}$$
$$y_1 + y_2 + y_3 = 1 \tag{4.75}$$
$$x_1, x_2 \in \mathbb{R}_0 \tag{4.76}$$
$$y_1, y_2, y_3 \in \{0, 1\} \tag{4.77}$$

Because of (4.75), only one of the three constraint sets (4.63)-(4.66), (4.67)-(4.70), and (4.71)-(4.74) will be active with all the M on the right-hand side disapperaring. The other two sets, whose y will be set to 0, will be redundant. Note that, because of the larger size of the problem at hand, we left a generic M instead of customizing them so that they are as small as needed. We leave this additional exercise to interested readers.

We also want to point out that Example 4.6 addresses a specific combination of (K, N) values(specifically $(1, 3)$), but that the general version of the K-out-of-N constraint can be generalized as follows. Let us consider $\mathcal{S}$ groups of constraints indexed by s, where $|\mathcal{S}| = N$ defines the overall number of such groups. Each $s \in \mathcal{S}$ can potentially contain a different number of constraints (many K-out-of-N constraints feature a single constraint per set s): we store the indices i of the rows of the A_{in} coefficient matrix forming this set $\mathcal{R}_s$. We then define each constraint set $s \in \mathcal{S}$ as $\sum_{j \in x} A_{ij} x_j \leq b_i \;\; \forall i \in \mathcal{R}_s$, where with $\sum_{j \in x}$ we imply that we sum over every decision variable. Note that we employ a generic expression $\leq$, but in each constraint set the actual constraints can appear in $\leq$, $\geq$, or (less commonly) $=$ form. We generalize (assuming a max problem) the K-out-of-N constraint as follows:

$$\max \; C^T x \tag{4.78}$$

s.t.:

$$\sum_{j \in x} A_{ij} x_j \leq b_i + M(1 - y_1) \qquad \forall i \in \mathcal{R}_1 \tag{4.79}$$

$$\dots$$

$$\sum_{j \in x} A_{ij} x_j \leq b_i + M(1 - y_N) \qquad \forall i \in \mathcal{R}_N \tag{4.80}$$

$$\sum_{s \in \mathcal{S}} y_s = K \tag{4.81}$$

$$x_j \in \mathbb{R}_0 \qquad \forall j \in x \tag{4.82}$$

$$y_s \in \{0, 1\} \qquad \forall s \in \mathcal{S} \tag{4.83}$$

where (4.79)-(4.80) each define a set $s \in \mathcal{S}$ of constraints that the model will decide to activate or not. We are forcing the model to activate K of them via (4.81). (4.82)-(4.83)[14] define the nature of the decision variables. We discuss a variant of the K-out-of-N constraint in the 💡 **Notation variation in the *K*-out-of-*N* constraint set** box.

14: With a slight abuse of notation, in (4.82) with $\forall j \in x$ we intend all the indices defining the decision variable set x, e.g., $\{0, 1, 2, \cdots\}$. Note that we use the general notation $x_j \in \mathbb{R}_0$ to keep the formulation general, but some x_j variables can be integer.

💡 Notation variation in the *K*-out-of-*N* constraint set

In our notation, we employ $y_s = 1$ to activate constraint set $s \in \mathcal{S}$ by multiplying M on all its right-hand sides by $(1 - y_s)$, so that $y_s = 1$ makes all Ms disappear and the constraint set active. In other references, all Ms are multiplied by y_s. This entails that a constraint set is active when $y_s = 0$ and not when $y_s = 1$ as in our case. This also calls for a change in (4.81), as now the right-hand side should define the number of alternatives we want to be switched off, namely $N - K$, hence $\sum_{s \in \mathcal{S}} y_s = N - K$.
We can express the revised version of the K-out-of-N constraint as follows:

$$\max \; C^T x \tag{4.84}$$

s.t.:

$$\sum_{j \in x} A_{ij} x_j \leq b_i + M y_1 \qquad \forall i \in \mathcal{R}_1 \tag{4.85}$$

$$\dots$$

$$\sum_{j \in x} A_{ij} x_j \leq b_i + M y_N \qquad \forall i \in \mathcal{R}_N \tag{4.86}$$

$$\sum_{s \in \mathcal{S}} y_s = N - K \tag{4.87}$$

$$x_j \in \mathbb{R}_0 \qquad \forall j \in x \tag{4.88}$$

$$y_s \in \{0, 1\} \qquad \forall s \in \mathcal{S} \tag{4.89}$$

Concluding this section, it is crucial to highlight the extreme cases

$K = 0$ and $K = N$. In our notation, when $K = 0$, it indicates that all constraint sets should be de-activated, while $K = N$ implies that all constraint sets should be activated, effectively depriving the model of any choice. Therefore, this constraint type holds true significance when $1 \leq K \leq N - 1$.

4.8.4 Fixed charge constraints

A **fixed charge** constraint is a constraint where a single decision variable or a combination of decision variables can only be greater than zero if another binary decision variable takes a unitary value. In general terms, such a constraint can be expressed as

$$\sum_{j \in x} A_{ij} x_j \leq M y_i \tag{4.90}$$

where y_i is the "activating" binary variable and $M = \max \left\{ \sum_{j \in x} A_{ij} x_j \right\}$ entails that we set M equal to the maximum value the left-hand side can take. The name *fixed charge* can be explained by focusing on the two terms separately:

- **fixed**: activating the binary variable y_i on the right-hand side has no bearing on the number of decision variables activated on the left-hand side. This independence is facilitated by the precise selection of the value for M, as previously elaborated. Therefore, activating y_i constitutes a fixed and necessary condition if we desire any combination of decision variables on the left-hand side to be active as well. Let us consider the following example:

  $$x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 \leq 15y_1 \tag{4.91}$$

  where $x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}$. If all five binary variables were unitary, then the left-hand side would be equal to $1 \times 1 + 2 \times 1 + 3 \times 1 + 4 \times 1 + 5 \times 1 = 15$, hence we set $M = 15$. If $y_1 = 0$, then $x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 \leq 0$ implies that all five variables are only allowed to be 0. Whether we want to activate just x_1 (for a contribution of 1 unit to the left-hand side) or all five variables (for a contribution of 15 units to the left-hand side), we are forced to activate y_1 anyway;
- **charge**: Expanding on the previous point and delving into the real-life implications of such a constraint, it is crucial to note that activating the left-hand side of (4.90) typically does not happen without cost. Without some form of penalty for the choice $y_i = 1$, the model tends to favor it, as it grants more flexibility in selecting appropriate decision variables. To mitigate this tendency, a penalty can be imposed through an additional (for a max problem) $-C_i y_i$ term in the objective function for each binary decision variable y_i controlling the right-hand side of a fixed charge constraint. Here, C_i represents a monetary cost (or an equivalent *charge*, hence the name), which must be incurred to activate the associated left-hand side.

In summary, fixed charge constraints carry a compelling economic interpretation. They address scenarios where an initial "flat fee" (the

fixed charge) must be paid to initiate an activity, alongside a variable revenue or cost directly linked to the activity level. In profit-oriented contexts, the goal typically revolves around maximizing overall profit, while in cost-centric situations, it is about minimizing overall costs. Despite their economic analogy, fixed charge constraints remain applicable even when the initiation of an activity is not explicitly tied to economic incentives or disincentives. To substantiate this claim, we display two examples. In Example 4.7, we tackle a problem explicitly addressing economic implications. In contrast, Example 4.8 pertains to a problem initially devoid of cost considerations, but we introduce a variant that incorporates costs.

Example 4.7 *An aircraft manufacturer is contemplating the launch of two new aircraft models: a narrow-body and a wide-body. The estimated development costs for these models are 1 Billion € and 5 Billion €, respectively. Anticipating market demand, the manufacturer expects to sell each narrow-body aircraft for 110 Million € and each wide-body for 340 Million €. Operating with separate assembly lines for narrow-body and wide-body aircraft, the manufacturer projects a maximum production capacity of 90 narrow-body and 60 wide-body aircraft within the initial two years post-development. Due to workforce constraints, the combined assembly capacity is capped at 120 aircraft. Given these parameters and focusing solely on revenue from aircraft sales and costs stemming from the development, the manufacturer seeks to evaluate the feasibility of developing both aircraft models and determine the optimal production mix in terms of the number of aircraft per type in the first two years after initial deployment. It is important to note that the manufacturer is optimistic, assuming that every aircraft produced will be sold. Our task is to develop a mathematical model that can answer the manufacturer's questions using profit maximization as the objective.*

Our model features a single set, i.e., the set of aircraft types $\mathcal{A} = \{1, 2\}$, indexed by a, where 1 and 2 map the narrow- and wide-body aircraft types, respectively. The aircraft type-specific parameters are C_a, the development cost of aircraft type a, R_a, the revenue per aircraft produced (and purchased) of type a, and N_a, the maximum number of aircraft of type a produced in two years. Additionally, N_{max} specifies the maximum number of aircraft of both types that can be produced within the two-year timeframe. In our context, we have $C_1 = 1,000$, $C_2 = 5,000$, $R_1 = 110$, and $R_2 = 340$ (we divided all monetary values by 1,000,000). In addition, $N_1 = 90$, $N_2 = 60$, and $N_{max} = 120$. To map the aircraft produced (and purchased) per type, we define the integer decision variable x_a. Recalling the definition of fixed charge constraints, we define a second set of binary decision variables y_a that, if unitary, map the commitment of the manufacturer to develop aircraft type a. We define our mathematical model as:

$$\max \quad \sum_{a \in \mathcal{A}} R_a x_a - \sum_{a \in \mathcal{A}} C_a y_a \tag{4.92}$$

s.t.:

$$x_a \leq N_a y_a \qquad \forall a \in \mathcal{A} \tag{4.93}$$
$$\sum_{a \in \mathcal{A}} x_a \leq N_{max} \tag{4.94}$$
$$x_a \in \mathbb{N}_0 \qquad \forall a \in \mathcal{A} \tag{4.95}$$
$$y_a \in \{0, 1\} \qquad \forall a \in \mathcal{A} \tag{4.96}$$

(4.92) defines the profit of the aircraft manufacturer. The first term is the revenue deriving from aircraft sales, the second term is the fixed development cost of both aircraft types. (4.93) is the set of fixed charge constraints. If no development cost for aircraft a is allowed, then no aircraft of that type can be produced ($y_a = 0 \rightarrow x_a \leq 0 \rightarrow x_a = 0$). If the development cost is incurred, then as many aircraft as the maximum production allows can be sold ($y_a = 1 \rightarrow x_a \leq N_a$). (4.94) limits the overall production of aircraft to stay within the bounds imposed by the available workforce, while (4.95)-(4.96) define the integer (resp. binary) nature of decision variables x_a and y_a. The extended mathematical formulation, for the sake of completeness, is:

$$\max \; 110x_1 + 340x_2 - 1,000y_1 - 5,000y_2 \tag{4.97}$$

s.t.:

$$x_1 \leq 90y_1 \tag{4.98}$$
$$x_2 \leq 60y_2 \tag{4.99}$$
$$x_1 + x_2 \leq 120 \tag{4.100}$$
$$x_1, x_2 \in \mathbb{N}_0 \tag{4.101}$$
$$y_1, y_2 \in \{0, 1\} \tag{4.102}$$



Figure 4.5: A glimpse of the dome of Catania.

Example 4.8 *A traveler is flying to the beautiful city of Catania, Italy (see Figure 4.5) in a few days. They want to carry a set of items $\mathcal{I}$, indexed by i, ranging from clothes, to books, electronics, and outdoor gear. Each item features a specific weight W_i and volume V_i. The traveler has at their disposal a set of luggage $\mathcal{L}$, indexed by l, each capable of accommodating a volume $\overline{V}_l$. In addition, the airline the traveler is flying with requires a maximum weight per luggage equal to $\overline{W}$. The traveler is adamant they have a sufficiently large set of luggage to transport all the needed items $i \in \mathcal{I}$. Because every piece of luggage after the first one must be paid extra via a flat rate, the traveler would like to devise a packing strategy that avoids paying unnecessary luggage fees. Therefore, our objective is to devise a mathematical model aimed at assisting the traveler, with the goal of minimizing the required pieces of luggage used.*

For this problem, we have already introduced the sets and parameters. We can directly focus on the decision variables. One of our tasks is to assign every item to a piece of luggage, hence we can define x_{il} as a binary variable that takes a unitary value if item i is assigned to luggage l. Additionally, we need to track the number of pieces of luggage being used. To address this, we define another binary variable y_l that

takes a unitary value if piece of luggage l is utilized. This second set of decision variables directly leads to the definition of the objective function, namely min $\sum_{l \in \mathcal{L}} y_l$. We must link our x_{il} and y_l variables. The concept is straightforward: we do not need to transport empty luggage, but every piece of luggage containing at least one item must accompany us (this should resonate with the fixed charge constraint type). Furthermore, we must ensure that items allocated to a piece of luggage do not surpass the available volume (as limited by the piece of luggage itself) or weight (as restricted by the airline). We directly present and then discuss the formulation:

$$\min \sum_{l \in \mathcal{L}} y_l \tag{4.103}$$

s.t.:

$$\sum_{l \in \mathcal{L}} x_{il} = 1 \qquad \forall i \in \mathcal{I} \tag{4.104}$$

$$\sum_{i \in \mathcal{I}} W_i x_{il} \leq \overline{W} \qquad \forall l \in \mathcal{L} \tag{4.105}$$

$$\sum_{i \in \mathcal{I}} V_i x_{il} \leq \overline{V}_l \qquad \forall l \in \mathcal{L} \tag{4.106}$$

$$\sum_{i \in \mathcal{I}} x_{il} \leq |\mathcal{I}| y_l \qquad \forall l \in \mathcal{L} \tag{4.107}$$

$$x_{il} \in \{0, 1\} \qquad \forall i \in \mathcal{I}, l \in \mathcal{L} \tag{4.108}$$

$$y_l \in \{0, 1\} \qquad \forall l \in \mathcal{L} \tag{4.109}$$

(4.103) aims at minimizing the used pieces of luggage. (4.104) ensures that the traveler carries to their destination every item they need[15]. (4.105)-(4.106) guarantee that each piece of luggage adheres to weight and volume restrictions. In (4.105), for a given $l \in \mathcal{L}$ the left-hand side comprises x_{il} multiplied by their respective weight W_i. Any combination of items can be chosen as long as the cumulative weight does not exceed $\overline{W}$. Similarly, (4.106) operates on a volume basis. The crux lies in (4.107), the pivotal fixed charge constraint. When $y_l = 0$, then $\sum_{i \in \mathcal{I}} x_{il} \leq 0$ effectively bars the utilization of the piece of luggage, as it implies all x_{il} should be 0 for the current piece of luggage l. Conversely, if $y_l = 1$, we allow piece of luggage l to be used and potentially packed with all items ($\sum_{i \in \mathcal{I}} x_{il} \leq |\mathcal{I}|$) pending weight and volume requirements. **However, this decision comes at a cost: the objective function increases by one unit**.

15: Note that (4.104) is quite a stringent constraint as mentioned in the text and assumes the traveler has enough capacity (in terms of number and weight/volume of luggage) to transport everything. Otherwise, the model might be infeasible.

We now introduce a slight variation. In this scenario, the traveler realizes they have overlooked adding any checked-in luggage. They discover that the airline does not impose a flat rate per piece of luggage. Instead, each piece of luggage $l \in \mathcal{L}$ is associated with a specific purchase cost C_l if carried on-board. The updated challenge is to modify the model to incorporate this variation.

We realize the only change we need to apply is to Equation 4.103, which is translated into

$$\min \sum_{l \in \mathscr{L}} C_l y_l \tag{4.110}$$

(4.104)-(4.109) still apply to the model, because we did not change any other parameter or requirement. This small variation entails quite a substantial change. With a fixed flat rate C per piece of luggage, $\min \sum_{l \in \mathscr{L}} C y_l = \min C \sum_{l \in \mathscr{L}} y_l = C \min \sum_{l \in \mathscr{L}} y_l$. Hence, minimizing the number of pieces of luggage on-board is equivalent to minimizing the overall cost, which lead to (4.103). Conversely, since in (4.110) every piece of luggage $l \in \mathscr{L}$ has a specific cost, it might be more advantageous to check-in more “cheap" pieces of luggage than fewer “expensive" ones, pending weight and volume restrictions are satisfied. For example, a hat-trick of pieces of luggage costing 20, 30, and 40€ is more advised, given (4.110), than a single piece of luggage costing 100€.

4.9 Final remarks

To make sure that a mathematical model is correct, we should always verify the following:

- **connection between objective function and constraints**: are variables connected in a way that enforces relationships that are meaningful mathematically and practically?
- **feasibility of the mathematical model and of the underlying original problem**. Is the model triggering the right results if we force certain decision variables, for example, to be 0?
- **type and amount of constraints**. Are we imposing the right amount of constraints? For example, in Example 4.8 we need one constraint per piece of luggage $l \in \mathscr{L}$ to ensure weight restrictions are met. We should verify we have $|\mathscr{L}|$ constraints of that type;
- **indexes**. Are the indices right and in the right place (e.g., in the summations $(\sum \cdots)$ versus in the definition of how many constraints we need of a certain type $(\forall \cdots)$?

The mathematical notation introduced in this chapter may initially appear cumbersome, as do the mentioned remarks. However, both are crucial for improving the readability of a mathematical model and ensuring that the defined model is accurate and meets the requirements of the original problem.

5 Linearization techniques

A line is a dot that went for a walk.

Paul Klee

5.1 Product of decision variables 45
5.1.1 Product of two binary variables 46
5.1.2 Product of a binary and a continuous decision variable 48
5.2 Absolute value 49
5.3 Piecewise linear formulations 51
5.4 If-else statement 53

In Chapter 4, we delved into how the mathematical models emphasized in this book adhere to linearity. This necessity extends to both the objective function and every constraint, stipulating that decision variables can only appear in linear combinations. The requirement seems quite stringent at first glance and indeed prevents some problems from being tackled and solved with the solution methods described in Chapters 6, 7, and 8. Notwithstanding, **many non-linear operators can be mathematically linearized so that the resulting model is linear**. The price to pay, because of the "no free lunch" theorem[1], is that **the linearization process generally entails the addition of auxiliary decision variables and/or constraints, hence contributing to the increase of the complexity of the original mathematical model**.

1: For a more formal definition of such a theorem, we refer interested readers to this Wikipedia page.

Some readers might be surprised to realize that, in Chapter 4, we already "implicitly" applied some linearization techniques. One example is the either-or constraint from Section 4.8.2 because such a constraint maps the (non-linear) logical operator $\vee$[2]. The "algorithmic" price we pay is the addition of binary decision variable y. Another fitting example is the fixed charge constraint (Section 4.8.4). If we assume that the initial fixed charge cost C allows the generation of revenue proportionally to x, where x maps the level of productivity or sales (e.g., aircraft sales in Example 4.7), via parameter R (revenue per unit x), we can define the profit P as

2: Logical or. See this Wikipedia page for more info.

$$P = \begin{cases} 0 & \text{if x=0} \\ Rx - C & \text{otherwise} \end{cases} \tag{5.1}$$

which is a non-linear function featuring a discontinuity in $x = 0$. Leveraging the addition of binary variable y both in the constraints and the objective facilitates the linearization of such a discontinuity.

Despite their inherent non-linear nature, we opted to incorporate the aforementioned constraints in Chapter 4 because they are "standard" constraint types as per OR standards. In the subsequent sections, we shift our focus to other constraint types. Although widely utilized in mathematical modeling, these types are more renowned (or notorious, depending on the context) for their non-linear characteristics, necessitating a linearization process.

5.1 Product of decision variables

Often, our problem formulations necessitate representing quantities of interest through the product of two decision variables. For instance,

consider a scenario involving the Champions League final. If two specific teams (indexed i and j) qualify for the final, we aim to capture this event in our mathematical model. We could define binary variables x_i and x_j, each taking a value of 1 if team i or team j reaches the final, respectively. Consequently, we can introduce another binary variable $y_{ij} = x_i x_j$, which equals 1 if both teams qualify for the final ($y_{ij} = 1 \times 1 = 1$), and 0 otherwise. However, this relationship is non-linear due to the product of two binary decision variables. Fortunately, it is linearizable, as described in Section 5.1.1. Similarly, situations may arise where we encounter the product of a binary and a continuous decision variable, which is also linearizable, as discussed in Section 5.1.2.

5.1.1 Product of two binary variables

We already introduced, for this specific case, our intended goal. Given binary variables x_i and x_j, our goal is to capture $y_{ij} = x_i x_j$, but in a linear fashion. We achieve this by defining the following constraints:

$$y_{ij} \geq x_i + x_j - 1 \tag{5.2}$$
$$y_{ij} \leq x_i \tag{5.3}$$
$$y_{ij} \leq x_j \tag{5.4}$$
$$y_{ij}, x_i, x_j \in \{0, 1\} \tag{5.5}$$

where (5.2) is the key constraint. It forces y_{ij} to be unitary if both x_i and x_j are unitary $y_{ij} \geq 1 + 1 - 1 = 1 \rightarrow y_{i,j} = 1$ and leaves it the freedom to be either 1 or 0 otherwise. If all the y_{ij} decision variables appear in the objective function in a term that should be minimized, then the model will assign them a value of 0 as this is beneficial for the objective. In such a case, (5.3)-(5.4) are not strictly needed, but they help with the linear relaxation of the problem (see Chapters 6-8). They force y_{ij} to be 0 as soon as one of the two original binary variables is 0 ($x_i = 0 \rightarrow y_{ij} \leq 0 \rightarrow y_{ij} = 0$. The same applies if $x_j = 0$). Similarly, in the case of a maximization problem, the model will favor assigning y variables to be 1 all the time. Yet having (5.3)-(5.4) prevents this as soon as one variable between x_i and x_j is not unitary. We showcase an application of such a linearization in Example 5.1.

Example 5.1 *In many hub airports worldwide, a significant portion of passengers are transfer passengers, meaning the hub airport neither marks the origin nor the destination of their journey. For such airports, a critical KPI is the connection time, closely linked to the distance for transfer passengers. Requiring transfer passengers to disembark from their first flight at one end of the airport and then traverse a considerable distance to connect to their subsequent flight can lead to discomfort and impede the overall passenger experience. The problem of efficiently assigning aircraft to gates is generally called the Gate Assignment Problem (GAP), and can have different objectives reflecting the different needs of the main stakeholders involved, namely the airport, the airlines, and passengers. In this example, we employ the perspective of the passengers. The airport knows the set of inbound/outbound flights $\mathcal{F}$ (indexed by f) that are to be operated on a specific day. The airport is also characterized by a set of gates $\mathcal{G}$ (indexed by g), such as the ones shown in Figure 5.1, where $D_{g_1 g_2}$ is the walking distance*

between gates g_1 and g_2. Additionally, the airport has received by all airlines data on the number of transfer passengers $P_{f_1 f_2}$ that are expected to transfer inside the terminal from flight f_1 to flight f_2. For example, if f_1 is a flight from Milan Linate Airport to Amsterdam Schiphol Airport, f_2 is a flight from Amsterdam Schiphol Airport to John F. Kennedy International Airport, and $P_{f_1 f_2} = 15$, the hub airport (Amsterdam Schiphol Airport in this case) is aware that 15 passengers are connecting there from Italy on their way to the United States. We aim to formulate a suitable objective function for the GAP, aiding the hub airport in minimizing the total distance traveled by transfer passengers within a given day.



Figure 5.1: Aircraft gated at Amsterdam Schiphol Airport.

The airport knows the overall number of transfer passengers expected during the analyzed day, as the total number is $\sum_{f_1 \in \mathcal{F}} \sum_{f_2 \in \mathcal{F}} P_{f_1 f_2}$. We consider each pair of flights and aggregate the number of transfer passengers between them. It is important to note that we should account for both directions, i.e., (f_1, f_2) and (f_2, f_1). For instance, in the scenario mentioned earlier, passengers may also transfer in the Netherlands from the United States on their way to Italy. The crucial decision that drives our objective is how to assign aircraft to gates. In fact, if we assign flight f_1 to gate g_1 and flight f_2 to gate g_2, the overall walking distance related to those two flights is $D_{g_1 g_2} \left(P_{f_1 f_2} + P_{f_2 f_1}\right)$, which is obtained by multiplying the distance between the two gates by the number of transfer passengers in both directions (from f_1 to f_2 and vice versa).

Because the airport needs to assign flights to gates, a reasonable decision variable could be $x_{fg} \in \{0, 1\}$, taking unitary value if flight f is assigned to gate g. Then, our goal is to add to our objective function every term in the $D_{g_1 g_2} \left(P_{f_1 f_2} + P_{f_2 f_1}\right)$ form *if flight f_1 is assigned to gate g_1 and flight f_2 to gate g_2*. We could rewrite this statement in more mathematical terms as *"if $x_{f_1 g_1} x_{f_2 g_2} = 1$"* and notice the expression entails the product of two binary decision variables. Hence, we could rely on (5.2)-(5.4), with the complication that now each binary variable depends on two indices, and hence y depends on four indices. We define $y_{f_1 g_1 f_2 g_2} \in \{0, 1\}$ which takes a unitary value if flight f_1 is assigned to gate g_1 and flight f_2 to gate g_2. If we sort flights by increasing order, i.e., $\mathcal{F} = \{1, 2, 3, \cdots\}$, we should define such a decision variable $\forall f_1, f_2 \in \mathcal{F} : f_2 > f_1, g_1, g_2 \in \mathcal{G}$. **With $f_2 > f_1$ we mean that flight f_2 comes after flight f_1 in the set (basically, that index f_2 is greater than index f_1). This is done to reduce the variables to be defined. In fact, stating that f_1 is assigned to g_1 and f_2 to g_2 is equivalent to stating that f_2 is assigned to g_2 and f_1 to g_1. Conversely, we need to cycle over all gates twice. As a matter of fact, $y_{f_1 g f_2 g}$ is a valid decision variable as it allows for the assignment of two flights to the same gate g if they are sufficiently spaced apart in time.** We can fully formalize our y decision variables as:

$$y_{f_1 g_1 f_2 g_2} \geq x_{f_1 g_1} + x_{f_2 g_2} - 1 \qquad \forall f_1, f_2 \in \mathcal{F} : f_2 > f_1, g_1, g_2 \in \mathcal{G} \qquad (5.6)$$

$$y_{f_1 g_1 f_2 g_2} \leq x_{f_1 g_1} \qquad \forall f_1, f_2 \in \mathcal{F} : f_2 > f_1, g_1, g_2 \in \mathcal{G} \qquad (5.7)$$

$$y_{f_1 g_1 f_2 g_2} \leq x_{f_2 g_2} \qquad \forall f_1, f_2 \in \mathcal{F} : f_2 > f_1, g_1, g_2 \in \mathcal{G} \qquad (5.8)$$

Thanks to the introduction of the $y_{f_1 g_1 f_2 g_2}$ decision variables, we can now express our objective function (in a linear fashion) as:

$$\min \sum_{f_1 \in \mathcal{F}} \sum_{f_2 \in \mathcal{F}: f_2 > f_1} \sum_{g_1 \in \mathcal{G}} \sum_{g_2 \in \mathcal{G}} D_{g_1 g_2} \left(P_{f_1 f_2} + P_{f_2 f_1} \right) y_{f_1 f_2 g_1 g_2} \tag{5.9}$$

(5.9) expresses the overall walking distance of transfer passengers and depends on the gate assignment choices that the airport performs. Note that we purposely decided not to provide a full GAP formulation, as there is no standard formulation in the first place. Apart from some generic constraints (e.g., every flight f should be assigned to a gate g: $\sum_{g \in \mathcal{G}} x_{fg} = 1 \; \forall f \in \mathcal{F}$), there is a lot of variability related to the type of airport, the modeling assumptions, the stakeholder's perspective when deciding the objective function. We refer interested readers to Daş et al., 2020 for a comprehensive review of GAP formulations. Finally, we comment on the size of the decision variable set y for large hubs in 💡 **A note on the number of $y_{f_1 f_2 g_1 g_2}$ decision variables in the GAP** box.

💡 A note on the number of $y_{f_1 f_2 g_1 g_2}$ decision variables in the GAP

Hub airports such as Amsterdam Schiphol Airport can handle more than 1,000 flights per day and can be characterized by roughly 200 gates. It should be noted that a flight can generally be assigned only to a subset of gates because of customs restrictions (e.g., Schenghen vs. non-Schenghen flights), airline preferences, or other reasons. Notwithstanding, let us assume a hub airport that, on a given day, must handle $|\mathcal{F}| = 1,000$ flights and that each flight can be assigned to a subset $\mathcal{G}_f$ of gates that "only" comprises 10 options. Hence, we could get a rough estimate of the number of $y_{f_1 f_2 g_1 g_2}$ decision variables needed as:

$$\begin{aligned} |y_{f_1 f_2 g_1 g_2}| &= \frac{|\mathcal{F}|(|\mathcal{F}| - 1)}{2} \times \mathcal{G}_f \times \mathcal{G}_f \\ &= \frac{1,000 \times 999}{2} \times 10 \times 10 \simeq 50,000,000 \end{aligned}$$

which highlights the complexity of such a problem for large hub airports. **It is also important to highlight that the GAP is generally not solved for a full day, but smaller planning windows are solved sequentially (which entails more manageable problem sizes). Because of flight delays and many other unforeseen circumstances, a GAP solution will be subject to continuous changes throughout the day anyway. We have all experienced at least once in our lifetime a sudden gate change for one of our flights!**

5.1.2 Product of a binary and a continuous decision variable

Some mathematical formulations might require the product of a continuous[3] and a binary variable in the constraints or the objective. Let $x \in \{0,1\}$ be the binary variable and $y \leq U$ the continuous variable where U is a maximum value (upper bound) such a variable can take. Introducing a new continuous variable z, defining $z = xy$ achieves

3: Note: the decision variable can also be an integer with no changes in the linearization process.

our intended goal in a non-linear fashion. The linearization of this relation entails the reworking of the xy product via the following set of constraints:

$$z \leq Ux \tag{5.10}$$
$$z \leq y \tag{5.11}$$
$$z \geq y - U(1 - x) \tag{5.12}$$
$$x \in \{0, 1\} \tag{5.13}$$
$$y \leq U \tag{5.14}$$
$$z \geq 0 \tag{5.15}$$

where in (5.10) and (5.12) U plays the role of a big-M.

Let us verify that the intended goal is achieved by analyzing the behavior of (5.10)-(5.12) for different combinations of x and y. If $x = 0$ and $y = 0$ we obtain $z \leq 0$, $z \leq 0$, and $z \geq 0$, which implies $z = 0$. If $x = 0$ and $y > 0$ we obtain $z \leq 0$, $z \leq y$, and $z \geq y - U$, which implies $z = 0$. If $x = 1$ and $y = 0$ we obtain $z \leq U$, $z \leq 0$, and $z \geq 0$, which implies $z = 0$. Finally, if $x = 1$ and $y > 0$ we obtain $z \leq U$, $z \leq y$, and $z \geq y$, which implies $z = y$ as requested.

5.2 Absolute value

Some mathematical formulations rely on the absolute value of the difference between two continuous or integer variables to function correctly. Consider a model tasked with efficiently packing two-dimensional boxes into a two-dimensional bin along an $x - z$ vertical plane. Due to Earth's gravity, floating boxes are not allowed. Therefore, box i can only have a z-coordinate greater than zero if stacked on top of another box j. This requirement can be verbalized as *"if the upper side of box j has the same height as the lower side of box i, then box i can be stacked on top of box j"*.

We can mathematically translate the absolute value $|x-y|$ of the difference of continuous decision variables x and y with the following set of equations:

$$x - y \leq z_{xy} \tag{5.16}$$
$$y - x \leq z_{xy} \tag{5.17}$$
$$z_{xy} \leq x - y + M(1 - m_{xy}) \tag{5.18}$$
$$z_{xy} \leq y - x + Mm_{xy} \tag{5.19}$$
$$m_{xy} \in \{0, 1\} \tag{5.20}$$
$$x, y, z_{xy} \geq 0 \tag{5.21}$$

where z_{xy} is an auxiliary continuous variable which is ensured to be equal to $|x - y|$, M is an upper bound (big-M) on the value z_{xy} can take, and $m_{xy} \in \{0, 1\}$ takes a unitary value if $x \geq y$. Let us verify the

validity of (5.16)-(5.19) considering the three distinct cases $x - y = \alpha > 0$, $x - y = 0$, and $x - y = \alpha < 0$.

In the first case ($x - y = \alpha > 0$), we have

$$\begin{aligned} \alpha &\leq z_{xy} \\ -\alpha &\leq z_{xy} \\ z_{xy} &\leq \alpha \\ z_{xy} &\leq \alpha + M \end{aligned}$$

with the second and fourth constraints being dummy[4] constraints, i.e., redundant. The first and third constraints ($z_{xy} \geq \alpha$ and $z_{xy} \leq \alpha$) imply $z_{xy} = |x - y| = x - y$ as required.

4: Note; we define *dummy* a constraint that is always satisfied. Dummy constraints are found oftentimes in formulations where big-Ms and a binary variable appear in two linked constraints. Depending on the value taken by the binary, one of the two constraints will be active and the other dummy (or vice versa). Previously in the book, we also used the term de-activated. Hence, we might be using the terms dummy, redundant, or de-activated interchangeably.

In the second case ($x = y$), we have:

$$\begin{aligned} 0 &\leq z_{xy} \\ 0 &\leq z_{xy} \\ z_{xy} &\leq 0 \\ z_{xy} &\leq M \end{aligned}$$

with the fourth constraint being dummy. The first (or second) and third constraints ($z_{xy} \geq 0$ and $z_{xy} \leq 0$) imply $z_{xy} = |x - y| = 0$ as required. **Note that this second case is the limit case of both the first and third one when $\alpha = 0$, but we opted to highlight it as a special case of its own**.

Finally, in the third case ($y - x = \alpha > 0$), we have

$$\begin{aligned} -\alpha &\leq z_{xy} \\ \alpha &\leq z_{xy} \\ z_{xy} &\leq -\alpha + M \\ z_{xy} &\leq \alpha \end{aligned}$$

with the first and third constraints being dummy. The second and fourth constraints ($z_{xy} \geq 0$ and $z_{xy} \leq 0$) imply $z_{xy} = |x - y| = y - x$ as required.

Note that, in some formulations, it may be much easier to handle absolute value cases. For example, if we have a constraint:

$$|x_1 - x_2| \leq 2$$

it is easy to represent the same with two constraints as follows:

$$x_1 - x_2 \leq 2 \tag{5.22}$$

$$-x_1 + x_2 \leq 2 \tag{5.23}$$

which actually represents the feasible region given in Figure 5.2.

However, we need to be careful if we have the case with

$$|x_1 - x_2| \geq 2$$

which creates a non-convex region as given in Figure 5.3 where the same trick cannot be used. In this case, we have two disjoint feasible regions and we can resort to the techniques discussed in Section 4.8.2.

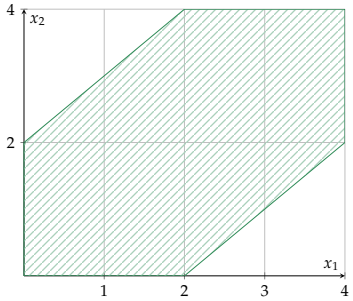


Figure 5.2: Feasible region with the absolute value constraint - convex case.

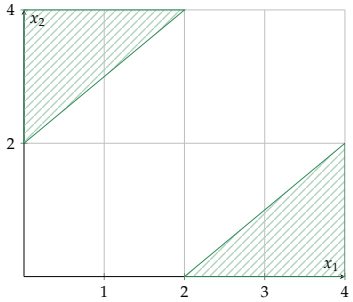


Figure 5.3: Feasible region with the absolute value constraint - non-convex case.

5.3 Piecewise linear formulations

In some mathematical formulations, the objective function and/or constraints may be represented by a nonlinear function such as polynomials or an exponential curve. For example, the profit as the objective function of a mathematical model might have a diminishing rate of returns depending on decision variable x as in Figure 5.4. This logarithmic function can be approximated by a piecewise linear curve as already indicated on the figure. Therefore, the profit curve can be represented by those three linear segments with the variable profits of c_1, c_2, and c_3 respectively, corresponding to their slopes.

In order to represent this piecewise linear relation, we need to adapt the mathematical formulation. First of all, we need to represent the original decision variable x in terms of three new decision variables corresponding to the linear segments as follows:

$$x = \delta_1 + \delta_2 + \delta_3 \tag{5.24}$$

where δ_1 is the linear segment for the values of x between 0 and the first break point b_1, δ_2 corresponds to the segment between b_1 and b_2 and finally δ_3 corresponds the the segment between b_2 and b_3. The δ variables are therefore given as follows:

$$0 \leq \delta_1 \leq b_1 \tag{5.25}$$
$$0 \leq \delta_2 \leq b_2 - b_1 \tag{5.26}$$
$$0 \leq \delta_3 \leq b_3 - b_2 \tag{5.27}$$

The objective function then can be reformulated as:

$$\max \; c_1\delta_1 + c_2\delta_2 + c_3\delta_3 \tag{5.28}$$

so that it approximates the original full line representing the non-linear function $f(x)$ in Figure 5.4 with the piecewise linear dashed line. **It follows that the more segments we choose, the more accurate (in general) the approximation, at the cost of an increased size of the linearized model**.

For this piecewise linear transformation to be valid, we need to ensure that $\delta_1 = b_1$ whenever $\delta_2 > 0$ and similarly that $\delta_2 = b_2$ whenever $\delta_3 > 0$.

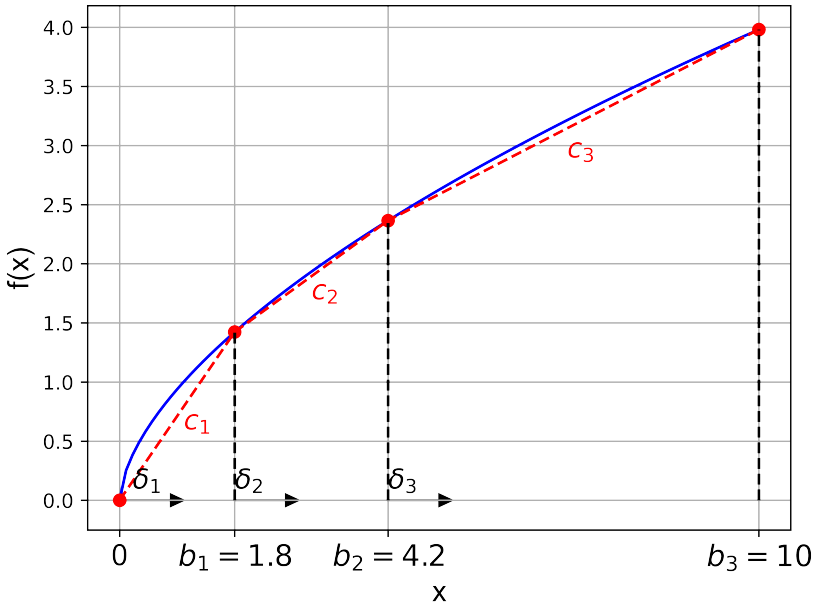


Figure 5.4: Piecewise linear approximation of a non-linear function.

As per (5.28), we are reconstructing x as the summation of the three segments, but we only need the second one if we exceed the bound b_1 of the first one and the third one if we exceed the bound b_2 of the second segment. Note that if $c_1 > c_2 > c_3$ this will already be ensured in case of a maximization problem as we have at hand. Nevertheless, to have a general formulation that addresses other cases we need to define binary variables to ensure this. Let us define:

$w_1 \in \{0, 1\}$, 1 if $\delta_1 = b_1$, 0 otherwise
$w_2 \in \{0, 1\}$, 1 if $\delta_2 = b_2$, 0 otherwise.

Namely, if $w_1 = w_2 = 0$, only the first line segment is active and $x \leq b_1$. If $w_1 = 1$ and $w_2 = 0$, the first and second line segments are active and $b_1 \leq x \leq b_2$. Finally, if $w_1 = w_2 = 1$, all three segments are active and $x \geq b_2$. **Therefore, we represent the decision variable x through three linear segments with the definition of three continuous variables (δ) and two binary decision variables (w). The number of the binary decision variables is always equal to the number of linear segments minus one.**

Considering the above definitions and requirements, we can provide the full formulation as follows:

$$\max \quad c_1\delta_1 + c_2\delta_2 + c_3\delta_3 \tag{5.29}$$

s.t.:

$$b_1 w_1 \leq \delta_1 \leq b_1 \tag{5.30}$$
$$(b_2 - b_1)w_2 \leq \delta_2 \leq (b_2 - b_1)w_1 \tag{5.31}$$
$$0 \leq \delta_3 \leq (b_3 - b_2)w_2 \tag{5.32}$$
$$w_1, w_2 \in \{0, 1\} \tag{5.33}$$

This formulation can be applied to piecewise linear curves with any number of segments. In that case, we would need to have a general variable δ_j for each segment j with a length of L_j, and the constraints will read:

$$L_j w_j \leq \delta_j \leq L_j w_{j-1} \tag{5.34}$$

5.4 If-else statement

In some mathematical formulations, we might require a constraint to become active if a certain choice is being made and to be redundant otherwise. This requirement can be interpreted as **an if-else statement where if a certain condition is met, the constraint is active, while it is made redundant otherwise**. We show a standard application in Example 5.2.

Example 5.2 *Let us consider a truck that must perform deliveries to a set of customers $\mathcal{C}$ (indexed by i or j) scattered across town. We need to assist the trucking company in assessing the best sequence of customers to visit, given that they all have specified different preferred delivery times. Hence, we want to deliver everything on time while avoiding unnecessary detours.*

In Section 4.1, we already anticipated that a routing decision variable x_{ij} maps, if unitary, that the truck moves from customer i to customer j. In addition, because keeping track of time is important in this problem to avoid delays, we can define continuous decision variable t_i as the time when our truck starts the delivery service at customer $i \in \mathcal{C}$. Additionally, we assume that performing the delivery at a customer $i \in \mathcal{C}$ takes P_i time-units and that the traveling time between customers i and j requires T_{ij} time-units. In such a setting, keeping track of time is easy if a sequence of customers is pre-assigned to us. Let us assume the first three customers are, in sequence, $c = 1, 2, 3$ and that the truck starts the journey from a depot indexed by 0 at time 0. Hence, $t_1 = T_{01}$ implies that the start of service of customer 1 is simply equal to the traveling time from the depot to the customer (assuming we can start the service as soon as we arrive there). Conversely, $t_2 = t_1 + P_1 + T_{12} = T_{01} + P_1 + T_{12}$ implies that the start of service of customer 2 is equal to the start of service of customer 1 (t_1) plus the time needed for the delivery (P_1) and the traveling time between customers (T_{12}). Following the same logic, $t_3 = t_2 + P_2 + T_{23} = T_{01} + P_1 + T_{12} + P_2 + T_{23}$.

Our main issue is that we are not given a pre-assigned sequence of customers, as defining the most appropriate sequence is exactly our task. Hence, we need to enforce a **time-precedence constraint** such as

$$t_j \geq t_i + P_i + T_{ij} \tag{5.35}$$

only if the truck moves from i to j[5]. This is exactly an if-else constraint type because we can rephrase the following requirement as *if the truck drives from i to j, then time-precedence constraint (5.35) must be enforced; else, it must be made redundant*. We visually display the situation in Figure 5.5, where we assume that the truck moves from i to j, but represent a third

5: Note that in (5.35) we do not have an equality, but an $\geq$ inequality. As is many applications the goal is to minimize time, the equality will hold. Another reason why the $\geq$ is preferred is that the truck might have to wait at the customer because it arrived too early. Hence the start of the service time could be delayed.

customer l symbolizing that the arc (l, j) is also an option we could consider in our mathematical model.

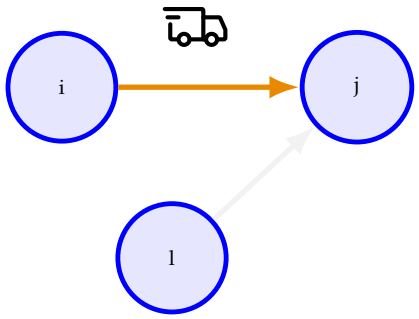


Figure 5.5: Representation of a customer pair (i, j) served in sequence by a truck. Because of the sequence, a time-precedence constraint $t_j \geq t_i + P_i + T_{ij}$ must hold.

We achieve the intended goal by leveraging the ability of a big-M constant to activate or de-activate a constraint depending on the value taken by a binary decision variable which, in this case, is x_{ij}. The correct form of the constraint is

$$t_j \geq t_i + P_i + T_{ij} - M_{ij}(1 - x_{ij}) \qquad \forall i, j \in \mathcal{N} \tag{5.36}$$

where we are specifying that such a constraint must be enforced for every pair on nodes i, j in our network that comprises both the depot and all customers (with $\mathcal{N} = \{0, 1, \cdots, |\mathcal{C}|\}$ we mean the full set of nodes). The additional term $M_{ij}(1 - x_{ij})$ satisfies our needs. If $x_{ij} = 1$, then we obtain the original constraint (5.35) that we want to enforce. If $x_{ij} = 0$, then we obtain

$$t_j \geq t_i + P_i + T_{ij} - M_{ij} \tag{5.37}$$

which can be re-written as

$$M_{ij} \geq t_i - t_j + P_i + T_{ij} \tag{5.38}$$

Note that in (5.36) (and as a consequence (5.37)-(5.38)) we use M_{ij} to reflect that the big-M can be customized for each (i, j) customer pair to avoid excessively large constants. Let us assume that every customer specifies a time-window where they would like their delivery to start being carried out, where the earliest time is E_i and the latest time is L_i. This constraint is stringent, implying that deliveries beyond the planned time-window are not permissible. We can extend this rigidity by defining analogous values for the depot, indicating the earliest time a truck can depart (start of shift) and the latest time it should return (end of shift). Therefore, in constraint (5.38), we can calculate the maximum possible value for the right-hand side as follows: since t_i is preceded by a positive sign, we substitute it with the latest (i.e., largest) permissible time node i can be visited, denoted as L_i. Conversely, as t_j is preceded by a negative sign, we substitute it with the earliest (i.e., smallest) value it can assume, denoted by E_j. Constants such as P_i and T_{ij} remain unchanged and should be preserved as they are. Hence, we could assign to every M_{ij} the value of

$$M_{ij} = \max\left\{0, L_i - E_j + P_i + T_{ij}\right\} \tag{5.39}$$

which guarantees that M_{ij} is "large enough" to make a constraint redundant, if needed, without being excessively large. Note that (5.39) displays a max operator with the first terms being 0. In fact, $L_i - E_j + P_i + T_{ij} < 0$ implies $E_j > L_i + P_i + T_{ij}$, meaning that the delivery at customer j will have to be carried out after the delivery at customer i (and not necessarily as the next one after i). **Hence, no M_{ij} is formally needed as $t_j \geq t_i + P_i + T_{ij}$ will be satisfied anyway**.

Referring back to Figure 5.5 and focusing on node j as the "destination" node, if we assume it is visited right after node i the application of (5.36) yields

$$t_j \geq t_i + P_i + T_{ij} \tag{5.40}$$
$$t_j \geq t_l + P_l + T_{lj} - \max\left\{0, L_l - E_j + P_l + T_{lj}\right\} \tag{5.41}$$

where (5.40) is active as the time-precedence between i and j must be enforced, while (5.41) is made redundant. Note that (5.41) does not imply that customer l is not visited before customer j, but that **they are not visited immediately before customer j, which is a substantial difference modeling-wise. The final routing could, for example, be $l \rightarrow i \rightarrow j$, and (5.40)-(5.41) would still hold**. We let readers verify that, in such a case, a similar time-precedence constraint is active between l and i as $x_{li} = 1$.

We conclude the section by providing a general form of the if-else constraint. Let us assume that our goal is to allow our model to activate or not a generic expression $\sum_{j \in x} A_{ij} x_j \leq b_i$ using a binary decision variable y_i. Then, the generic form of an if-else constraint can be expressed as

$$\sum_{j \in x} A_{ij} x_j - b_i \leq M(1 - y_i) \tag{5.42}$$

where in (5.42) the constraint is active if $y_i = 1$ and redundant if $y_i = 0$. In Part IV-Part V we will showcase several if-else constraints playing a crucial role in different mathematical models.

Part III

Solution Methods

The simplex method 6

> Still round the corner there may wait A new road or a secret gate And though I oft have passed them by A day will come at last when I Shall take the hidden paths that run West of the Moon, East of the Sun.
>
> John Ronald Reuel Tolkien

6.1 Graphical representation of an LP and corner points 59
6.2 Augmented form of an LP 67
6.2.1 Inequality constraints in the $\leq$ form 68
6.2.2 Equality constraints . . 70
6.2.3 Inequality constraints in the $\geq$ form 71
6.2.4 Final remarks 73
6.3 The simplex method: description of the algorithm 73
6.3.1 Basic and non-basic variables 73
6.3.2 Basic solutions 75
6.3.3 The simplex tableau . . 77
6.3.4 The simplex algorithm: how to iterate and when to stop 80
6.4 Examples 90
6.5 Additional considerations 100

6.1 Graphical representation of an LP and corner points

Mathematical models representing real-life situations could entail millions of decision variables and constraints. This makes their graphical representation not possible in an interpretable way. Mathematical models with three decision variables (namely x_1, x_2, and x_3) could still be mapped in a three-dimensional space, but we will focus on two-dimensional examples (with decision variables x_1 and x_2) that can be displayed in a two-dimensional plane. In such a plane, constraints are half-planes if they are inequalities and lines if they are equalities as long as the model is an LP (and hence, all constraints are linear). Being able to properly translate a two-dimensional LP into its graphical representation is a first step towards the understanding of the formal solution method that efficiently can tackle larger models.

Let us consider the following example.

Example 6.1 *A street food company has a budget of 360,000€ to invest in two different types of trucks. Trucks of the first type serve tacos. They cost 30,000€ each, there is a maximum of 8 available for purchase, and it is expected that will attract 2,000 customers each per week. Trucks of the second type serve burritos. They cost 40,000€ each, there is a maximum of 6 available for purchase, and it is expected that will attract 5,000 customers each per week. Our goal is to determine how many trucks of the first type (decision variable x_1) and of the second type (decision variable x_2) to purchase to maximize the number of expected customers while satisfying constraints on availability and budget.*

We can translate this problem into the following LP:

$$\max \; Z = 2x_1 + 5x_2 \tag{6.1}$$

s.t.:

$$x_1 \leq 8 \tag{6.2}$$
$$x_2 \leq 6 \tag{6.3}$$
$$3x_1 + 4x_2 \leq 36 \tag{6.4}$$
$$x_1 \geq 0 \tag{6.5}$$
$$x_2 \geq 0 \tag{6.6}$$

where (6.1) defines the objective function (divided by 1,000), (6.2) defines the maximum number of trucks of the first type that can be purchased, (6.3) defines the maximum number of trucks of the second type that can be purchased, and (6.4) ensures the purchase of trucks as long as the company remains within the budget (akin to (6.1), this constraint has been divided by 10,000). We can visualize the problem in the (x_1, x_2) plane as shown in Figure 6.1.

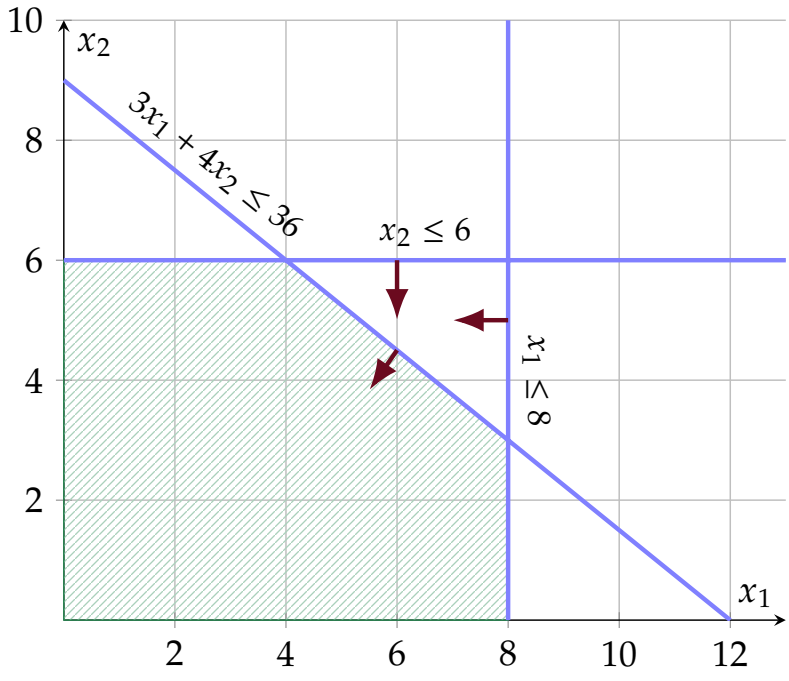


Figure 6.1: Feasible region for the street food company problem of Example 6.1.

The three constraints (6.2)-(6.4) are shown with full blue lines. The arrow perpendicular to each blue line represents the half-plane where the inequality holds, i.e., where the constraint is satisfied. Only the first quadrant is shown because of the non-negativity requirement on x_1 and x_2 due to (6.5)-(6.6). The green-shaded area is the portion of the (x_1, x_2) plane where all constraints are simultaneously satisfied, also known as **feasible region** .

Our goal is to determine which point, inside this region, maximizes our objective as defined by (6.1). To achieve such a goal, in this specific case, we can leverage the fact that our solution is defined in the (x_1, x_2) two-dimensional space, and we can re-write Equation 6.1 as

$$x_2 = -\frac{2}{5}x_1 + \frac{1}{5}Z \tag{6.7}$$

where Z is the objective value. For example, if $Z = 0$ we have the line $x_2 = -\frac{2}{5}x_1$ passing through the origin of the (x_1, x_2) plane, while if $Z = 5$ we have the line $x_2 = -\frac{2}{5}x_1 + 1$ that is parallel to the previous one, but intercepts the x_2-axis in point $(0, 1)$. **Any (x_1, x_2) pair of values on a given line yields the same objective value**. In this instance, the slope of every line remains consistent, set at $-\frac{2}{5}$. Consequently, an increase in Z shifts our line towards the top-right corner of Figure 6.1. Considering this characteristic alongside the shape of the feasible region depicted in Figure 6.1, **determining the optimal solution to our mathematical problem entails identifying the maximum value of Z where the resulting line intersects the feasible region, even if only partially**. We display several lines in orange, for increasing values of Z, in Figure 6.2.

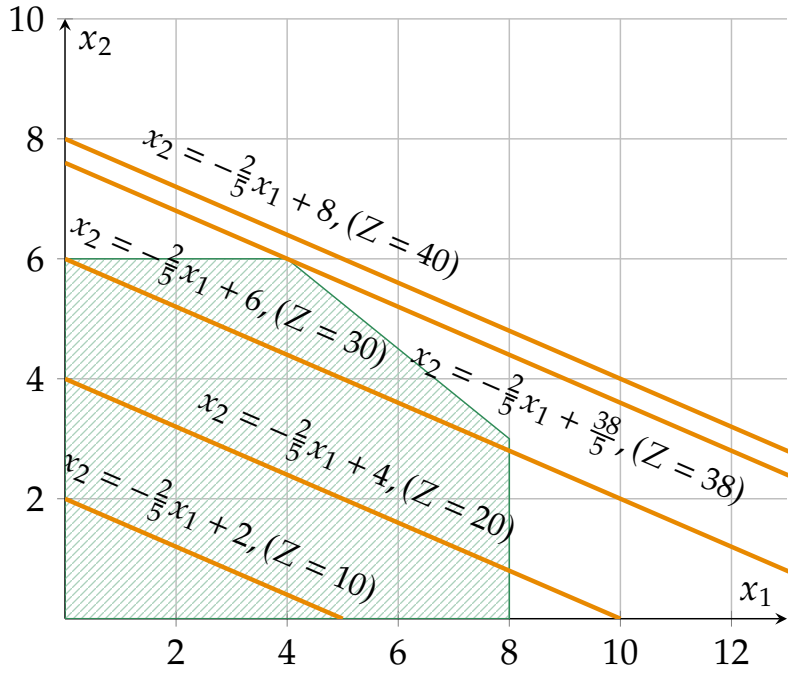


Figure 6.2: Feasible region for the street food company problem of Example 6.1 with some representative lines defining different objective values Z.

For $Z = 10$, $Z = 20$, and $Z = 30$ the line cuts the feasible region, hence providing a wide range of (x_1, x_2) combinations returning a feasible solution characterized by the same objective value. Careful readers might argue that many combinations are not realistic. For example, purchasing $\frac{5}{2}$ trucks of the first type and 3 of the second type, results in a number of customers equal to $2,000 \times \frac{5}{2} + 5,000 \times 3 = 20,000$ (that combination of points belongs to line $x_2 = -\frac{2}{5}x_1 + 4$ from Figure 6.2). Because of the nature of the problem, we clearly cannot purchase $\frac{5}{2}$ trucks of the first type. To address this type of problem, we ask the careful readers to be patient and wait for Chapter 7.

Going back to Figure 6.2, we also notice that for $Z = 40$ the line does not intersect at all the feasible region. Hence, we cannot attract 40,000 customers with the current model. Focusing on the line associated with $Z = 38$, it appears that such a line intersects the feasible region in a single point, i.e., $(x_1, x_2) = (4, 6)$. Luckily, this solution is practically implementable, as the number of trucks to purchase is an integer for both types. **Given the current parameters and, hence, the current slope of our lines representing the relationship between x_1 and x_2 for a given value of the objective Z, $Z = 38$ seems the optimal solution.**

This means we should purchase 4 trucks of the first type and 6 of the second type so that we can attract 38,000 customers. This solution makes sense quantitatively, as trucks of the second type attract more customers (5,000 versus 2,000). Hence, the optimal solution is to purchase as many trucks of the second type as possible, i.e., 6. This results in a cost of $6 \times 40,000 = 240,000$€ leaving the company with 120,000€ that can be used to buy 4 trucks of the first type.

If the interpretation of the optimal solution of Example 6.1 depicted in Figure 6.2 is clear, it also follows that **the optimal solution $(x_1, x_2) = (4, 6)$ does not change if we just slightly change the slope of the line mapping the objective function**. While the objective value will change, point $(x_1, x_2) = (4, 6)$ will be the last point touched by the objective line before leaving the feasible region if we slightly increase or decrease its slope. Let us now re-write the objective in more general terms as $Z = C_1x_1 + C_2x_2$ (so that in the original setting $C_1 = 2$ and $C_2 = 5$). If we decrease the slope of the objective line so that it matches the slope of constraint $3x_1 + 4x_2 \leq 36 \rightarrow x_2 \leq -\frac{3}{4}x_1 + 9$, we obtain the situation shown in Figure 6.3 (in this specific case, $C_1 = \frac{15}{4}$ and $C_2 = 5$).

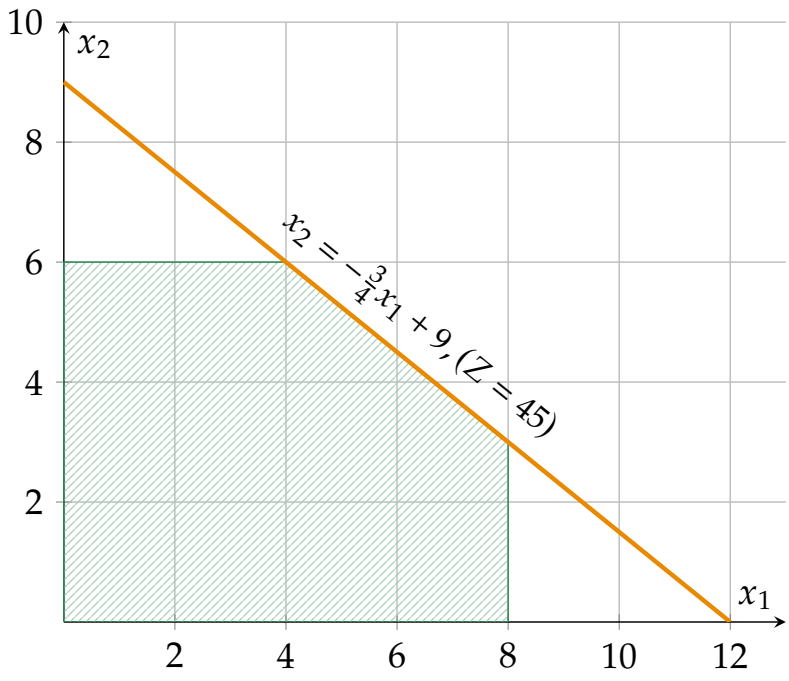


Figure 6.3: Feasible region for the street food problem of Example 6.1 and objective line if C_1 is increased from 2 to $\frac{15}{4}$.

This is a peculiar case where all points along the line $x_2 = -\frac{3}{4}x_1 + 9$ such that $4 \leq x_1 \leq 8$ and $3 \leq x_2 \leq 6$ are mathematically speaking, equally optimal. We then have **multiple optimal solutions**. As already anticipated, the only two points that yield practically feasible solutions are $(x_1, x_2) = (4, 6)$ and $(x_1, x_2) = (8, 3)$. In both cases, the company managed to attract 45,000 customers, which is an improvement over the baseline case of 38,000 because trucks of the first type were made more attractive (by increasing C_1 from 2 to $\frac{15}{4}$). We can verify that both combinations are equivalent in terms of objective as $\frac{15,000}{4} \times 4 + 5,000 \times 6 = 45,000$ and $\frac{15,000}{4} \times 8 + 5,000 \times 3 = 45,000$ as well.

We could take a step further and try to make trucks of the first type even more attractive so that $C_1 > \frac{3}{4}C_2$. For any such combination of (C_1, C_2)

values, now the only optimal solution will be $(x_1, x_2) = (8, 3)$: because the slope of the objective line is now steeper, that is the last point touched by the objective line before leaving the feasible region.

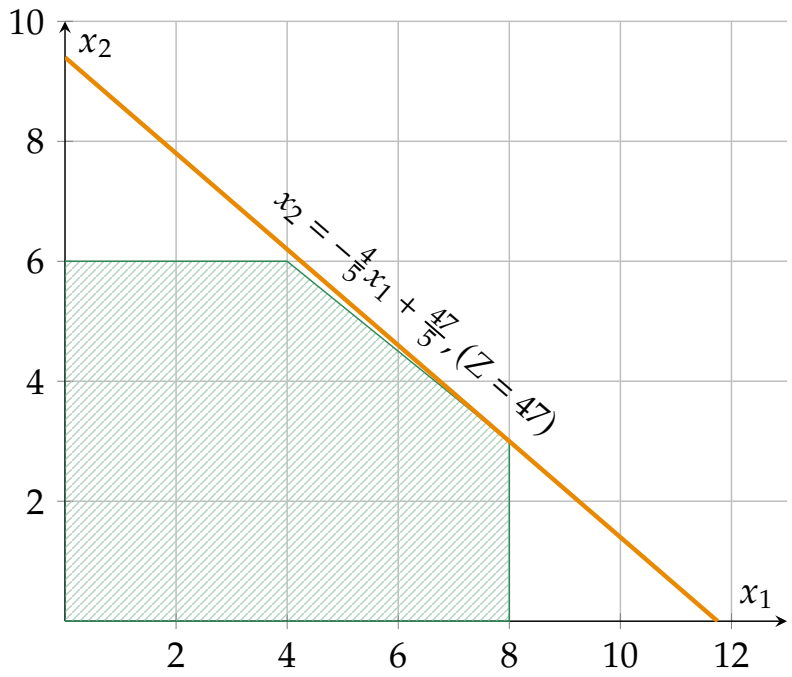


Figure 6.4: Feasible region for the street food company problem of Example 6.1 and objective line if C_1 is increased from 2 to 4.

In Figure 6.4 we show the case where $C_1 = 4$, which yields an optimal solution of 47,000 customers. Note that the new optimal solution suggests purchasing as many trucks of the first type as possible, albeit they are still "less efficient" than trucks of the second type in attracting customers (4,000 versus 5,000 per truck). This is due to the additional two trucks that the company can purchase (8 versus 6).

Note that, for a maximization problem with two-decision variables, the feasible region has generally a shape similar to the one represented in Figure 6.1. Either both decision variables have an upper bound, or they are linked via one constraint in the $\leq$ form. The reason why at least one of the two conditions above should be met is to avoid a feasible region that moves indefinitely toward the upper-right corner, i.e., an **unbounded feasible region**. Imagine, for example, reversing all three constraints of the street food company model from $\leq$ to $\geq$. The associated feasible region is represented in Figure 6.5 .

We could indefinitely increase x_1 and x_2, hence increasing our objective while remaining in the feasible region. Thus, in such a case we can conclude that the optimal solution is $(x_1, x_2) = (\infty, \infty)$ so that $Z = \infty$ (this means that no optimal solution could be found). This is an example of an **unbounded optimization model**. Referring back to Section 1.2, such an occurrence generally entails that the modeler failed to capture some features of the original problem. For example, a maximization problem might entail profit, satisfaction, connectivity, etc. as objectives, while featuring, among other constraints, limits on budget, working hours, etc., which bound the set of decisions that can be taken and hence ensure a bounded feasible region. Note that, while in our example the unbounded feasible region results in no optimal solution as x_1 and x_2 can grow indefinitely, this is not always the case. If we were to

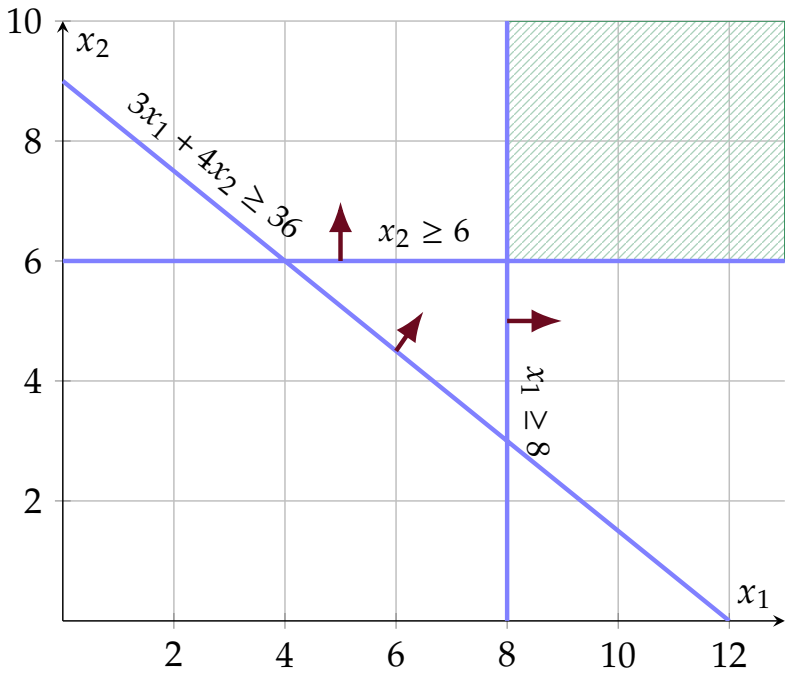


Figure 6.5: Feasible region for the street food company problem of Example 6.1 if all constraints were in the ≥ form.

minimize, rather than optimize, the very mathematical model represented in Figure 6.5, we would get the optimal solution $(x_1, x_2) = (8, 6)$ yielding $Z = 2,000 \times 8 + 5,000 \times 6 = 46,000$. **On a similar note, a minimization problem whose objective function has non-negative terms should never feature a feasible region that contains the origin (regardless of the number of decision variables involved). In such a case, we could set all decision variables to zero, hence obtaining an optimal objective value $Z = 0$. This situation is, again, a potential indication that the developed mathematical model is not correctly mapping the constraints stemming from the real-life problem**. For example, a minimization problem might entail distance, completion time, makespan, etc., as objectives, while featuring, among other constraints, limits on budget, working hours, etc., which bound the set of decisions that can be taken and hence ensure a bounded feasible region. Let us consider the minimization problem of Example 6.2

Example 6.2

$$\min \quad Z = x_1 + 2x_2 \tag{6.8}$$

s.t.:

$$x_1 \leq 10 \tag{6.9}$$
$$x_2 \leq 8 \tag{6.10}$$
$$x_1 + 2x_2 \geq 8 \tag{6.11}$$
$$x_1 \geq 0 \tag{6.12}$$
$$x_2 \geq 0 \tag{6.13}$$

whose feasible region and optimal solution are shown in Figure 6.6.

Because of (6.11), the feasible region does not contain the origin, which means our objective is minimized in point $(x_1, x_2) = (0, 4)$ where $Z = 4$.

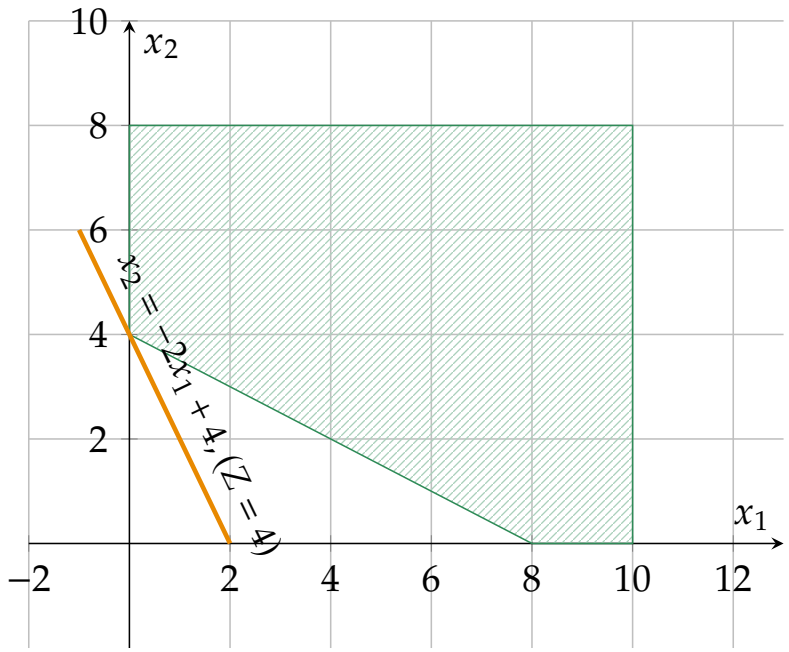


Figure 6.6: Example of feasible region for the minimization problem presented in Example 6.2.

Finally, let us cover one last model, whose graphical solution is meaningful and easy to interpret, in Example 6.3.

Example 6.3 We graphically represent the following maximization problem

$$\max \quad Z = x_1 + x_2 \tag{6.14}$$

s.t.:

$$x_1 + 2x_2 \geq 12 \tag{6.15}$$
$$2x_1 + x_2 \leq 8 \tag{6.16}$$
$$x_1 \geq 5 \tag{6.17}$$
$$x_1 \geq 0 \tag{6.18}$$
$$x_2 \geq 0 \tag{6.19}$$

as shown in Figure 6.7 (note that, in practice, (6.18) is redundant because of the presence of (6.17)). Because no region of the (x_1, x_2) solution space exists where all three functional constraints are simultaneously satisfied, we can already conclude that this is an **infeasible model**.

In this section, we have analyzed, for a two-dimensional optimization problem, how to graphically construct the feasible region and how to determine the optimal solution. We also analyzed how the optimal solution changes with a change in the objective function, and the special case of multiple optimal solutions. We also highlighted the two extreme

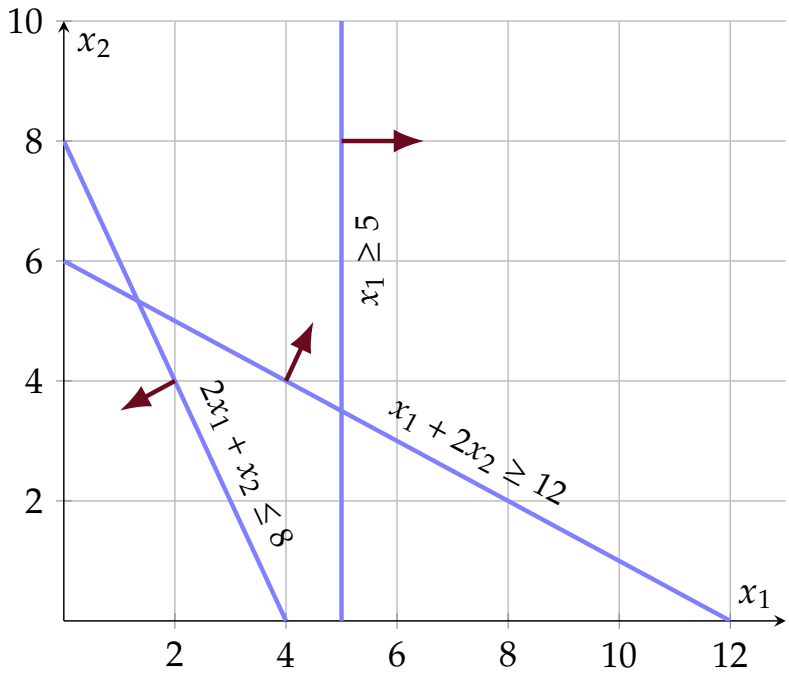


Figure 6.7: Maximization problem of Example 6.3 characterized by the lack of a feasible region.

cases of an unbounded optimization model (due to an unbounded feasible region) and of an infeasible model where no feasible region exists.

When the feasible region exists, our preliminary insight is that the optimal solution to an optimization problem is always located at the intersection of two constraint segments (as was the case for both points $(x_1, x_2) = (4, 6)$ and $(x_1, x_2) = (8, 3)$ in the street food company problem of Example 6.1). This might not be surprising because, in this two-dimensional representation, the objective function is a line and the feasible region is a polygon. Hence, as we try to move the objective line toward the upper-right corner (maximization) or the origin (minimization) of the first quadrant, there is a high chance we intersect a final **corner point** before leaving the feasible region. In the street food company problem of Example 6.1, we characterize a corner point as every intersection of any two constraints that characterize our optimization problem.

Considering that $n = 2$ (two decision variables) and $m = 3$ (three functional constraints), we have $n + m$ overall constraints (we also need to consider the non-negativity constraints $x_1 \geq 0$ and $x_2 \geq 0$). Hence, in this example, the number of expected corner points is equal to the number of combinations of two constraints out of the five available, namely $C_2^5 = \frac{5!}{(5-2)! \times 2!} = 10$. We highlight all the corner points of the street food company problem in Figure 6.8 with gray or red circles.

It can be noted that only 8 corner points are present in Figure 6.8 and not 10. In our case, it can be noted how constraint $x_2 \leq 6$ is parallel to the x_1 axis (that maps the $x_1 \geq 0$ constraint), and hence no corner point is associated with this pair of constraints. Similarly, no corner point is associated with the intersection of constraints $x_1 \leq 8$ and $x_2 \geq 0$. Hence, we only have 8 corner points out of the 10 theoretically available. In general, this might be expected as:

- some constraints might never intersect with each other (as shown

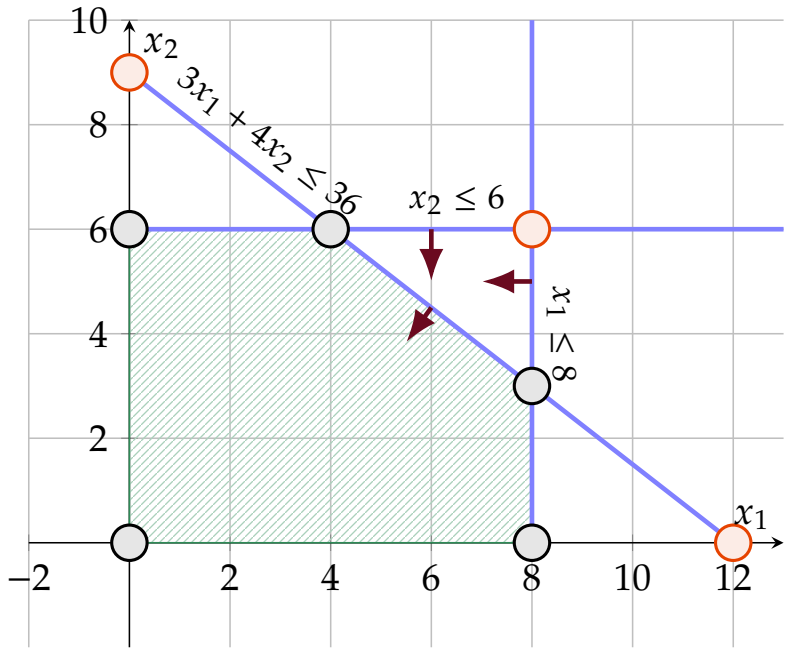


Figure 6.8: Corner points for the street food company example.

in Figure 6.8);

- some corner points might be the intersection of more than two constraints.

Figure 6.8 highlights corner points in two distinct colors for a specific reason. Gray corner points lie on the feasible region and are hence defined **feasible corner points** because they represent combinations of the decision variables that satisfy all the constraints, hence yielding a feasible solution. Red corner points do not lie on the feasible region and are hence defined **infeasible corner points** because they represent combinations of decision variables where at least one constraint is violated.

We anticipated that we could use an ad-hoc algorithm to find the solution to an LP in an equivalent fashion to what we did graphically for the street food company case. Because such an algorithm resembles the solution to a linear system, which is generally written in the equality form $Ax = b$, we first need to convert all constraints into equality form. This also entails that the corner points we intuitively showed in Figure 6.8 will undergo some form of modification to ensure the aforementioned equality form. The description of how to adapt an optimization model so that it is in a form that is suitable for such an algorithm to work, i.e., the **augmented form of an LP**, is described in Section 6.2 while we will elaborate how corner points are handled by the algorithm in Section 6.3.2.

6.2 Augmented form of an LP

When dealing with an LP, constraints can appear in three forms, namely $\leq$, $=$, or $\geq$. We will tackle each constraint type separately when describing how to convert them into augmented[1] form, as they need to be treated differently. In general, the augmented form of an LP must satisfy the following two conditions:

1: In some other references, *augmented* could be replaced by the term *proper* or *standard*.

- all constraints should be converted into equality constraints. This is achieved by **introducing additional decision variables to the original model**;
- if the original decision variables of the LP are $x_1, x_2, \cdots, x_n$, the augmented form of the LP should be initialized in a way that allows to start the algorithm from the (mathematically) feasible corner point $(x_1, \cdots, x_n) = (0, \cdots, 0)$. This is a key step to **initialize the algorithm that moves across corner points until the optimal one is identified**.

To meet both conditions, it is essential to address the three types of constraints with slight variations in treatment.

6.2.1 Inequality constraints in the $\leq$ form

The $\leq$ is the simplest constraint type to be put in augmented form. Let us consider a generic constraint

$$\sum_{i=1}^{n} C_i x_i \leq b \tag{6.20}$$

We can transform (6.20) into proper form by adding decision variable x_s to the left-hand side and replacing the inequality sign with an equality sign as follows

$$\sum_{i=1}^{n} C_i x_i + x_s = b \tag{6.21}$$

The role of x_s in (6.21) is quite intuitive. Because the original left-hand side $\sum_{i=1}^{n} C_i x_i$ cannot be greater than b, the value of x_s in (6.21) is the difference between b and $\sum_{i=1}^{n} C_i x_i$. **If we assume b to be some form of capacity (e.g., budget), x_s maps how much of that capacity we are not using**. Every decision variable that is equivalent in form to x_s is called a **slack variable** because it compensates the slack the left-hand side is missing to reach the right-hand side value. If we consider (6.4) from Example 6.1, i.e., the budget constraint, we can re-write that constraint as

$$3x_1 + 4x_2 + x_s = 36 \tag{6.22}$$

In the optimal solution, the company purchased 4 trucks of the first type and 6 of the second type, hence using a budget of $30,000 \times 4 + 40,000 \times 6 = 360,000$€. In such a case, we have that $x_s = 0$ because all the budget available has been used. If the company were to purchase just 2 trucks of the first type and 3 of the second type, it would use a budget of $30,000 \times 2 + 40,000 \times 3 = 180,000$€ hence $x_s = 18$ (recall we divided all terms of this constraint by 10,000) to highlight that we still have 180,000€ left in terms of budget. We can apply the same logic to all the functional constraints of the street food company problem, yielding the following augmented form:

$$\max \; 2x_1 + 5x_2 + 0x_3 + 0x_4 + 0x_5 \tag{6.23}$$

s.t.:

$$x_1 + x_3 = 8 \tag{6.24}$$
$$x_2 + x_4 = 6 \tag{6.25}$$
$$3x_1 + 4x_2 + x_5 = 36 \tag{6.26}$$
$$x_1, x_2 \geq 0 \tag{6.27}$$
$$x_3, x_4, x_5 \geq 0 \tag{6.28}$$

where x_3, x_4, and x_5 are the additional slack variables which should also be non-negative, as highlighted by (6.28). We notice that now this revised model is in augmented form. All functional constraints are = constraints and we can set $x_1 = 0$ and $x_2 = 0$ to obtain a feasible corner point where to start the algorithm. This is possible because x_3, x_4, and x_5 take up all the slack in Equation 6.24, Equation 6.25, and Equation 6.26 respectively by taking the values $x_3 = 8$, $x_4 = 6$, and $x_5 = 36$. **Readers may contend that the starting point seems suboptimal since it involves purchasing no trucks and consequently yielding no profit. While valid, we will delve into the rationale behind this observation in Section 6.3.**

A few important takeaways are:

- adapting an LP in augmented form increases the overall number of decision variables. For the street food company case of Example 6.1, we went from two to five decision variables, for example;
- related to the previous point, the solution space increases, as it becomes a five-dimensional space in the street food company case of Example 6.1;
- while we increase the number of decision variables, slack variables do not appear in the objective function. **Because they are additional variables that are needed for algorithmic purposes, they play no role in the objective which, instead, is based on the original decision variables that relate to the original practical problem at hand. Conversely, they have a meaning if we consider the constraints, as they identify which constraints are "saturated" and which are not**;
- to start the algorithm looking for the optimal corner point, we mentioned it is good practice to start with a corner point where all the original decision variables are set to zero. This implies that the initial objective is $Z = 0$, which is the worst initial objective value for a max problem with positive coefficients in the objective function. Notwithstanding, it is a feasible initial solution to start the algorithm. Using the slack variables, we can achieve this goal for Example 6.1 because the solution $(x_1, x_2, x_3, x_4, x_5)$, $(0, 0, 8, 6, 36)$ defines a feasible corner point where $Z = 0$ because both x_1 and x_2 are 0.

6.2.2 Equality constraints

Equality constraints might seem a trivial case, as they are already in the right (=) form. Readers should not forget that a second condition should hold, i.e., that we want to initialize our algorithm from a feasible corner point where all original decision variables are set to 0. Let us analyze the street food company case of Example 6.1 with a slight variation to address this case. In particular, let us replace (6.6) with the same constraint in = form.

In such a case, the feasible region changes and turns into the segment highlighted in Figure 6.9: any feasible solution must lie on the $3x_1 + 4x_2 = 36$ line while being bounded by the constraints $x_1 \leq 8$ and $x_2 \leq 6$. It is also relevant to note that the optimal solution to this problem does not change, as point $(x_1, x_2) = (4, 6)$ lies on the segment.

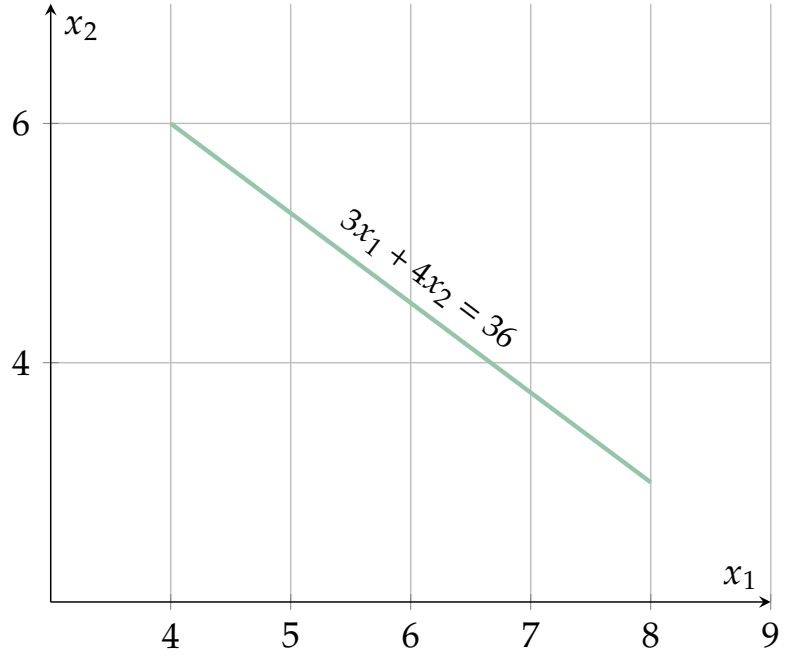


Figure 6.9: Feasible region for the street food company case of Example 6.1 if inequality constraint (6.6) is turned into an equality constraint.

Having two inequality constraints in the ≤ form and an equality constraint, we might be tempted to use the same logic shown in Section 6.2.1 for the inequality constraints and leave the equality constraint unchanged. This would lead to the following revised model:

$$\max \quad 2x_1 + 5x_2 \tag{6.29}$$

s.t.:

$$x_1 + x_3 = 8 \tag{6.30}$$
$$x_2 + x_4 = 6 \tag{6.31}$$
$$3x_1 + 4x_2 = 36 \tag{6.32}$$
$$x_1, x_2 \geq 0 \tag{6.33}$$
$$x_3, x_4 \geq 0 \tag{6.34}$$

While all constraints are now in the equality form, we cannot define a feasible solution where the original decision variables x_1 and x_2 are 0. Constraints (6.30)-(6.31) are satisfied if we set $x_3 = 8$ and $x_4 = 4$, respectively. On the other hand, constraint (6.32) is not satisfied with such a choice as $3 \times 0 + 4 \times 0 = 0 \neq 36$.

To solve this issue, we introduce an additional decision variable x_a in (6.32), that we define **artificial variable**, as follows: $3x_1 + 4x_2 + x_a = 36$. It serves a different purpose than the slack variable x_5 we used in Section 6.2.1 because **if it takes a non-zero value, it means the original constraint is not satisfied and we are dealing with an infeasible solution**. To deal with this shortcoming, we modify the objective function by introducing a penalty in the form of $-Mx_a$, where M is a large enough big-M as shown before (e.g., Section 4.8.1) in this book. This is an example of a soft constraint where we allow constraint (6.32) to be violated, but we considerably decrease the objective (for a max problem, a penalty entails reducing the objective). Hence, the correct augmented form of this LP is:

$$\max \quad 2x_1 + 5x_2 - Mx_a \tag{6.35}$$

s.t.:

$$x_1 + x_3 = 8 \tag{6.36}$$
$$x_2 + x_4 = 6 \tag{6.37}$$
$$3x_1 + 4x_2 + x_a = 36 \tag{6.38}$$
$$x_1, x_2 \geq 0 \tag{6.39}$$
$$x_3, x_4, x_a \geq 0 \tag{6.40}$$

Now, we can set $x_1 = 0$ and $x_2 = 0$ while satisfying (6.38) by setting $x_a = 36$. The price we pay is that now our initial objective is not $Z = 0$ but $Z = -36M$. We achieved our goal of initializing our problem from corner point $(x_1, x_2) = (0, 0)$ in a way that is mathematically feasible, but the activation of the penalty in the objective is a red light that, actually, we are violating a constraint in reality. This is (generally) not a problem as the solution algorithm will soon move away from this corner point, as we will show in Section 6.3.

6.2.3 Inequality constraints in the $\geq$ form

Finally, we deal with the $\geq$ case. At first glance, this constraint type might seem very similar to the $\leq$ counterpart. While, with a constraint in the $\leq b$, we cannot exceed the right-hand side b, with a constraint in the $\geq b$ form, we cannot go below the right-hand side b. Such a constraint generally represents a minimum supply level, quality level, grade, etc., that we need to achieve to satisfy a practical constraint of our real-life problem. Hence, we might be tempted to use a similar approach to a slack variable, but a simple example will show why the $\geq$ is different from the $\leq$ case.

Let us consider a course whose final grade is the weighted average of a written test and a group assignment. We define x_1 as the mark of the test and x_2 as the mark of the assignment and assume they contribute equally to the final grade. We assume a grading scale up to 10 (for both marks and the final grade) and that we need a full 6 to pass the course. Hence, we can map the successful completion of the course with the following constraint

$$\frac{1}{2}x_1 + \frac{1}{2}x_2 \geq 6 \tag{6.41}$$

which imposes that the weighted average of the two marks should be at least 6. We could translate the inequality into equality by introducing a **surplus variable** x_{sp} as follows

$$\frac{1}{2}x_1 + \frac{1}{2}x_2 - x_{sp} = 6 \tag{6.42}$$

We notice that x_{sp} appears with a minus in front. This is correct because of the original $\geq$ sign. **The original left-hand side ($\frac{1}{2}x_1 + \frac{1}{2}x_2$) is allowed to exceed the right-hand side and, if so, x_{sp} cancels that surplus to ensure the equality holds**. Let us assume we obtained an 8 both in the exam and assignment. (6.42) needs to be $\frac{1}{2}8 + \frac{1}{2}8 - 2 = 6$ to be satisfied: $x_{sp} = 2$ maps the fact that we are happily two full marks above the minimum passing grade and takes the needed surplus value to ensure the equality holds.

We successfully converted an inequality into equality, but we are facing issues if we want to initialize our algorithm with an initial solution where all the original decision variables are set to 0. In fact, considering (6.43)

$$\frac{1}{2}0 + \frac{1}{2}0 - x_{sp} = 6 \rightarrow x_{sp} = -6 \tag{6.43}$$

the only way to satisfy such equality with $(x_1, x_2) = (0, 0)$ is to set $x_{sp} = -6$, **which is mathematically infeasible as the surplus variable, like every other decision variable we deal with, should be non-negative**. To solve the issue, we use the same approach we introduced in Section 6.2.2 and add an artificial variable x_a to the constraint as follows

$$\frac{1}{2}x_1 + \frac{1}{2}x_2 - x_{sp} + x_a = 6 \tag{6.44}$$

and a $-Mx_a$ term to the objective in case of a maximization problem or a Mx_a term in case of a minimization problem. In our example, (6.44) can now be satisfied by selecting $x_1 = x_2 = x_{sp} = 0$ and $x_a = 6$: this is a reminder that this choice is actually infeasible because we are not satisfying the requirements to pass the exam. Hence, in such a setting, x_a will only be activated if the original left-hand side is smaller than the right-hand side (to ensure the equality holds), which defines an infeasible situation as mapped by the activation of the penalty in the objective. **This is generally the case during the initialization of the algorithm that will be explained in Section 6.3. Once the left-hand side of an original $\geq$ constraint is greater or equal to the right-hand side, the artificial variable is not needed and the surplus one will be activated instead**.

6.2.4 Final remarks

In this section, we highlighted how the constraints of an optimization model should all be converted to the = form so that a proper algorithm can identify the optimal solution to such a problem by moving across corner points. Assuming a max problem, we generally want to start from the corner point where **all the original decision variables are set to 0**. If we combine these requirements, we have that

- a $\leq$ constraint is put into augmented form by adding a slack variable x_s to the left-hand side and replacing the $\leq$ with =;
- a = constraint is put into augmented form by adding an artificial variable x_a to the left-hand side and a $-Mx_a$ term to the objective (for a max problem) and a Mx_a term to the objective (for a min problem);
- a $\geq$ constraint is put into augmented form by adding both a surplus variable x_{sp} (with a minus in front) and an artificial variable x_a to the left-hand side and a $-Mx_a$ term to the objective (for a max problem) and a Mx_a term to the objective (for a min problem).

6.3 The simplex method: description of the algorithm

In Section 6.2 we described how a generic LP needs to be converted into an augmented form so that an ad-hoc algorithm can move across corner points and identify the optimal one. In this section, we finally reveal the details of such an algorithm, i.e., the **simplex algorithm**. In geometry, a *simplex* is a generalization of the notion of a triangle or tetrahedron to arbitrary dimensions. Such a name is fitting for the simplex algorithm, because it navigates across different corner points of the feasible region of a problem looking for the optimal one.

6.3.1 Basic and non-basic variables

According to linear algebra, we can generally solve linear systems in the $Ax = b$ if A is a square and invertible matrix[2]. In Section 6.2 it was explained how bringing an LP in augmented form increases the number of decision variables because, on top of the original ones, we must add slack, artificial, and surplus decision variables (depending on the type of constraint). Because of this insight, the overall number of decision variables N for a given optimization problem is greater or equal to the number of original decision variables n plus the number of functional constraints m[3].

If we forget for a moment about the objective function, the insight implies that the $Ax = b$ system where:

- x defines the vector with the full set of decision variables;
- A defines the coefficient matrix;
- b defines the vector containing the right-hand sides of all functional constraints;

2: There might be exceptions to this statement, but let us consider this as the benchmark here.

3: As a reminder, the functional constraints of an LP are all the constraints characterizing it apart from the non-negativity constraints of all decision variables. In Example 6.1, the three functional constraints are (6.2)-(6.4).

is generally "horizontal" with A not being square and with more columns (decision variables) than rows (constraints). Let us consider the original street food company problem of Example 6.1. We can map its augmented form as

$$\underbrace{\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 3 & 4 & 0 & 0 & 1 \end{pmatrix}}_{A=(3\times5)} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}}_{x=(5\times1)} = \underbrace{\begin{pmatrix} 8 \\ 6 \\ 36 \end{pmatrix}}_{b=(3\times1)} \tag{6.45}$$

We cannot solve this linear system as it is, but we need to fix two decision variables so that the resulting system is (3×3) and hence solvable (recall Chapter 3). For example, if we set $x_1 = 0$ and $x_2 = 0$ we eliminate the first two columns from A in (6.45) and x_1, x_2 from x, hence obtaining the reduced linear system

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 3 & 4 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cancel{x_1} \\ \cancel{x_2} \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ 36 \end{pmatrix} \tag{6.46}$$

so that

$$\begin{pmatrix} x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 8 \\ 6 \\ 36 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ 36 \end{pmatrix} \tag{6.47}$$

Note that, in this particular case, the values of x_3, x_4, and x_5 are directly the 3 values stored in the b vector (in the original order) because the reduced A was the identity matrix. The identity matrix is a special case of **orthonormal basis**, i.e., a square matrix where rows/columns are unit vectors and are all orthogonal to each other. Let us consider the following two examples

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ 36 \end{pmatrix} \rightarrow \begin{pmatrix} x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 8 \\ 6 \\ 36 \end{pmatrix} \rightarrow \begin{pmatrix} x_4 \\ x_3 \\ x_5 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ 36 \end{pmatrix} \tag{6.48}$$

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ 36 \end{pmatrix} \rightarrow \begin{pmatrix} x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}^{-1} \begin{pmatrix} 8 \\ 6 \\ 36 \end{pmatrix} \rightarrow \begin{pmatrix} x_5 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ 36 \end{pmatrix} \tag{6.49}$$

which depict different cases of orthonormal matrices. In (6.48), the x vector is re-shuffled into $(x_4, x_3, x_5)^T$, while in (6.49), the x vector is re-shuffled into $(x_5, x_3, x_4)^T$. **Notwithstanding, each decision variable**

is directly mapped into one element of b in all the examples shown because we chose special orthonormal matrices where each row/column features a single 1 and 0s anywhere else.

We are now ready to introduce the concept of basic and non-basic variables in the context of the simplex algorithm. Given an optimization problem with N overall decision variables and m functional constraints, we have that:

- m decision variables can take non-zero values. These decision variables are called **basic variables** because they are pre-multiplied by an orthonormal matrix with only $0/1$ values and, hence, their values are directly mapped by the current b vector as shown in (6.47), (6.48), or (6.49);
- the remaining $N - m$ decision variables are arbitrarily set to 0 so that the resulting system is square and invertible (and hence the aforementioned basic variables can be computed). These are defined **non-basic variables**.

We already provided some insights into the initial selection of basic and non-basic decision variables, and we will shortly explain how to iteratively modify this choice, given a specific problem at hand, looking for an optimal solution. Note that a specific choice of basic and non-basic decision variables identifies a specific corner point as introduced in Section 6.1. Hence, anytime we change such a choice we are moving our solution to a different (hopefully better) corner point.

6.3.2 Basic solutions

We now expand the concept of corner points we intuitively hinted at in Section 6.1. In Section 6.3.1, we discussed how a model with n original decision variables and m functional constraints ends up having $N \geq n+m$ overall decision variables in the augmented form (we need to add one extra variable for every constraint in the $\leq$ and $=$ form and two extra variables for every constraint in the $\geq$ form). Because the underlying set of linear equations is not square ($N > m$), it can only be solved if we assign to $N - m$ decision variables an arbitrary value (non-basic variables), which we decide to be 0, so that we can solve the remaining $(m \times m)$ square system (basic variables).

Given a problem in augmented form, readers might be wondering how many different options exist to select m basic variables (and hence $N - m$ non-basic decision variables). Similar to what we did in Section 6.1, we need to compute

$$C_m^N = \frac{N!}{(N-m)!m!} \tag{6.50}$$

Expression (6.50) defines the number of **basic solutions** characterizing an LP, where **a basic solution is a solution of the augmented LP with m basic variables and $N - m$ non-basic variables**. Not all basic solutions are feasible (recall that all basic variables should be non-negative), hence we distinguish between **basic feasible solutions** and **basic infeasible solutions**. In addition, each basic solution is associated with a specific corner point as defined by the original set of n decision variables.

Let us consider the original street food company problem of Example 6.1 and its corner points as shown in Figure 6.8. We also discussed its augmented form in Section 6.2.1. Corner point $(x_1, x_2) = (0, 6)$ (which is feasible because it lies on the boundary of the feasible region) is associated with the basic feasible solution $(x_1, x_2, x_3, x_4, x_5) = (0, 6, 8, 0, 12)$. In such a solution, x_1 and x_4 are non-basic, which allows solving the resulting (3×3) system in x_2, x_3, and x_5.

Given expression (6.50), we might consider fully enumerating all combinations, solving all the resulting linear systems, and picking the basic feasible solution with the highest value if we are dealing with a max problem. This is of course possible for small-scale models, but full enumeration becomes challenging as the size of the model increases. Let us consider $n = 50$, $m = 70$, and $N = 120$. These values identify a model larger than Example 6.1, but very small compared to many models used for real applications. Using (6.50) we get $C_{70}^{120} = \frac{120!}{50!70!} \simeq 1.8 \times 10^{34}$: even for a relatively small problem, the number of options to fully consider explodes.

Luckily for us, we can leverage a property of the corner points (and associated basic solutions) within the context of the simplex method. Let us consider again the feasible region of the street food company (Figure 6.8), and let us focus on feasible corner points A $(x_1, x_2) = (0, 0)$, B $(x_1, x_2) = (8, 0)$, and C $(x_1, x_2) = (0, 6)$. The associated basic feasible solutions are, respectively, $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 8, 6, 36)$, $(x_1, x_2, x_3, x_4, x_5) = (8, 0, 0, 6, 12)$, and $(x_1, x_2, x_3, x_4, x_5) = (0, 6, 8, 0, 12)$ with objectives $Z_A = 0$, $Z_B = 16$, and $Z_C = 30$. From corner point A, we have two **adjacent corner points** (B and C), i.e., the two corner points that we can reach from A by moving along one edge of the feasible region. If we focus on the associated basic feasible solutions, we realize that going from A to B, we swap a basic variable with a non-basic (and vice versa) and that the same happens if we move from A to C.

If we move from A to B, x_1 switches from being basic to non-basic, with x_3 switching from non-basic to basic. The other original non-basic variable, i.e., x_2, retains its value of 0. The other two basic variables (x_4 and x_5) retain their non-negative values. Note that the value of x_4 is unchanged: we are still not investing in trucks of the second type, hence in the associated constraint the slack variable x_4 retains its value of 6. Conversely, x_5 reduces from 36 to 12 (as already explained) because in B part of the budget is used. **This is also consistent with the concept of feasible corner point and associated basic solution. When moving from a feasible corner point to an adjacent feasible corner point, we swap one basic decision variable with a non-basic one in the process. Because both corner points are associated with basic solutions, we still need to ensure that m variables are basic and the remaining $N - m$ non-basic to have a square system we can solve. Hence, if we consider all the remaining non-basic variables that were not swapped as part of the move, they should retain their 0 value. For the remaining basic variables that were not swapped as part of the move, we might (and most likely see for some) changes in their value because the corner point maps a different system of equations**.

For the sake of completeness, we briefly consider what happens when we move from A to C. x_2 switches from being non-basic to basic, with

x_4 switching from basic to non-basic. x_1 retains its 0 value, x_3 is still basic with an unchanged value, while x_5 is reduced from 36 to 12 (again because now part of the budget is used).

If we recall that $Z_A = 0$, $Z_B = 16$, and $Z_C = 30$, **it seems logical that, assuming we are positioned in A, the smartest move in Figure 6.8 is to go from A $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 8, 6, 36)$ to C $(x_1, x_2, x_3, x_4, x_5) = (0, 6, 8, 0, 12)$ by swapping with non-basic variable x_2 (hence making it basic) with the basic variable x_4 (hence making it non-basic) because we increase our objective the most**. This intuition is the very cornerstone of the simplex method, as explained in the **Basic intuition behind the simplex method** box.

Basic intuition behind the simplex method

Given an original LP and its augmented form, the simplex method entails the following steps for a max problem:

- **identify a initial corner point** and the associated basic solution;
- **check all the adjacent feasible corner points**, identify the one where the associated basic solution **yields the highest increase in the objective**, and **move there** (the "movement" entails swapping a basic variable with a non-basic one);
- repeat the process until a **feasible corner point** is reached where all adjacent points are associated with basic solutions with a worse (lower) objective. This feasible corner point is the **optimal** one.

This intuition also highlights that if we are very smart (or lucky) with the choice of the initial corner point, we might need no iterations at all, as our current corner point is already the optimal one. Considering the original street food company problem of Example 6.1 and its feasible region in Figure 6.8, we graphically assessed that the optimal solution is $x_1 = 4$ and $x_2 = 6$. Hence, the optimal corner point is $(x_1, x_2) = (4, 6)$ and the optimal basic solution is $(x_1, x_2, x_3, x_4, x_5) = (4, 6, 4, 0, 0)$ where x_1, x_2, and x_3 are the basic variables and x_4 and x_5 are non-basic. Let us label this corner point D so that $Z_D = 36$. Let us also label the fifth and remaining feasible corner point $(x_1, x_2) = (8, 3)$ E, whose basic feasible solution is $(x_1, x_2, x_3, x_4, x_5) = (8, 3, 0, 3, 0)$ and $Z_E = 31$. If we select D as our initial corner point, both neighboring corner points B and E are characterized by a worst objective value. Hence, we are currently located on a corner point whose adjacent corner points all feature a worse objective. This guarantees the basic solution associated with the current corner point is the optimal solution[4].

4: For more technical details on why this guarantees that the current corner point is the best (optimal) one we refer readers to Hillier and Lieberman (2015).

6.3.3 The simplex tableau

The last step before formalizing the intuition shown in Section 6.3.2 into a proper algorithmic setting entails introducing the **simplex tableau**. The tableau is a special matrix that captures all the information of an LP in augmented form, considering both the objective function, the functional constraints, the basic and non-basic decision variables. In addition, **such a tableau can be iteratively updated with elementary row operations**

(as hinted at in Chapter 3) while looking for the optimal solution to a given optimization problem.

Using, as usual, the street food company problem of Example 6.1 as a reference, we have already shown that its LP in augmented form is:

$$\max \; Z = 2x_1 + 5x_2 + 0x_3 + 0x_4 + 0x_5 \tag{6.51}$$

s.t.:

$$x_1 + x_3 = 8 \tag{6.52}$$
$$x_2 + x_4 = 6 \tag{6.53}$$
$$3x_1 + 4x_2 + x_5 = 36 \tag{6.54}$$
$$x_1, x_2, x_3, x_4, x_5 \geq 0 \tag{6.55}$$

Such a problem can be initialized with the basic feasible solution $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 8, 6, 36)$ in a simplex tableau as shown in Table 6.1.

Table 6.1: Initial simplex tableau for the street food company problem of Example 6.1.

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	-2	-5	0	0	0	0
(x_3)	0	1	0	1	0	0	8
(x_4)	0	0	1	0	1	0	6
(x_5)	0	3	4	0	0	1	36

Note that Table 6.1 carries more information than (6.51)-(6.55), because it maps a feasible corner point (and hence a basic feasible solution to the problem). In this case, the current basic feasible solution is, as mentioned already above, $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 8, 6, 36)$

We now proceed to explain all the highlighted blocks of Table 6.1:

- : row vector containing the labels of the objective function Z and all the decision variables. It highlights which column is associated with which element in the tableau and is used for the sake of readability and interoperability of the tableau;
- : objective row. It stores the coefficient multiplying the objective function Z (fixed and equal to 1) and the **reduced cost** of each decision variable. At every iteration of the simplex algorithm, the reduced cost of non-basic variables should be 0. More will be discussed about reduced costs in Section 6.3.4[5];
- : the current value of Z;
- : coefficients of the non-basic variables in the functional constraints. They are highlighted as two distinct columns (for Example 6.1, but they are $N - m$ in general) because they are not necessarily contiguous, as we shall see soon;
- : coefficients of the basic variables in the functional constraints. They are highlighted as three distinct columns (for Example 6.1, but they are m in general), similar to the non-basic ones, because they are not necessarily contiguous, as we shall see soon. In addition, note

5: For a thorough analysis of the reduced cost, we refer readers to Hillier and Lieberman, 2015.

that the three columns form an orthonormal basis as mentioned above;
: column vector containing the values of the current basic variables.

We now analyze more in detail how to read the tableau shown in Table 6.1. The objective row tells us that $x_1 = 0$ and $x_2 = 0$ because they have non-zero coefficients there[6]. Because each coefficient in the tableau is multiplied by the associated label right above as stored in , Table 6.1 is equivalent to (6.56):

6: We will shortly elaborate that, in a simplex tableau, all non-basic decision variables should have a non-zero coefficient in the objective row and all basic variables a zero cofficient instead.

$$\begin{cases} 1 \times Z - 2 \times x_1 - 5 \times x_2 + 0 \times x_3 + 0 \times x_4 + 0 \times x_5 = 0 \\ 0 \times Z + 1 \times x_1 + 0 \times x_2 + 1 \times x_3 + 0 \times x_4 + 0 \times x_5 = 8 \\ 0 \times Z + 0 \times x_1 + 1 \times x_2 + 0 \times x_3 + 1 \times x_4 + 0 \times x_5 = 6 \\ 0 \times Z + 3 \times x_1 + 4 \times x_2 + 0 \times x_3 + 0 \times x_4 + 1 \times x_5 = 36 \end{cases} \tag{6.56}$$

which, using the information that $x_1 = 0$ and $x_2 = 0$, becomes

$$\begin{cases} 1 \times Z - 2 \times 0 - 5 \times 0 + 0 \times x_3 + 0 \times x_4 + 0 \times x_5 = 0 \\ 0 \times Z + 1 \times 0 + 0 \times 0 + 1 \times x_3 + 0 \times x_4 + 0 \times x_5 = 8 \\ 0 \times Z + 0 \times 0 + 1 \times 0 + 0 \times x_3 + 1 \times x_4 + 0 \times x_5 = 6 \\ 0 \times Z + 3 \times 0 + 4 \times 0 + 0 \times x_3 + 0 \times x_4 + 1 \times x_5 = 36 \end{cases} \rightarrow \begin{cases} Z = 0 \\ x_1 = 0 \\ x_2 = 0 \\ x_3 = 8 \\ x_4 = 6 \\ x_5 = 36 \end{cases} \tag{6.57}$$

The information contained in (6.57) is the same information contained in the tableau of Table 6.1, but the latter is more compact and better suited for the algebraic operations that will be discussed in Section 6.3.4.

We provide a final note on the format of Table 6.1, and its leftmost and rightmost columns specifically. The rightmost column (which we labeled R.H.S. as it contains the right-hand side of the objective and the functional constraints) maps the objective value and the values of the basic variables for the current iteration. In the leftmost column, we highlight which basic variable the associated row refers to, so that, for example, we can immediately associate x_3 with a value of 8, etc., in Table 6.1. This is done just for visual purposes, as that column is formally redundant. **To identify which basic decision variable is associated with which value of the rightmost column, we just need to find where the unitary value in the orthonormal basis appears**. For x_3, the 1 appears in the first row (note: we consider the objective row separately, as also highlighted by the horizontal separation line between it and the bottom part of the tableau), hence $x_3 = 8$. For x_4, the 1 appears in the second row, hence $x_4 = 6$. For x_5, the 1 appears in the third row, hence $x_5 = 36$.

We will now explain how to start from an initial tableau, like the one shown in Table 6.1, and how to perform elementary row operations to move to a different (better) corner point and how to assess when to stop because an optimal solution has been found.

6.3.4 The simplex algorithm: how to iterate and when to stop

In Section 6.3.1-Section 6.3.3 we provided some preliminary notions that are key to understanding the simplex algorithm, namely basic and non-basic variables and how to set up and interpret an initial simplex tableau.

We also discussed how the rightmost column maps the values of the basic variables for the current iteration. Because we deal with non-negative decision variables, some readers might be wondering how to deal with functional constraints with negative right-hand sides which, albeit not common, can be found in some models. In such a case, using the same logic shown in Table 6.1, we would assign a negative value to a basic variable, which is not allowed. The workaround to tackle this issue is very simple, and it entails rearranging the constraint so that it features a positive right-hand side. Let us consider the following constraint

$$3x_1 - 2x_2 \leq -2 \tag{6.58}$$

which, once put in augmented form with a slack variable x_s, might result in $x_s = -2$ if we followed the same initialization logic shown above. We simply need to re-arrange (6.58) as

$$-3x_1 + 2x_2 \geq 2 \tag{6.59}$$

to obtain a constraint that features a non-negative right-hand side. **Note that, due to the change, we are also changing the type of inequality we deal with (from $\leq$ to $\geq$). This also means we will have to add different types of auxiliary decision variables to the constraint (a surplus and an artificial one instead of a slack variable)**. We elaborate on a concept stemming from the above consideration in the **Inequality constraints with a negative right-hand side** box.

Inequality constraints with a negative right-hand side

Inequality constraints in the $\leq$ form with a negative right-hand side are infeasible if all the coefficients on the left-hand side are positive. For example, a constraint such as $3x_1 + 2x_2 \leq -2$ cannot be satisfied because the left-hand side is non-negative. Conversely, inequality constraints in the $\geq$ form with a negative right-hand side are redundant if all the coefficients on the left-hand side are positive. For example, a constraint such as $3x_1 + 2x_2 \geq -2$ is always satisfied because the left-hand side is non-negative. **Hence, such a constraint can be omitted from the model because it will play no binding role**.

Having clarified how to tackle constraints with negative right-hand sides, we now proceed with the description of the simplex algorithm. In Section 6.3.1, we stated how an LP in augmented form with N overall decision variables and m functional constraints can only be solved if we set $N - m$ variables to 0 (non-basic variables) and solve the remaining square system for the remaining m variables (basic variables). The goal of the simplex algorithm, in a nutshell, is the following:

- at every iteration, **identify a non-basic variable that should be "activated" and made basic** and **identify a basic variable that should be "de-activated" and made non-basic** (recall the number of basic variables is fixed and equal to m);
- keep doing so **until there is a need for such a swap**.

We will first analyze how to assess, given a specific tableau, if another iteration is needed and what non-basic variable should become basic. Then, we will describe which basic variable should follow the opposite path and become non-basic, and finally, we will describe how to update the tableau.

6.3.4.1 Identifying the entering basic variable

The identification of the new entering basic variable, i.e., a variable that is currently set to 0 and should take a positive value, is the easiest step of the algorithm. What needs to be done is to check the coefficients of the current non-basic decision variables in the objective row (i.e., the reduced costs) and identify the **most negative** one. The associated decision variable is the one that should become basic. While we leave the mathematical details of the reduced cost outside the scope of the book[7], we can interpret the reduced cost of a non-basic decision variable as the **increase (removing the minus sign from the actual reduced cost) in the objective function for every unit value of increase of such a variable**.

7: We refer readers to Hillier and Lieberman (2015) for a full explanation.

Taking the street food company problem of Example 6.1, in the initial tableau we have that x_1 and x_2 are the non-basic variables, with a reduced cost of -2 and -5 respectively. Note that, at this stage, those reduced costs are the actual coefficients of those variables in the expression of the objective function as shown in (6.23). The explanation, in this case, is quite intuitive. We are purchasing neither trucks of the first nor of the second type. Hence, the additional number of customers we can attract by purchasing trucks of the first type is $2 \times x_1$: we attract 2,000 customers for every truck of first type we decide to buy (recall the division by 1,000). With a similar reasoning, for the second type, we have $5 \times x_2$: we attract 5,000 customers for every purchased truck of the second type.

Hence, in the street food problem, it is recommended to "activate" x_2, hence purchasing some trucks of the second type and exploring a different corner point where x_1 remains 0 and where x_2 is now greater than zero.

6.3.4.2 Identifying the leaving basic variable

To assess which value the entering basic variable can take and which basic variable should swap places with it and should become non-basic, we should keep in mind the following two characteristics of every LP we deal with:

- every decision variable (original and additional) is **non-negative**;
- in the rightmost column of the tableau, the b vector maps the **values of the basic variables for the current iteration**.

If we consider the two features together, we conclude that the entering basic variable can take the **maximum value possible so that every other basic variable remains non-negative**.

We should remember that any tableau, like the one shown in Table 6.1, is a linear system. Hence the aforementioned requirement can be checked by analyzing each equation (row) where the entering basic variable plays a role. **Note that we do not need to check the objective row here, although we shall see in Section 6.3.4.3 that we will have to update that row, and hence the Z value as well**. Taking again Table 6.1 as a reference, and remembering from Section 6.3.4.1 that our goal is to make x_2 basic, the first equation reads (note that we omit the term depending on Z as it is always multiplied by 0)

$$1 \times x_1 + 0 \times x_2 + 1 \times x_3 + 0 \times x_4 + 0 \times x_5 = 8 \tag{6.60}$$

which, once we get rid of the terms multiplied by 0 and once we recall that $x_1 = 0$ because it is non-basic, becomes $0 \times x_2 + 1 \times x_3 = 8$. Hence, as x_2 is multiplied by 0, this equation is of little use to us at this stage. Let us now move to the second equation that, once we get rid of the null terms, reads

$$1 \times x_2 + 1 \times x_4 = 6 \tag{6.61}$$

This equation is more interesting because it tells us that we cannot increase indefinitely x_2 without making x_4 negative. In fact, we can rearrange (6.61) as $x_4 = \frac{6 - 1 \times x_2}{1}$ which becomes 0 if $x_2 = 6$ and becomes negative if $x_2 > 6$. Hence, from this equation, we can conclude that setting x_4 as the new non-basic variable allows us to raise the value of x_2 from 0 to 6.

We should not forget that there is a third equation we have not analyzed yet, namely

$$4 \times x_2 + 1 \times x_5 = 36 \tag{6.62}$$

Using the same logic used for (6.61), we can write $x_5 = \frac{36 - 4 \times x_2}{1}$, which remains non-negative as long as $x_2 \leq 9$. Because all basic variables should stay non-negative, we cannot make x_5 non-basic by setting it to 0, hence assigning x_2 a value of 9. This would result in $x_4 = \frac{6 - 1 \times 9}{1} = -3$ from (6.61).

On the other hand, if we set $x_2 = 6$ so that x_4 becomes non-basic, we have that $x_5 = \frac{36 - 4 \times 6}{1} = 12$, which is allowed. Hence, we should select x_4 as the exiting basic variable. Such a change implies that we are purchasing 6 trucks of the second type. Recall that x_4 maps the slack in the original $x_2 \leq 6$ constraint. Hence, $x_4 = 0$ entails we are purchasing as many trucks of that type as we can. By doing so, we are using 4×6 ($\times 10,000$) $= 240,000$€ leaving us with $360,000 - 240,000 = 120,000$€ of spare budget ($x_5 = 12$). This explains the value of x_5 in relation to the new solution we obtained.

After the insights gained from this example, let us now formalize how to compute the exiting basic variable. When dealing with both x_4 and x_5, we wrote an expression that can be generalized as

$$x_{ex} = \frac{b_{ex} - c_{ex,ent} \times x_{ent}}{1} \tag{6.63}$$

where x_{ent} is the unknown value of the entering basic variable identified as explained in Section 6.3.4.1, x_{ex} is a candidate exiting basic variable, b_{ex} is its current value (as taken by the rightmost column in the tableau), and $c_{ex,ent}$ is the coefficient of x_{en} in the equation row associated with x_{ex}. **Note that the denominator in Equation 6.63 is set to 1 because it is the coefficient of a basic variable in its row of the tableau. Because of the orthonormal basis requirement, such a coefficient is 1**. In the initial street food company problem of Example 6.1 as displayed in Table 6.1, for x_4 we have $b_{ex} = 6$ and $c_{ex,ent} = 1$, while for x_5 we have $b_{ex} = 36$ and $c_{ex,ent} = 4$.

Because the numerator $b_{ex} - c_{ex,ent} \times x_{ent}$ of every candidate exiting basic variable should remain non-negative, we will set the value of x_{ent} equal to

$$\min_{i \in x_{ex}} \left\{ \frac{b_i}{c_{i,ent}} \right\} \tag{6.64}$$

so that the associated basic variable i is set to 0, becomes non-basic, and is replaced by x_{en} while the remaining basic variables remain basic with non-negative values.

Referring back to the street food company example, we are setting $x_2 = \min\left\{\frac{6}{1}, \frac{36}{4}\right\} = 6$ so that $x_4 = 0$ and $x_5 = 12$ because the reverse case would assign a negative (and hence infeasible) value to x_4. We refer to this process as the **minimum ratio test**.

A final note regards which rows of the entering basic variable column need to be considered to perform the minimum ratio test. Only rows with positive coefficients should be considered. We explain why this is the case in the **Why only rows with positive coefficients are considered for the minimum ratio test** box.

6.3.4.3 Updating the tableau

Finally, to complete an iteration we need to need to update the tableau so that it still complies with the simplex algorithm requirements, namely:

- the column associated with each basic variable should feature a 1 in the row mapping the value of the variable and 0s everywhere else, so that an orthonormal basis with 0/1 values is preserved;
- the coefficient in the objective row of every basic variable should be 0.

We can use a basic property of linear systems to achieve the aforementioned goal, as introduced in Chapter 3, namely that **a linear system does not change if we replace an equation with a linear combination**

Why only rows with positive coefficients are considered for the minimum ratio test

Let us start with the 0 coefficient case. We can use again (6.60) for illustrative purposes (where all the unnecessary zero terms were already removed).

$$0 \times x_2 + 1 \times x_3 = 8$$

Recall that we assessed that x_2 is the entering basic variable. Because it is multiplied by 0, no matter which value we assign to it, the equation will always yield $x_3 = 8$. Hence, there is no option to lower x_3 to 0 and make it non-basic.
A similar reasoning can be applied to negative coefficients. Let us now assume x_2 is still the entering basic variable and that one row of the tableau at a certain iteration is (after all the unnecessary zero terms are removed)

$$-3 \times x_2 + 1 \times x_3 = 8$$

We can express x_3 as $x_3 = \frac{8+3\times x_2}{1}$. Because the numerator is $8 + 3 \times x_2$, we can increase indefinitely x_2 without ever driving x_3 to 0.

Hence, for different reasons, **both rows with 0 or negative coefficients associated with the entering basic variable should not be considered when performing the minimum ratio test**.

of (some of) the original equations of such a system. Let us refer back to the street food company problem of Example 6.1, and to the initial tableau of Table 6.1 where we highlight the rows and columns that will need to change, in Table 6.2. As highlighted in Section 6.3.4.1 and Section 6.3.4.2, we expect x_2 to become basic with a value of 6 and x_4 to become non-basic.

Table 6.2: Initial simplex tableau for the street food company problem of Example 6.1 with the row and column that must undergo changes highlighted in red.

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	-2	-5	0	0	0	0
(x_3)	0	1	0	1	0	0	8
(x_4)	0	0	1	0	1	0	6
(x_5)	0	3	4	0	0	1	36

Not surprisingly, in Table 6.2 the row we need to modify is associated with the exiting variable x_4 (and, after the modifications, will be associated with x_2) while the column we need to modify is associated with the entering variable x_2.

The coefficient where the highlighted row and column overlap should be modified so that it is unitary (note that the fact that in Table 6.2 such a number is already 1 is a coincidence). **This is the case because that row will map the entering basic variable and that column will be one of the columns of the new basis. Hence, we expect a 1 in that row and 0s everywhere else**.

We achieve the first goal by dividing the whole row by $c_{ex,ent}$ (in this case we have that $c_{ex,ent} = 1$). By doing so, we modify the coefficients of the other non-basic variables, the coefficient of the exiting basic variable (that

becomes $1/c_{ex,ent}$), and the value on the right-hand side. Such a value takes the value of $b_{ex}/c_{ex,ent}$, which we assessed to be the new value assigned to the entering variable. We display such a change in Table 6.3, where we also acknowledge that the change entails the row does not map x_4 any longer but x_2.

Z	x_1	x_2	x_3	x_4	x_5	R.H.S.	
0	0	1/1	0	1/1	0	6/1	**(original x_4 row)**
0	0	1	0	1	0	6	**(updated x_2 row)**

Table 6.3: Updated row of the exiting basic variable for the street food company problem of Example 6.1 after the first iteration.

All the remaining values of the highlighted column should be set to 0 instead, to ensure:

- the reduced cost of the entering basic variable in the objective row is set to 0;
- all the other coefficients of that column (aside from $c_{ex,ent} = 1$) are set to 0 so that the tableau is still characterized by an orthonormal matrix mapping the basic variables.

We achieve both goals by using the properties of linear systems again. For example, in the objective row, the coefficient of x_2 is -5 and should become 0 to highlight that x_2 is now a basic variable. We can make this happen by replacing the current objective row with a linear combination of the objective row and the highlighted row in Table 6.3. While other rows (rather than the highlighted one) could be used in the linear combination mathematically speaking, this is the only one to be used to preserve the properties of the tableau. **It is the only row where the coefficients of the remaining basic variables (x_3 and x_5 in this case) are guaranteed to be 0. Hence, their coefficients in the objective row will remain 0 after the replacement as well. This is necessary to ensure that those two variables satisfy the requirements of basic variables at the start of the next iteration. In addition, because the original coefficient of x_4 in the objective row is 0 and the coefficient of such a variable is 1 in the highlighted row (x_4 is the exiting basic variable), its new coefficient in the objective row after the replacement will be different from 0, hence labeling x_4 as a non-basic variable**.

We carry out the row replacement as follows. Let us define R_i a row in the tableau where, in the column associated with x_{ent}, we have a non-zero coefficient $c_{i,ent}$ that should become zero. In addition, let us define R_{ent} as the row that was originally associated with the exiting basic variable and that we have already modified by dividing all values by $c_{ex,ent}$ (hence, now it maps the entering basic variable). Using again the objective row as an example (hence, it is our current R_i), we have that $c_{i,ent} = -5$ and we would like this value to become 0. We can achieve this by replacing R_i with the linear combination $R_i - c_{i,ent} \times R_{ent}$, which is shown in Table 6.4.

Z	x_1	x_2	x_3	x_4	x_5	R.H.S.		
1	-2	-5	0	0	0	0	-	**(original objective row)**
-5×0	-5×0	-5×1	-5×0	-5×1	-5×0	-5×6	=	**(modified highlighted row)**
1	-2	0	0	5	0	30		**(updated objective row)**

Table 6.4: Updated objective row for the street food company problem of Example 6.1 after the first iteration.

The new objective row is then $1 \times Z - 2 \times x_1 + 0 \times x_2 + 0 \times x_3 + 5 \times x_4 + 0 \times x_5 = 30$. This implies that swapping x_2 with x_4 produced an increase in the objective of 30, as now we have $Z = 30$. This is the case because

x_1 is still non-basic and hence 0, x_3 and x_5 are still basic and hence multiplied by a zero coefficient, x_2 is the entering basic variable and hence is multiplied by a zero coefficient as well, and x_4 is the exiting basic variable and has just become non-basic taking a 0 value. Hence $1 \times Z - 2 \times 0 + 0 \times x_2 + 0 \times x_3 + 5 \times 0 + 0 \times x_5 = 30 \rightarrow Z = 30$.

The updated objective value should not surprise us if we think about how it was obtained. The original objective function of the street food company in Example 6.1 is $Z = 2x_1 + 5x_2$. Let us forget, for now, about x_3, x_4, and x_5 as they are needed in the tableau, but play no role in the original objective. We started from a condition where $x_1 = 0$ and $x_2 = 0$, which resulted in $Z = 0$ as highlighted by the right-hand side of the objective row in Table 6.2. We have shown in Section 6.3.4.1 that the entering basic variable is $x_2 = 6$, hence the updated objective is $Z = 2 \times 0 + 5 \times 6 = 30$, as confirmed by the right-hand side of the objective row in Table 6.4. **Note that we obtain such a value because we need to cancel the -5 associated with** x_2 **from the objective row. We cancel that value with the operation** $-5 - (-5 \times 1) = 0$ **(which is part of the linear combination of two rows of the tableau) that updates the objective value as** $0 - (-5 \times 6) = 30$**. The incremental term derives mathematically from the necessity to set to 0 the coefficient of the entering basic variable (**x_2**) in the tableau**. The equivalent practical explanation is that the switch from a solution where we purchase no trucks at all ($x_1 = 0$, $x_2 = 0 \rightarrow Z = 0$) to a solution where we purchase 6 trucks of the second type ($x_1 = 0$, $x_2 = 6 \rightarrow Z = 30$).

We now need to modify the remaining two rows (associated with x_3 and x_5) of the tableau to replace their current coefficients with 0s. For the row associated with x_3 we have that $c_{i,ent} = 0$. R_i should be replaced by $R_i - 0 \times R_{ent} = R_i$, which means no change at all, as displayed for the sake of completeness in Table 6.5.

Table 6.5: Updated (x_3) row for the street food company problem of Example 6.1 after the first iteration.

Z	x_1	x_2	x_3	x_4	x_5	R.H.S.		
0	1	0	1	0	0	8	-	**(original (x_3) row)**
-0×0	-0×0	-0×1	-0×0	-0×1	-0×0	-0×6	=	**(modified highlighted row)**
0	1	0	1	0	0	8		**(updated (x_3) row)**

The fact that this row did not need any change was expected. We required the current $c_{i,ent}$ coefficient to become 0, and since such a coefficient is already 0 for the row under scrutiny, no replacement with any linear combination is needed. Note that this lack of change has also another interpretation that links more closely to the mathematical problem at hand. During the current iteration, we are modifying the values of x_2 (from 0 to 6) and x_4 (from 6 to 0). Changing these 2 values affects:

- the objective row, which depends on x_2;
- constraint $x_2 + x_4 = 6$, which depends on both;
- constraint $3x_1 + 4x_2 + x_5 = 36$, which depends on x_2.

On the other hand, constraint $x_1 + x_3 = 8$ remains unaffected because x_1 is still non-basic, which means $x_3 = 8 - x_1 = 8 - 0 = 8$. **Because neither the entering nor the exiting basic variables appear in such a constraint, the associated row in the tableau needs no modification at all so that** x_3 **retains its current value of 8**.

Finally, for the row associated with x_5 we have that $c_{i,ent} = 4$. R_i should be replaced by $R_i - 4 \times R_{ent} = R_i$ as shown in Table 6.6.

Z	x_1	x_2	x_3	x_4	x_5	R.H.S.		
0	3	4	0	0	1	36	-	**(original (x_5) row)**
4×0	4×0	4×1	4×0	4×1	4×0	4×6	=	**(modified highlighted row)**
0	3	0	0	-4	1	12		**(updated (x_5) row)**

Table 6.6: Updated (x_5) row for the street food company problem of Example 6.1 after the first iteration.

According to Table 6.6, the new x_5 value drops from 36 to 12. Due to the replacement of the original row with the linear combination so that the coefficient $c_{i,ent} = 4$ is turned into a 0, the right-hand side undergoes a reduction equal to $4 \times 6 = 24$. Similar to the objective row, there is an explanation that relates directly to the problem at hand behind this reduction. Recall that, in (6.26), x_5 maps the budget we are not using to purchase trucks. If $x_1 = 0$ and $x_2 = 0$, then $x_5 = 36$ because all the budget is still available. We are now switching to a solution where $x_1 = 0$ still holds, but where $x_2 = 6$. Hence, $x_5 = 36 - 3 \times 0 - 4 \times 6 = 12$ maps we are using 240,000€ out of the 360,000€ available, saving 120,000€ in the process.

Table 6.7 displays the complete revised tableau after the first iteration is completed.

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	-2	0	0	5	0	30
(x_3)	0	1	0	1	0	0	8
(x_2)	0	0	1	0	1	0	6
(x_5)	0	3	0	0	-4	1	12

Table 6.7: Simplex tableau for the street food company problem of Example 6.1 after the first iteration.

Note that the 3 basic variables do not follow their increasing index order. As the leftmost column suggests, the first row maps x_3, the second one x_2 (which replaced the original x_4), and the third x_5. We also want to remind readers that such a column is just added for readability, but it is not needed because the right sequence can be retrieved by analyzing where the 1s appear in the orthonormal basis. The basis is highlighted again in red. Note that now the columns of the basis are neither continuous nor define an identity matrix, but this poses no issues to the simplex algorithm. For x_2, the 1 appears in the second row, hence such a row maps x_2 and since the right-hand side is 6, we can write $x_2 = 6$. For x_3, the 1 appears in the first row, hence $x_3 = 8$. For x_5, the 1 appears in the third row, hence $x_5 = 12$.

Table 6.7 carries all the information needed to interpret the new solution, that is $(x_1, x_2, x_3, x_4, x_5) = (0, 6, 8, 0, 12)$ with an objective $Z = 30$ in the rightmost column. In addition, we can confirm it satisfies the requirements of a tableau at any iteration: the coefficients of the basic variables are 0 in the objective row, and (as mentioned above) the basic variables are mapped with an orthonormal basis in the tableau.

Finally, we can confirm this iteration is not the last one because one of the non-basic variables (x_1) has a negative coefficient in the objective row. As explained in Section 6.3.4.1, such a variable should be the next one to become basic.

6.3.4.4 The Algorithm in a nutshell

In Section 6.3.4.1, Section 6.3.4.2, and Section 6.3.4.3 we discussed separately the steps defining a single iteration to update a simplex tableau. Given an LP, we can solve it with the simplex method following the sequence of steps described in the **Algorithm of the simplex method** box. In **(red)** we highlight, for certain mathematical operations in the algorithm, the associated geometric interpretation as defined in Section 6.3.2. We also remind readers that we assume the problem at hand is a max problem. Hence, our goal is to increase as much as possible the objective value.

Algorithm of the simplex method

- Convert the LP into augmented form;
- Choose an initial solution that satisfies the tableau requirements **(identify a starting corner point and the associated basic solution)** → the **coefficients of the basic variables are 0 in the objective row** and the **associated columns in the rest of the tableau form an orthonormal basis**;
- WHILE there is **non-basic variables** with a **negative reduced cost** in the objective row **(the current corner point is not optimal)**:
  - set the variable with the most negative reduce cost as the **entering basic variable** x_{ent} (Section 6.3.4.1);
  - identify the **exiting basic variable** x_{ex} via the **minimum ratio test (identify which adjacent corner point we should move to)** (Section 6.3.4.2);
  - identify the **coefficient** $c_{ex,ent}$ at the intersection of the row of the exiting basic variable and the column of the entering basic;
  - divide the row associated with x_{ex} by $c_{ex,ent}$. **This row, labeled** R_{ent}**, now maps** x_{en}. This step also sets the value of x_{ent} equal to $\min_{i \in x_{ex}} \left\{ \frac{b_i}{c_{i,ent}} \right\}$;
  - **replace the objective row** with a linear combination of that row and R_{ent} so that the coefficient of x_{ent} is set to 0 (Section 6.3.4.3);
  - **replace each row below the objective row that is not** R_{ent} with a linear combination of that row and R_{ent} so that the coefficient of x_{ent} is set to 0 (Section 6.3.4.3).
- the **rightmost column** of final tableau stores the **optimal objective in the objective row**, and the **optimal values of the basic variables in the rows below**. The **sequence** of the optimal basic variables is given by the **location of the 1s in the final orthonormal basis** (Section 6.3.4.3) **(we are now positioned in the optimal corner point)**.

6.4 Examples

We now showcase three examples of the simplex method, from the original LP formulation to its augmented form, to the initial tableau, to the final optimal solution.

Example 6.4 *The first example is the original version of the street food company problem*. The original LP is:

$$\max \quad Z = 2x_1 + 5x_2 \tag{6.65}$$

s.t.:

$$x_1 \leq 8 \tag{6.66}$$
$$x_2 \leq 6 \tag{6.67}$$
$$3x_1 + 4x_2 \leq 36 \tag{6.68}$$
$$x_1, x_2 \geq 0 \tag{6.69}$$

while its augmented form is

$$\max \quad Z = 2x_1 + 5x_2 \tag{6.70}$$

s.t.:

$$x_1 + x_3 = 8 \tag{6.71}$$
$$x_2 + x_4 = 6 \tag{6.72}$$
$$3x_1 + 4x_2 + x_5 = 36 \tag{6.73}$$
$$x_1, x_2, x_3, x_4, x_5 \geq 0 \tag{6.74}$$

We have already shown in Table 6.1 that, if we start from feasible corner point $(x_1, x_2) = (0, 0)$ and the associated basic feasible solution $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 8, 6, 36)$, the initial tableau is the one shown in Table 6.8.

Table 6.8: Initial simplex tableau for Example 6.4.

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	-2	-5	0	0	0	0
(x_3)	0	1	0	1	0	0	8
(x_4)	0	0	1	0	1	0	6
(x_5)	0	3	4	0	0	1	36

In Table 6.8 we highlight in red the column associated with the entering basic variable x_2 and the row associated with the leaving basic variable x_4. Hence, after elementary row operations, we will move to a different corner point and basic solution where x_2 is non-zero, x_4 is 0, x_1 remains

non-basic, and x_3 and x_5 remain basic. Note that, for this example and specific iteration, we provided all the details in Section 6.3.4.3. After the first iteration, the new tableau is

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	-2	0	0	5	0	30
(x_3)	0	1	0	1	0	0	8
(x_2)	0	0	1	0	1	0	6
(x_5)	0	3	0	0	-4	1	12

Table 6.9: Simplex tableau for Example 6.4 after the first iteration.

where we are already highlighting the column of the new entering basic variable x_1 and exiting basic variable x_5. From Table 6.9, we infer already that $x_1 = \frac{12}{3} = 4$. We will also have to translate the -2 in the objective row and the 1 in the (x_3) row into 0s. After these row operations, the tableau after the second iteration is

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	0	0	0	13/3	2/3	38
(x_3)	0	0	0	1	4/3	-1/3	4
(x_2)	0	0	1	0	1	0	6
(x_1)	0	1	0	0	-4/3	1/3	4

Table 6.10: Simplex tableau for Example 6.4 after the second iteration.

where no more rows/columns are highlighted in red as the solution is optimal: all the coefficients in the objective row of current non-basic variables are now positive. Hence, the optimal corner point is $(x_1, x_2) = (4, 6)$ and the associated optimal basic solution is $(x_1, x_2, x_3, x_4, x_5) = (4, 6, 4, 0, 0)$. The optimal value of the objective is $Z = 38$. Notice that the value coincides with the one found graphically in Section 6.1.

Example 6.5 *We now tackle a first variation of the street food company problem of Example 6.4. In this variation, the third constraint is in equality form, hence forcing us to use all the budget available*. The original LP is:

$$\max \; Z = 2x_1 + 5x_2 \tag{6.75}$$

s.t.:

$$x_1 \leq 8 \tag{6.76}$$
$$x_2 \leq 6 \tag{6.77}$$
$$3x_1 + 4x_2 = 36 \tag{6.78}$$
$$x_1, x_2 \geq 0 \tag{6.79}$$

while its augmented form is:

$$\max \; Z = 2x_1 + 5x_2 - Mx_5 \tag{6.80}$$

s.t.:

$$x_1 + x_3 = 8 \tag{6.81}$$
$$x_2 + x_4 = 6 \tag{6.82}$$
$$3x_1 + 4x_2 + x_5 = 36 \tag{6.83}$$
$$x_1, x_2, x_3, x_4, x_5 \geq 0 \tag{6.84}$$

If we were to follow the procedure from Example 6.4, we would start from corner point $(x_1, x_2) = (0, 0)$ and the basic solution $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 8, 6, 36)$. If we were to fill in all the necessary numbers in the initial tableau, we would get the tableau depicted in Table 6.11.

Table 6.11: Initial simplex tableau for Example 6.5 before correct initialization.

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	-2	-5	0	0	M	0
(x_3)	0	1	0	1	0	0	8
(x_4)	0	0	1	0	1	0	6
(x_5)	0	3	4	0	0	1	36

A first wake-up call is that, as mentioned in Section 6.3.4.4, we have a basic variable (x_5) whose coefficient in the objective row is different than 0. Related to this point, we should always recall that we can easily check, at any iteration, if the objective value is correct by plugging the values of the basic variables into the objective function. Table 6.11 is associated with the basic solution $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 8, 6, 36)$ which, when plugged in Equation 6.80, should yield $Z = 2 \times 0 + 5 - M \times 36 = -36M$. Our current tableau states that the current objective value is 0, which is indeed wrong. **As anticipated, the key is to transform the M coefficient in the objective row into a 0**. This is achieved by replacing the objective row with a linear combination of the row itself minus M times the current (x_5) row. After this row operation, we finally get the correct initial tableau displayed in Table 6.12.

Table 6.12: Initial simplex tableau for Example 6.5 after correct initialization.

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	-2-3M	-5-4M	0	0	0	-36M
(x_3)	0	1	0	1	0	0	8
(x_4)	0	0	1	0	1	0	6
(x_5)	0	3	4	0	0	1	36

Table 6.12 satisfies now all the requirements of a simplex tableau. In particular, the coefficient of the starting basic variables x_3, x_4, and x_5 are all 0. We can also notice, not surprisingly, that we are now starting our algorithm from an infeasible corner point. As visible from Figure 6.9, **corner point $(x_1, x_2) = (0, 0)$ does not lie in the feasible region. This is generally not a problem, because we acknowledge the infeasibility of the original problem by highly penalizing the objective value (having an objective equal to $-36M$ implies infeasibility). As we are looking for adjacent corner points that improve our objective, we should be able to reach the feasible region in some iterations**.

In Table 6.12 we have highlighted, as usual, the column of the entering basic variable and the row of the exiting basic variable. Note that, albeit with M the concept of larger or smaller is a bit vaguer, we selected x_2 as the entering variable as -5-4M ≤ -2-3M. After the usual row operations, the revised tableau is shown in Table 6.13.

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	-2-3M	0	0	5+4M	0	30-12M
(x_3)	0	1	0	1	0	0	8
(x_2)	0	0	1	0	1	0	6
(x_5)	0	3	0	0	-4	1	12

Table 6.13: Simplex tableau for Example 6.5 after the first iteration.

In Table 6.13 we highlight that x_1 is the new entering basic variable and x_5 is the new exiting basic variable. In addition, the objective value is $Z = 30 - 12M$, which testifies that our new solution is still infeasible. This can easily be verified by checking that $x_5 = 12$ is still a basic variable, meaning we are not satisfying the original $3x_1 + 4x_2 = 36$ constraint yet.

After performing a new round of row operations, the revised tableau after the second iteration is depicted in Table 6.14.

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	0	0	0	7/3	2+3M	38
(x_3)	0	0	0	1	4/3	-1/3	4
(x_2)	0	0	1	0	1	0	6
(x_1)	0	1	0	0	-4/3	1/3	4

Table 6.14: Simplex tableau for Example 6.5 after the second iteration.

Analyzing the objective row of Table 6.14 we conclude the current solution is optimal. As already shown graphically in Section 6.1, the optimal solution to this example is the same as the one from Example 6.4, although the process to compute it is quite different.

For the sake of clarity, we show in Figure 6.10 the three functional constraints with blue lines, the resulting feasible region (which is a segment) in green, and the feasible and infeasible corner points in gray and red, respectively. In addition, with dark red arrows we highlight the sequence of corner points visited by the simplex method: $(x_1, x_2) = (0, 0) \rightarrow (x_1, x_2) = (0, 6) \rightarrow (x_1, x_2) = (4, 6)$. **Because in every iteration, when swapping a basic with a non-basic variable, we move from a corner point to a neighboring one, we could already expect we needed at least two iterations to converge to the optimal corner point $(x_1, x_2) = (4, 6)$. In fact, such a corner point is not a direct neighbor of the starting corner point $(x_1, x_2) = (0, 0)$.** For the same reason, if during iteration 1 we had decided to make x_1 instead of x_2 the entering basic variable (hence moving from $(x_1, x_2) = (0, 0)$ to $(x_1, x_2) = (8, 0)$), we would have needed 3 iterations to converge to the optimal corner point (we encourage readers to try that).

Example 6.6 *In this street food company scenario, our CEO allows unrestricted truck purchases, but we must meet a minimum spending requirement of 120,000€ as the fiscal year ends. However, production limits constrain availability for both truck types as we experienced with Example 6.4 and Example 6.5.* The LP that maps our problem is:

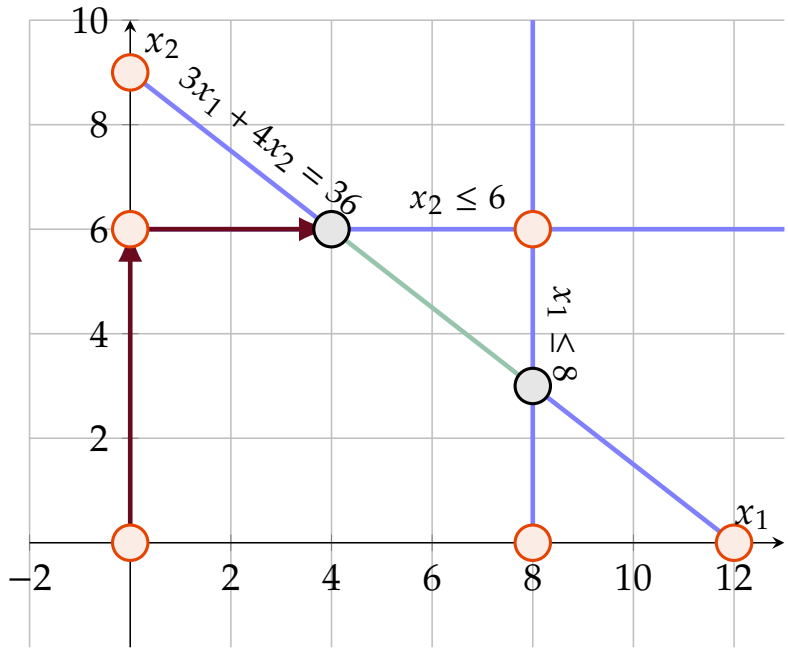


Figure 6.10: Sequence of corner points visited in Example 6.5.

$$\max \quad Z = 2x_1 + 5x_2 \tag{6.85}$$

s.t.:

$$x_1 \leq 8 \tag{6.86}$$
$$x_2 \leq 6 \tag{6.87}$$
$$3x_1 + 4x_2 \geq 12 \tag{6.88}$$
$$x_1, x_2 \geq 0 \tag{6.89}$$

while its augmented form is

$$\max \quad Z = 2x_1 + 5x_2 - Mx_6 \tag{6.90}$$

s.t.:

$$x_1 + x_3 = 8 \tag{6.91}$$
$$x_2 + x_4 = 6 \tag{6.92}$$
$$3x_1 + 4x_2 - x_5 + x_6 = 12 \tag{6.93}$$
$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \tag{6.94}$$

A thoughtful reader might question the necessity of employing the complete simplex process for this problem. **Since there is no upper spending limit and truck purchases are only restricted by production limits, the optimal corner point is** $(x_1, x_2) = (8, 6)$ **and the optimal basic**

solution is $(x_1, x_2, x_3, x_4, x_5, x_6) = (8, 6, 0, 0, 36, 0)$. Notwithstanding, we show the full algorithm applied to this case as well so surplus variables are also treated in one of our examples.

With a similar approach to what showed in Example 6.5, we need to already manipulate the original tableau because we want x_6 to be basic so that it initially takes a value equivalent to the right-hand side of (6.93). Hence, its value in the objective row should be 0 and not M. In Table 6.15 we show the initial tableau after the row operation to ensure this has already been performed. For any doubt, we refer readers to the conversion of Table 6.11 into Table 6.12.

	Z	x_1	x_2	x_3	x_4	x_5	x_6	R.H.S.
	1	-2-3M	-5-4M	0	0	M	0	-12M
(x_3)	0	1	0	1	0	0	0	8
(x_4)	0	0	1	0	1	0	0	6
(x_6)	0	3	4	0	0	-1	1	12

Table 6.15: Initial simplex tableau for Example 6.6 after correct initialization.

As usual, in Table 6.15 we highlight the column of the entering basic variable (x_2) and the row of the exiting basic variable (x_6). In addition, we can always verify, given that our current corner point $(x_1, x_2) = (0, 0)$ and associated basic solution $(x_1, x_2, x_3, x_4, x_5, x_6) = (0, 0, 8, 6, 0, 12)$ are both infeasible, the objective value is correctly set at $-12M$ to map such an infeasible starting point. After performing the necessary row operations to ensure the swap between x_2 and x_6, we get the following new tableau (Table 6.16).

	Z	x_1	x_2	x_3	x_4	x_5	x_6	R.H.S.
	1	7/4	0	0	0	-5/4	M+5/4	15
(x_3)	0	1	0	1	0	0	0	8
(x_4)	0	-3/4	0	0	1	1/4	-1/4	3
(x_2)	0	3/4	1	0	0	-1/4	1/4	3

Table 6.16: Simplex tableau for Example 6.6 after the first iteration.

Because we made x_6 (our artificial variable) non-basic, we moved to the feasible corner point $(x_1, x_2) = (0, 3)$ associated with the basic feasible solution $(x_1, x_2, x_3, x_4, x_5, x_6) = (0, 3, 8, 3, 0, 0)$ that yields $Z = 15$. Because the coefficient in the objective row of x_5 is still negative, we need at least another iteration where, because of the minimum ratio test, x_4 will become non-basic. The new iteration is shown in Table 6.17.

	Z	x_1	x_2	x_3	x_4	x_5	x_6	R.H.S.
	1	-2	0	0	5	0	M	30
(x_3)	0	1	0	1	0	0	0	8
(x_5)	0	-3	0	0	4	1	-1	12
(x_2)	0	0	1	0	1	0	0	6

Table 6.17: Simplex tableau for Example 6.6 after the second iteration.

Because of the value of x_2 doubled ($3 \rightarrow 6$) and both x_1 and x_6 are still non-basic, the objective value doubled as well: $Z = 30$. Because the coefficient of x_1 in the objective row is negative, we need row operations to make such a decision variable basic while making x_3 non-basic. After this additional iteration, we obtain the final tableau shown in Table 6.18. Because all the coefficients of the basic variables in the objective row are now positive, optimality is proven.

Table 6.18: Simplex tableau for Example 6.6 after the third iteration.

	Z	x_1	x_2	x_3	x_4	x_5	x_6	R.H.S.
	1	0	0	2	5	0	M	46
(x_1)	0	1	0	1	0	0	0	8
(x_5)	0	0	0	3	4	1	-1	36
(x_2)	0	0	1	0	1	0	0	6

The optimal corner point is $(x_1, x_2) = (8, 6)$ with the optimal basic solution $(x_1, x_2, x_3, x_4, x_5, x_6) = (8, 6, 0, 0, 36, 0)$ and $Z = 48$. Note that $x_5 = 36$ (our surplus variable) makes perfect sense. We were required to spend at least 120,000€ and we ended up spending $30,000 \times 8 + 40,000 \times 6 = 480,000$€, hence we are exceeding the minimum required value by 360,000€.

Furthermore, by plotting the feasible region and corner points, it was evident that a minimum of three iterations would be required to reach the optimal solution. We highlight this and, in particular, the sequence of corner points visited, in Figure 6.11

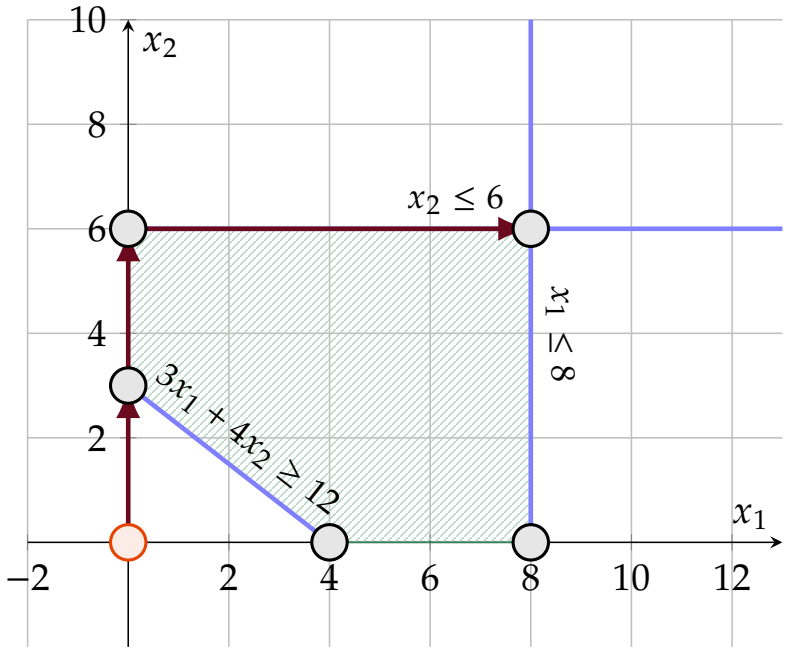


Figure 6.11: Sequence of corner points visited for Example 6.6.

So far, we have primarily discussed the simplex method in the context of max problems. However, in various instances, including practical applications, our objective is to minimize the objective function. In a minimization problem, we apply the same logic with row operations but focus on the non-basic variable with the most positive (rather than the most negative) coefficient in the objective row. **Alternatively, minimizing an objective is equivalent to maximizing the same objective with a minus sign**. In essence, we translate

$$\min \; Z = \sum_{i=1}^{n} C_i x_i \tag{6.95}$$

into

$$\max \quad -Z = -\sum_{i=1}^{n} C_i x_i \tag{6.96}$$

If we follow this approach, we can use the very same technique and logic explained previously in this chapter. We showcase this with an example where the original LP is a minimization problem.

Example 6.7 *We consider the minimization problem represented by (6.97)-(6.101). We want to translate it into a maximization problem and solve it with the simplex method.*

The original LP is:

$$\min \quad Z = x_1 + x_2 \tag{6.97}$$

s.t.:

$$2x_1 + x_2 \leq 8 \tag{6.98}$$
$$x_1 + 2x_2 \geq 4 \tag{6.99}$$
$$2x_1 + x_2 \geq 4 \tag{6.100}$$
$$x_1, x_2 \geq 0 \tag{6.101}$$

while its augmented form is:

$$\min \quad Z = x_1 + x_2 + Mx_5 + Mx_7 \tag{6.102}$$

s.t.:

$$2x_1 + x_2 + x_3 = 8 \tag{6.103}$$
$$x_1 + 2x_2 - x_4 + x_5 = 4 \tag{6.104}$$
$$2x_1 + x_2 - x_6 + x_7 = 4 \tag{6.105}$$
$$x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0 \tag{6.106}$$

Note that in Equation 6.102 we are penalizing the two artificial variables x_5 and x_7 with a plus sign because this is a minimization problem. Therefore, when either variable exceeds 0 (indicating a violated constraint), we substantially raise the objective as a representation of this infeasibility.

In Figure 6.12 we graphically represent the original problem at hand, with the feasible region and the corner points (both feasible and infeasible). Because this is the only minimization example we showcase, we also added the objective function line $x_2 = -x_1 + Z$ passing through corner point $(x_1, x_2) = (4/3, 4/3)$, where $Z = 8/3$ which represents the optimal solution. Because we deal with a minimization problem, our

goal (graphically) is to shift the objective function line $x_2 = -x_1 + Z$ as close as possible to the origin while remaining inside the feasible region.

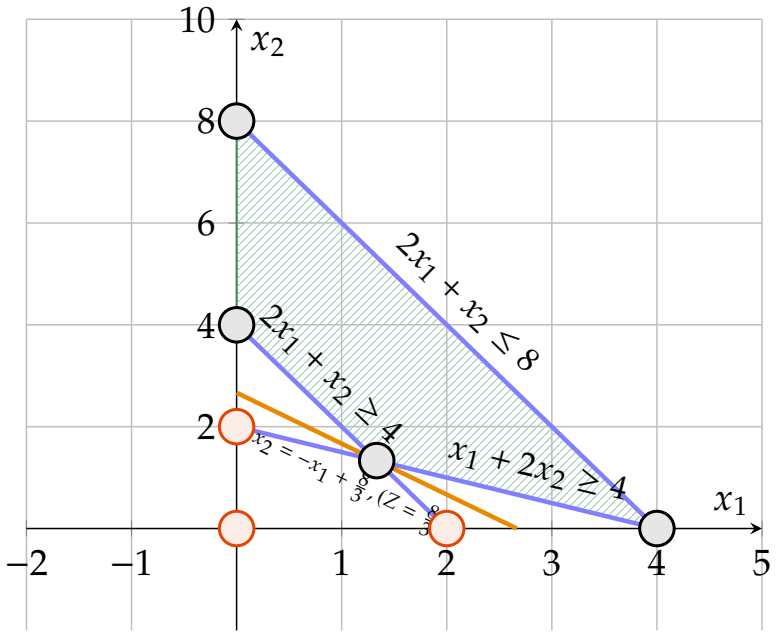


Figure 6.12: Corner points and feasible region for the Example 6.7.

We now proceed to use the simplex method and obtain the same optimal solution. We replace (6.102) with

$$\max \quad -Z = -x_1 - x_2 - Mx_5 - Mx_7 \tag{6.107}$$

so that the problem we need to solve features (6.107) as the objective and constraints (6.103)-(6.106). We want to start setting our original x_1 and x_2 variables and the surplus variables x_4 and x_6 to 0. By doing so, we let the slack variable x_3 and the artificial variables x_5 and x_7 take the value of the right-hand side of the constraint where they appear. In both Example 6.5 and Example 6.6 we showed that we need to modify the initial tableau to ensure every coefficient of a basic variable (x_3, x_5, and x_7 in this case) is 0 in the objective row. The correct initial tableau for our example is shown in Table 6.19. **Something important to notice is the coefficient of Z in the objective row, which is -1 instead of 1. This is not a typo but stems from the initial conversion** $\min Z = \max -Z$. We can also confirm this by "manually" checking the objective value of the starting infeasible corner point $(x_1, x_2) = (0, 0)$ and the associated basic infeasible solution $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (0, 0, 8, 0, 4, 0, 4)$. From (6.102) we obtain $Z = 1 \times 0 + 1 \times 0 + M \times 4 + M \times 4 = 8M$. Expanding the objective row from Table 6.19, we write $-1 \times Z + (1 - 3M) \times 0 + (1 - 3M) \times 0 + 0 \times 8 + M \times 0 + 0 \times 4 + M \times 0 + 0 \times 4 = -8M$ which simplifies into $-Z = -8M \rightarrow Z = 8M$.

An interesting aspect of this example, as highlighted in Table 6.19, is that we have two non-basic variables with the same negative coefficient in the objective row, namely x_1 and x_2. Formally, each of them could be chosen as our rule creates a tie here. We randomly pick x_2, as highlighted by the red column. Because of the minimum ratio test, x_5 is the leaving basic variable.

	Z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	R.H.S.
	-1	1-3M	1-3M	0	M	0	M	0	-8M
(x_3)	0	2	1	1	0	0	0	0	8
(x_5)	0	1	2	0	-1	1	0	0	4
(x_7)	0	2	1	0	0	0	-1	1	4

Table 6.19: Initial simplex tableau for Example 6.7 after correct initialization.

After the usual row operations, we obtain the tableau shown in Table 6.20. We can verify that the objective, while still featuring an M term which implies infeasibility, has decreased from $8M$ to $2 + 2M$. This is consistent with the current infeasible corner point $(x_1, x_2) = (0, 2)$ and associated basic infeasible solution $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (0, 2, 6, 0, 0, 0, 2) \rightarrow Z = x_1 + x_2 + Mx_5 + Mx_7 = 1 \times 0 + 1 \times 2 + M \times 0 + M \times 2 = 2 + 2M$.

With two non-basic variables having negative coefficients in the objective row, x_1 is selected as the entering basic variable, given its more negative coefficient. Consequently, x_7 follows the opposite path.

	Z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	R.H.S.
	-1	1/2-3/2M	0	0	1/2-1/2M	-1/2+1/2M	M	0	-2-2M
(x_3)	0	3/2	0	1	1/2	-1/2	0	0	6
(x_2)	0	1/2	1	0	-1/2	1/2	0	0	2
(x_7)	0	3/2	0	0	1/2	-1/2	-1	1	2

Table 6.20: Simplex tableau for Example 6.7 after the first iteration.

After another round of row operations, we obtain the tableau shown in Table 6.21. Because all non-basic variables have a positive coefficient in the objective row, we conclude our current corner point $(x_1, x_2) = (\frac{4}{3}, \frac{4}{3})$ is the optimal one and the associated basic solution $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (\frac{4}{3}, \frac{4}{3}, 4, 0, 0, 0, 0)$ yields the optimal value $Z = \frac{8}{3}$.

	Z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	R.H.S.
	-1	0	0	0	1/3	-1/3+M	1/3	-1/3+M	-8/3
(x_3)	0	0	0	1	0	0	1	-1	4
(x_2)	0	0	1	0	-2/3	2/3	1/3	-1/3	4/3
(x_1)	0	1	0	0	1/3	-1/3	-2/3	2/3	4/3

Table 6.21: Simplex tableau for Example 6.7 after the second iteration.

This example is quite interesting if we consider again the decision we took in the first iteration about the entering basic variable. x_1 and x_2 were characterized by the same negative reduced cost and we randomly decided to make x_2 the entering basic variable. It can be verified that, if we had chosen x_1 as the entering basic variable, we would have obtained the same optimal solution with the same number of steps. This is due to the symmetry of corner points $(x_1, x_2) = (0, 0)$, $(x_1, x_2) = (2, 0)$, $(x_1, x_2) = (0, 2)$, and $(x_1, x_2) = (4/3, 4/3)$. We highlight the sequence of corner points that was followed in the example, and the alternative sequence that would have been followed if x_1 had been chosen in Figure 6.13.

Another point of interest of Example 6.7 relates to the final optimal solution. The simplex method identified corner point $(x_1, x_2) = (4/3, 4/3)$ as the optimal one. We did not describe on purpose what practical problem the LP at hand was addressing. Let us assume it was about workforce allocation with x_1 and x_2 representing, respectively, the number of workers specialized in two different skills that we need to complete a production task. Our objective function then aims at minimizing their collective number (as a proxy of minimizing salary cost). **We identify**

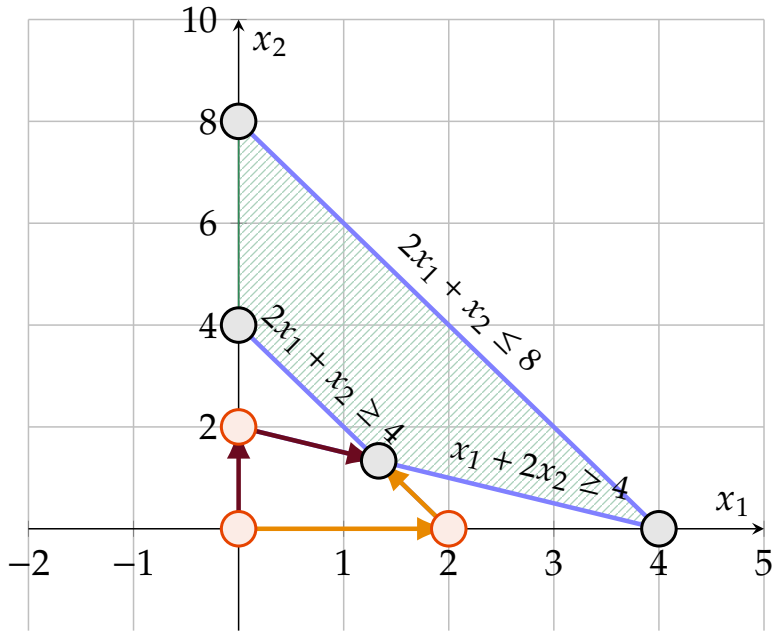


Figure 6.13: Sequence of corner points visited in Example 6.7 (in dark red) and alternative sequence of corner points if x_1 had been chosen in the first iteration (in orange).

an issue with the current optimal solution. While it makes perfect sense mathematically, it has no practical value as we cannot hire $\frac{4}{3}$ specialized workers. This simple example highlights an issue with the simplex method that relates to the specific nature of the decision variables characterizing a mathematical model. We will extensively cover this issue and how to cope with it in Chapter 7.

6.5 Additional considerations

We covered quite extensively the basic pillars of the simplex method. We started with the graphical representation of an LP, then moved to its augmented form that is necessary for the simplex algorithm to work. We discussed the need for basic and non-basic variables to have a square system that can be updated with row operations, and we discussed how each iteration is performed and the stopping criterion use to stop the simplex method. Notwithstanding, and to the potential surprise of the reader, we only scratched the surface of such an important topic in linear optimization. Some main aspects that we did not cover are:

- how the simplex method handles **tie-breakers** (such as in Example 6.7) or non-standard situations when selecting entering/exiting basic variables;
- **sensitivity analysis**, i.e., how to quantitatively assess if and to what extent changes in coefficients of the objective function or of the constraints affect the optimal solution. Recalling Example 6.4, the optimal solution is $(x_1, x_2) = (4, 6)$, indicating prioritizing trucks of the second type. The current cost and customer attraction factors prompt questions for the CEO: how much cheaper should the first type of trucks be, and how many more customers must each attract to justify more purchases? Sensitivity analysis offers tools for precise quantitative answers to such queries;

- **duality**, i.e., a property of every original LP (which we label "primal" in this context) that associates with it a "dual" problem. The primal problem involves maximizing or minimizing an objective function subject to constraints, while the dual problem involves minimizing or maximizing a different objective function under constraints derived from the primal problem. **The key feature of duality is that certain properties and relationships between the primal and dual problems hold. Specifically, the optimal value of one problem provides a bound on the optimal value of the other**. Additionally, the dual problem can provide insights into the sensitivity of the optimal solution to changes in the problem parameters as described in the previous bullet point.

We do not address the listed features in this book, but refer readers to Hillier and Lieberman (2015), for example, in case of need.

Branch & Bound (BB) 7

But magic is like pizza: even when it's bad, it's pretty good.

Neil Patrick Harris

7.1 Motivation for BB 103
7.2 Problem types 104
7.3 The basics of BB 104
7.4 Linear relaxation, root node, and tree structure 105
7.5 Best bound, best incumbent, and gap optimality 107
7.6 A note on functional constraints 110
7.7 Fathoming rules 111
7.8 Branching, bounding, and separation rules . . 111
7.8.1 Branching options . . . 112
7.8.2 Bounding and separation rules 114
7.9 The BB algorithm in a nutshell 118
7.10 Considerations on the algorithmic complexity of the BB algorithm . . 120
7.11 An illustrative example 120

7.1 Motivation for BB

In Example 6.7 we showed an example of LP where the optimal solution, while mathematically feasible, might not be physically meaningful. In essence, every decision variable in a model we design has a specific meaning and, hence, type. It is the modeler's role to ensure the solution to an optimization model is meaningful and implementable, on top of satisfying some mathematical optimality criteria.

Going back to the original street food company problem of Example 6.4 that has accompanied us for most of Chapter 6, we always displayed the *mathematical* feasible region without questioning too much if all possible (x_1, x_2) pairs inside such a region would also be practically feasible. The answer is no, because x_1 and x_2 define, respectively, the number of purchased trucks of the first and second type. **Hence, we should enforce that they only take integer values to have a solution that satisfies the practical requirements of the original problem at hand**.

Because of this reason, for Example 6.4 the original "continuous" feasible region translates into the discrete set of integer (x_1, x_2) points contained in the original feasible region, as shown in Figure 7.1.

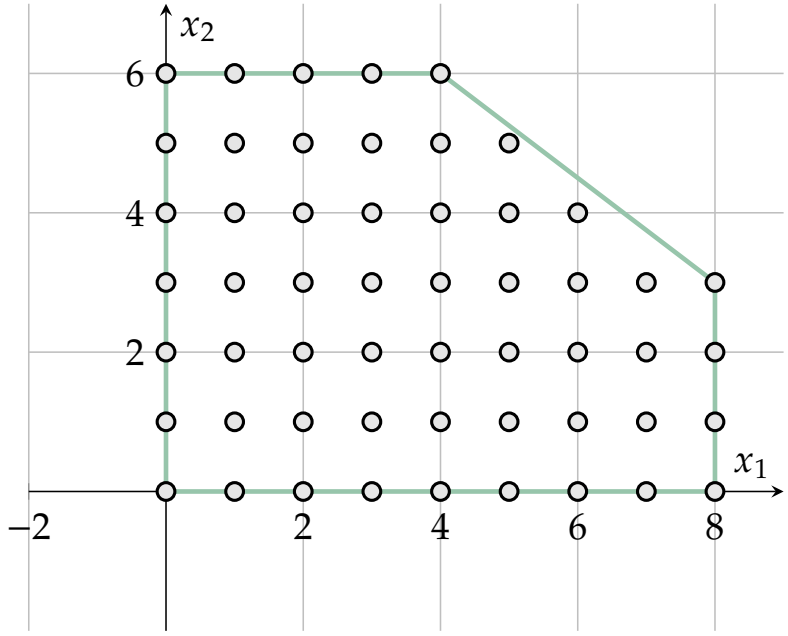


Figure 7.1: Integer feasible solutions for Example 6.4.

As all the corner points in Figure 7.1 are integers ((0, 0), (8, 0), (8, 3), (4, 6), and (0, 6)), and we know the optimal solution in an LP lies on one of these

points, applying the simplex method to Example 6.4 is straightforward and yields an integer solution.

If feasible corner points exhibit mismatches between decision variable values and their nature, as seen in Example 6.7, the simplex method may fail in finding an optimal solution where each decision variable is of the correct type. In such cases, we turn to an approach that retains the simplex method's efficiency in scanning through corner points of an LP while ensuring recognition of a fully mathematically feasible solution. This is a capability not inherent in the simplex method alone. We call such a method Branch & Bound (BB) because it embeds the simplex method into a decision tree structure (hence, the **branch** part of the name) and it uses ad-hoc mathematical properties to limit the exploration of such a tree only to relevant parts (hence, the **bound** part of the name)

7.2 Problem types

As per the initial findings in Section 7.3, if the problem we are dealing with is an LP, then using the simplex method suffices because every decision variable is allowed to take fractional (continuous) values. As soon as a single decision variable in our model is bound to be integer or binary, then BB is (formally)[1] needed. The models that can be handled by BB are:

- Binary Program (BP) models, i.e., models where **every decision variable is binary**;
- Integer Program (IP) models, i.e., models where **every decision variable is integer**;
- Mixed Integer Linear Program (MILP) models, i.e., models where **decision variables are of mixed type (binary, integer, and continuous)**.

1: We already elaborated that the simplex method will suffice if all corner points defining the feasible region of the problem are characterized by integer values. This is what happened in Example 6.4 but, in general, for large problems we have to assume not all the corner points are integer and hence must rely on BB.

7.3 The basics of BB

We have seen in Chapter 6 that every constraint we add to a model brings additional decision variables that are needed for the augmented form. This is also the case if we want to enforce a decision variable to take an integer value. Hence, a preliminary insight tells us that if we take an LP and transform it into an IP by forcing all the original continuous variables to be integer, **while the size of the original problem does not change, the size of the augmented problem will**. In addition, while for continuous variables we can let the simplex method assess their optimal value through the iterations, how do we know which integer value is the optimal one for each decision variable? For small problems, we might get away with full enumeration and test all feasible combinations, but this is not a viable approach for larger problems.

We can use the aforementioned insight, leveraging the fact that dealing with continuous variables is easier than dealing with integer ones. The main basic insight of BB is to remove all the additional complexity stemming from those variables and to **relax**[2] all decision variables to be continuous so that the simplex method can be applied. The price

2: In our context, *relaxing* a decision variable is equivalent to make it continuous. Hence, the relaxation of a continuous decision variable is the variable itself. The relaxation of an integer decision variable $x_i \in \{L_i, L_i + 1, \cdots, U_i - 1, U_i\}$, where L_i and U_i are the integer lower and upper value x_i can take, is $x_i \in [L_i, U_i]$. Notation-wise, with $\{L_i, U_i\}$ we mean the set of integer numbers between L_i and U_i (which is a finite set), while with $[L_i, U_i]$ we mean the set of continuous numbers between L_i and U_i (which is an infinite set).

we pay is that the optimal solution of such a relaxed model might be mathematically optimal, but might not guarantee that every decision variable is of the correct type. For example, if we have an MILP where $x_1 \in [0, 4]$ is continuous and $x_2 \in \{0, 1, 2, 3, 4\}$ is integer, we might get $(x_1, x_2) = (2.5, 3.7)$ as the optimal solution of the relaxed problem (where $x_2 \in [0, 4]$ has been relaxed to be continuous). We need to take some extra measures to restore the feasibility (in terms of the nature of the decision variables) of such a solution. This is where the branching and bounding will come into play.

Note that, as already hinted at in Chapter 6, we implicitly used the aforementioned relaxation for Example 6.4. While decision variables x_1 and x_2 should be integers, we did not force them to be integers when using the simplex method. Because all the corner points of Example 6.4 are integer-valued, the final optimal solution is also integer-valued.

7.4 Linear relaxation, root node, and tree structure

Building on the preliminary insights of Section 7.3, we mentioned that to solve an MILP[3] we need to use the BB technique which is based on a tree structure. Such a tree structure is composed by two sets of elements: nodes and directed edges (see Chapter 11 for more info). **Each node defines a different linear relaxation of the original MILP, where a different subset of binary/integer variables has been relaxed to be continuous**. **Each directed edge defines a relationship between two nodes, and hence two different variations of the original MILP**. We define the **parent node** as the node where the edge originates, and the **child node** as the node where the edge ends. **Every child node inherits all the properties of its parent node, plus an additional constraint that is added to "reduce" the linear relaxation of the parent node**.

3: For the sake of simplicity, we will be considering an MILP, but the same logic applies to an BP or IP.

There exists only one special node in an BB tree structure that has no parent nodes, but only children nodes. This special node is called the **root node**. In the root node, **every decision variable is relaxed to be continuous**. The underlying principle of BB is to apply the simplex method to the root node and solve it to optimality. Once the optimal solution is found, all the decision variables are checked against their original type. If all integer decision variables satisfy their nature, then the optimal solution is integer[4], and there is no need to branch at all (this is what happened with the street food company solution of Example 6.4). Otherwise, from the parent node (the first parent node being the root node) two children nodes are created, where each of the two children nodes is given an additional constraint that tries to re-instate the nature of a decision variable that was fractional in the parent node. **In a classic decision tree fashion, the two constraints define mutually exclusive sets**.

4: For the sake of simplicity, we mean here both integer and binary decision variables, being a binary a special type of integer.

If the fractional decision variable (let us define it x_1) is binary, then one of the two children nodes will feature the additional constraint $x_1 = 0$, and the other one will have the additional constraint $x_1 = 1$. This example is shown in Figure 7.2, where we assume all decision variables are binary. On the other hand, if the fractional decision variable is integer (let us

assume it was defined as $x_1 = \{0, 1, \cdots, 9, 10\}$, hence a decision variable that can take any integer value from 0 to 10), and was returned with the value $x_1 = 3.6$ in our relaxed model, one of the two children nodes will inherit the additional constraint $x_1 \leq 3$ while the other children node will inherit the additional constraint $x_1 \geq 4$. This example is shown in Figure 7.3, where we assume all decision variables are integers. **Both examples do not refer to a complete BB decision tree, but are used for the sole purpose of providing some insights into the methodology. A fully developed BB solution is showcased in Section 7.11.**

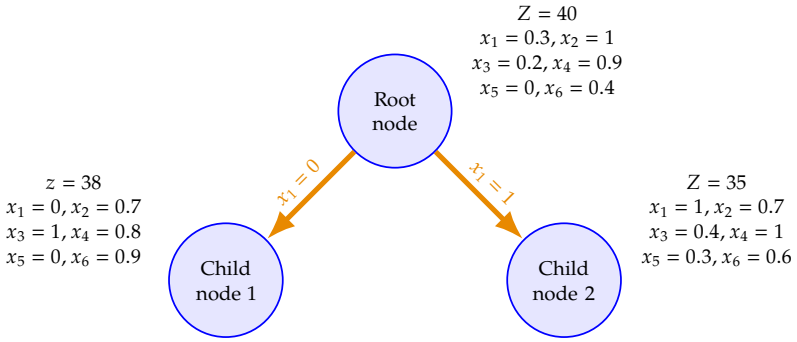


Figure 7.2: Tree structure of BB with binary-only decision variables. Note: in such a model, our goal is to maximize the objective function Z.

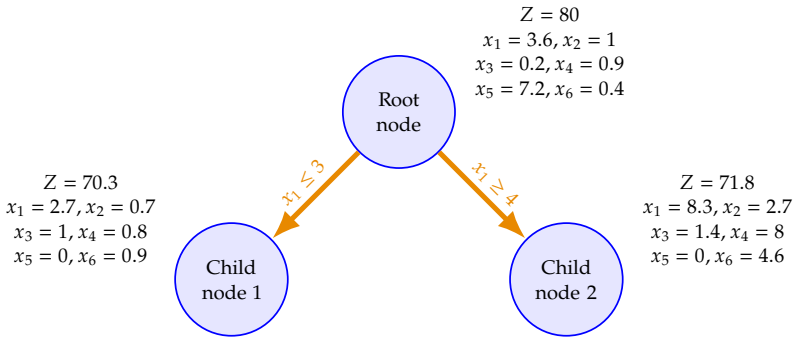


Figure 7.3: Tree structure of BB with integer-only decision variables. Note: in such model, our goal is to maximize the objective function Z.

We can highlight some similarities and differences between the examples shown in Figure 7.2 and Figure 7.3.

When it comes to similarities, we can notice how in both decision trees the quality of the solution degrades as we move from a parent to a child node (hence, Z decreases for a max problem and increases for a min problem). This is correct, because **every child node inherits all the constraints of the parent node, plus an additional constraint (the constraint depicted on the branch connecting the parent with the child). Hence, the child node represents a model that is more constrained than the parent, resulting in an objective that can only be worse**. Another similarity is that, in both decision trees, no node has been identified that yields a solution satisfying all the integrality constraints. In Figure 7.2 and Figure 7.3, both children nodes feature at least one decision variable that is fractional, hence not satisfying the binary (resp. integer) nature of the decision variables. Hence, in both cases we do not have a solution yet that is implementable in practice. Finally, in both cases multiple decision variables were fractional in the parent node (x_1, x_3, x_4, and x_6 in Figure 7.2, x_1, x_2, x_3, x_4, and x_6 in Figure 7.3). In both situations, we decided to use x_1 when adding constraints the the children nodes, but this choice was arbitrary. As will be explained in Section 7.8, there are different rules to decide which relaxed decision variable to use when

creating the two children nodes. **While different choices might increase or decrease the convergence time of the BB process, this will not affect the final solution as long as the BB decision tree is fully explored**.

In terms of differences, a major differences between fractional binary decision variables (Figure 7.2) and fractional integer decision variables (Figure 7.3) is the following. In the first case, in the two children nodes we are enforcing the value of such decision variable, as the two disjoint constraints are $x_1 \leq 0$ and $x_1 \geq 1$. Given than the original binary decision variable (i.e., $x_1 \in \{0, 1\}$) was relaxed to be continuous (i.e., $x_1 \in [0, 1]$), the two constraints become $x_1 = 0$ and $x_1 = 1$. In the second case, given the fractional integer decision variable, we compute the floor and ceiling of the fractional value (in the example, we have resp. $\lfloor 3.6 \rfloor = 3$ and $\lceil 3.6 \rceil = 4$), and impose that the decision variable should be smaller or equal to the floor, or greater or equal to the ceiling (i.e., $x_1 \leq 3$ and $x_1 \geq 4$ resp.). As such, in both children nodes the decision variable could still be fractional (as a matter of fact, it still is in Figure 7.3, with $x_1 = 2.7$ in the leftmost child node and $x_1 = 8.3$ in the rightmost child node). **Hence, when using a fractional integer decision variable the first time to separate[5] and create children nodes, we are generally not assigning a specific value to it in the two children nodes. Conversely, we are reducing the interval where the decision variable is defined**. In the example, we went from $x_1 \in [0, 10]$ in the parent node (the original decision variable is integer $x_1 \in \{0, 1, \cdots, 9, 10\}$ and has been relaxed to be continuous), to $x_1 \in [0, 3]$ and $x_1 \in [4, 10]$ in the left- and rightmost nodes, respectively. If we were to develop the BB decision tree further, we would probably reach a depth where even for fractional integer decision variables the additional inequality constraint becomes an equality constraint, but this is not generally achieved immediately, as it happens with fractional binary decision variables instead.

5: As we will elaborate more later, with the term *separation* we mean the process of creating two children nodes from a parent node, where each child node inherits an extra constraint related to the fractional integer decision variable that was selected for the separation.

7.5 Best bound, best incumbent, and gap optimality

We now use the definitions and knowledge acquired in Section 7.4 to take a step further and assess, at any point during the development of the BB decision tree, the quality of our solution. To do so, we provide an additional definition. We define **active node** a node in the BB decision tree that has not been solved or separated further with two branches (and associated children nodes). Because branches are associated with a fractional decision variable where each branch defines a smaller interval where that decision variable is defined (recall Figure 7.2 and Figure 7.3), **only nodes featuring a solution where at least one decision variable is fractional can be separated further. Hence, a node featuring a solution that is integer cannot be separated and hence cannot be an active node. For a similar reason, an infeasible node (i.e., a node that cannot be solved at all) cannot be further separated as well**. We will see in Section 7.7 how to deal with these situations.

This introduction paves the way for the two core concepts of **best bound** $\mathbb{BB}$ and **best incumbent** $\mathbb{BI}$.

Best bound (𝔹𝔹): the best bound is the best objective value Z (best means highest for a max problem, lowest for a min problem) across all **active nodes**. In both Figure 7.2 and Figure 7.3, the root node has been explored, while both children nodes are still active. In the first example, the current best bound is $Z = 38$ (leftmost child node), because that is the highest value across the two active nodes. In the second example, the best bound is $Z = 71.8$ (rightmost child node), because that is the highest value across the two active nodes. In essence, 𝔹𝔹 is a mathematically feasible "best" objective for our problem where at least one integer decision variable is fractional. Hence, it is an ideal solution that still provides a bound, as the name suggests, on our integer optimal solution. We elaborate on the special role of the root node in providing the first 𝔹𝔹 value in the **Root node and 𝔹𝔹** box.

Root node and 𝔹𝔹

In a BB tree, the root node is the only node that is a full linear relaxation of the original MILP, and hence it is the least constrained node where decision variables have the highest freedom. Hence, **in every BB tree the root node provides the initial value for 𝔹𝔹**. This means the highest value for a max problem and the lowest value for a min problem.

Best incumbent (𝔹𝕀): the best incumbent is the best objective value Z across all nodes with integer solutions. Note, again, that nodes featuring integer solutions cannot be active because they cannot be separated further with additional children nodes. **In both Figure 7.2 and Figure 7.3 we do not have a 𝔹𝕀 yet, as there is no node with a solution without any fractional decision variable**.

Readers should recall that the underlying principle of BB is to remove all the requirements on integrality of decision variables and to reinstate them little by little (via the branches). Hence, we start from the simplest problem (the root node) and every problem downstream will be slightly more complicated, inheriting all the additional constraints of all the branches leading back to the root node. We can use this insight to justify the following statement.

𝔹𝔹 cannot improve as more nodes are solved. This is because, as we reintroduce constraints on the integer nature of decision variables, we will make the fully relaxed model of the root node "less relaxed" or, in other words, more constrained. If we add constraints to a model, its optimal solution will either stay the same or get worse. As such, **𝔹𝔹 for a max problem is an upper bound, and will decrease over time. On the other hand, 𝔹𝔹 for a min problem is a lower bound, and will increase over time**. Analyzing Figure 7.2 again, we notice that the very first 𝔹𝔹 was 40 (objective value of the root node), while in the current setting 𝔹𝔹 dropped to $\max\{38, 35\} = 38$, i.e., the highest Z among the two active nodes (in this case, 𝔹𝔹 is associated with the leftmost child node).

In parallel, unless the original MILP is infeasible, eventually the first 𝔹𝕀 will be found. As more active nodes are explored, 𝔹𝕀 can only retain its current value or improve. This is true because, as more nodes are explored, the simplex method might find in a new node a combination of decision variables that satisfies all integrality requirements while

yielding a better Z than the current $\mathbb{BI}$. **As such, for a max problem $\mathbb{BI}$ will increase over time, while in a min problem $\mathbb{BI}$ will decrease over time**.

The considerations on $\mathbb{BB}$ and $\mathbb{BI}$ lead to the following insight. **For a max problem, we have that $\mathbb{BB} \geq \mathbb{BI}$ at any point during the BB process. For a min problem, we have that $\mathbb{BB} \leq \mathbb{BI}$ at any point during the BB process**. Because $\mathbb{BB}$ is an "ideal" solution where at least one integer decision variable is fractional, it is an overestimation for maximization problems and an underestimation for minimization problems. Hence, its value will degrade as more active nodes are explored. This is consistent with a decrease for max problems and an increase for min problems.

We can further exploit the relationship between $\mathbb{BB}$ and $\mathbb{BI}$ to define a parameter tracking the quality of our optimization process while BB is being performed. This parameter is called **optimality gap $\mathbb{OG}$** and is defined as

$$\mathbb{OG} = \left| \frac{\mathbb{BB} - \mathbb{BI}}{\mathbb{BI}} \right| \times 100 \tag{7.1}$$

The optimality gap is a percentile measure that assesses how far our $\mathbb{BI}$ is with respect to the "theoretical" optimal value of the MILP we are solving, i.e., $\mathbb{BB}$. Note that we use the absolute value in (7.1) so that the formula returns a positive percentage both for max problems (where $\mathbb{BB} \geq \mathbb{BI}$) and for min problems (where $\mathbb{BB} \leq \mathbb{BI}$).

We now focus on a maximization problem (the extension to a minimization problem will follow the same logic) to display a powerful relationship between the inequality $\mathbb{BB} \geq \mathbb{BI}$ and Equation 7.1. We stated that, in a max problem, $\mathbb{BB}$ is an overestimation of the real optimal value and hence will decrease over time without ever going lower than $\mathbb{BI}$, this is what the inequality $\mathbb{BB} \geq \mathbb{BI}$ ensures. Hence, if we use this knowledge in (7.1) we can claim that its numerator can never be negative and will reach 0 when $\mathbb{BB} = \mathbb{BI}$.

As a consequence, we can claim that the optimal solution of an MILP model is found when $\mathbb{OG} = 0\%$. This condition implies that the best bound has decreased enough, as sufficient nodes in the BB decision tree have been solved, to "touch" the current best incumbent. Hence, the current $\mathbb{BI}$ is the optimal solution.

In addition, while optimality is mathematically proven when $\mathbb{BB} = \mathbb{BI}$, an $\mathbb{OG}$ greater than 0% does not mean our current $\mathbb{BI}$ is not the optimal one. It could be the case that $\mathbb{BI}$ cannot be improved already, but that more nodes need to be explored so that $\mathbb{BB}$ can decrease to close the gap. We elaborate on this concept in the **Relationship between $\mathbb{GO}$ and optimality of a solution** box.

An important reflection on $\mathbb{OG}$ is needed to wrap up the topic. We showed the mathematical relationship that ensures an MILP is solved to optimality, i.e., $\mathbb{OG} = 0\%$. This being said, for large problems the number of nodes to be explored to drive $\mathbb{OG}$ to 0% might be extremely high and may even strain the computer processing the problem, potentially leading to out-of-memory issues. While the actual performance changes from problem to problem at hand, it is usually wise to set a "desired"

Relationship between $\mathbb{GO}$ and optimality of a solution

For the sake of clarity, let us consider the following example. We have an MILP we want to maximize and, given the current status of the BB process, we have $\mathbb{BB} = 200$ and $\mathbb{BI} = 100$, resulting in an $\mathbb{OG} = 100\%$. Does it mean we are 100% off from the optimal solution? Not necessarily. The optimal solution could be slightly lower than 200. It cannot be exactly 200, because otherwise the node currently associated with the $\mathbb{BB} = 200$ would have yielded a non-fractional solution. Going to the other extreme of the spectrum, it could be that our current $\mathbb{BI} = 100$ is already the optimal solution, but the BB decision tree needs to be explored further so that the best bound can decrease. It could also be that the optimal solution falls within the $[100, 200]$ interval, and more nodes need to be solved so that both $\mathbb{BB}$ decreases and $\mathbb{BI}$ increases.

gap optimality, say 5% for example, and stop the BB process anytime the current $\mathbb{OG}$ is below that threshold. Current BB algorithms generally excel at quickly reducing the optimality gap, yet they face challenges in efficiently eliminating the remaining gap. **In practical applications, prioritizing a slightly sub-optimal solution computed in advance may be more practical than grappling to prove optimality and potentially obtaining it too late for real-world application**.

7.6 A note on functional constraints

Up until now, we put a lot of focus on **fractional solutions** in an BB, i.e., solutions of nodes where at least one integer decision variable is fractional. We did not explicitly state, but are stating now for the sake of clarity, that **in any node of the BB process, functional constraints should be satisfied. They might, of course, be satisfied using a fractional value (if allowed) for the relaxed decision variables as this is one of the pillars of the BB routine**.

Let us consider the following example. We are solving an MILP with two decision variables: $x_1 \in \{0, 1\}$ (a binary) and $x_2 \in \{0, 1, 2\}$ (an integer). One of the constraints of our problem is $x_1 + 2x_2 \leq 2$. Solving the root node (where, because of the full relaxation, $x_1 \in [0, 1]$ and $x_2 \in [0, 2]$), we obtain a solution that features $x_1 = 0.37$ and $x_2 = 1.72$ and we decide to separate on x_2. One of the child nodes will inherit the additional constraint $x_2 \in [0, 1]$ and the other one the additional constraint $x_2 = 2$.

For the first child node, the constraint can still be satisfied by many combinations of the (x_1, x_2) decision variables. For example, if $(x_1, x_2) = (0, \frac{1}{2})$, then $1 \times 0 + 2 \times \frac{1}{2} = 1 \leq 2$. Hence, the functional constraint is satisfied, although the resulting solution is still fractional. For the second child node, because the additional constraint imposes $x_2 = 2$, we cannot leverage the fact that x_1 is still fractional. Even if we set $x_1 = 0$, then $1 \times 0 + 2 \times 2 = 4 \geq 2$, hence this child node will result in an infeasible solution. This example serves also as a link to the next topic covered in Section 7.7, i.e., fathoming rules.

7.7 Fathoming rules

In Section 7.3-Section 7.6 we elaborated on some aspects on the BB process and even dared to label it as “efficient" sometimes, but did not support this statement yet. Readers might wonder about the benefit of BB over full enumeration, as solving various models during branching introduces complexity through added constraints. The answer to this question is **fathoming**, i.e., a property of BB that allows to stop (fathom) the exploration of some portions of the decision tree, based on knowledge of the current $\mathbb{BB}$ and $\mathbb{BI}$.

There exist three ways a node can be fathomed. We list them in the following naming them fathoming of the first, second, and third type respectively[6]:

- **fathoming of first type**: a node is associated with an **infeasible model**. Because no solution is obtained, no further children nodes can be defined;
- **fathoming of second type**: a node is associated with a **feasible model yielding an integer solution**. If the objective value is lower than the current $\mathbb{BI}$, then the node is simply fathomed as such a node does not contribute to improving $\mathbb{BI}$. If the objective value Z is higher than the current $\mathbb{BI}$[7], then we still fathom the node (as there are no more fractional variables to separate), but we update $\mathbb{BI}$ with the new best value Z ($\mathbb{BI} \leftarrow Z$);
- **fathoming of third type**: a node is associated with a **feasible model yielding a fractional solution that is worse than the current $\mathbb{BI}$**. In principle, we could explore further the current node by selecting one of the fractional decision variables and defining the two children nodes. Every child node, as discussed in Section 7.3 and Section 7.5, is characterized by an objective that cannot be better than the objective of the parent node. Hence, as the objective of the parent node is already worse than $\mathbb{BI}$[8], such a parent node is fathomed and not explored further: any integer solution we might find downstream will be worse than $\mathbb{BI}$ anyway. **This fathoming rule is by far the most powerful in allowing an efficient exploration of an BB decision tree as it prevents the unnecessary exploration of portions of the decision tree**.

6: In the literature, variations might exist in the number and sequence of fathoming options.

7: We assume a max problem. For a min problem, the opposite holds.

8: Again, for a max problem worse means a lower objective value, for a min problem worse means higher.

7.8 Branching, bounding, and separation rules

A decision tree, including the BB decision tree, can be explored in various ways. For instance, in the examples of Figure 7.2 and Figure 7.3, after solving the root node and creating the children nodes, the decision remains on whether to solve the left or right child node first. Prioritizing certain nodes over others in a decision tree does not alter the outcome but can expedite convergence by leveraging fathoming rules or other model properties to eliminate redundant parts of the tree. In this section, we elaborate on this topic.

We first better formalize a couple of definitions. We define $\mathcal{A}$ the set of active nodes introduced in Section 7.5. As a reminder, nodes are labeled as active if they have not been solved yet. When a node is solved, we either

fathom it or subdivide it into two children nodes using a separation rule. Either way, a node is removed from $\mathcal{A}$ once solved.

If the node was associated with a feasible model, it is added to set $\mathcal{S}$, i.e., the set of **solved nodes**. **An important feature of $\mathcal{S}$ is that when the two children nodes of a parent node are both added to it, then the parent node is removed from it instead**. Let us clarify this point. If a node is further subdivided into two children nodes, it means that the node features a non-integer solution. Each child node, once solved, can output one of the following three outcomes. The node is infeasible. The node yields a non-integer solution which, given what was discussed in Section 7.4, is a "tighter" bound (tighter means lower for a max and higher for a min problem) for the problem. The node yields an integer solution. **Whichever combination characterizes the two children nodes, once they are both solved they "dominate" their parent node, which can be removed from $\mathcal{S}$ because its information is now redundant as it has been passed over to the children**. Note that $\mathcal{S}$ contains, at any point during our BB process, the current $\mathbb{BB}$ and $\mathbb{BI}$ values. For a max problem they are, respectively, the highest objective of a node with a non-integer solution and and the highest objective of a node with an integer solution.

Efficiently exploring a BB decision tree involves selecting a node from $\mathcal{A}$ to branch on and solve, followed by determining the fractional decision variable for creating two children nodes and their associated subproblems.

7.8.1 Branching options

Two main options exist when it comes to deciding which node in $\mathcal{A}$ to branch on[9] next: **backtracking** (also known as Last In First Out (LIFO)) and **jumptracking**.

In backtracking, we always **branch on the most recent node added to $\mathcal{A}$**. When separating on a fractional decision variable, two children nodes are added (formally) simultaneously and are hence the most recent additions to $\mathcal{A}$. Because which of them branch on next is still an open question, a solution might be to branch on the child node where the decision variable has been rounded down. For example, if we are separating on $x_1 = 0.3$ (with $x_1 \in \{0, 1\}$ being originally binary), we would branch on the child node where $x_1 = 0$. On a similar note, if we are separating on $x_2 = 3.7$ (with $x_2 \in \{0, 1, 2, 3, 4, 5\}$ being originally integer), we would branch on the child node where $x_2 \leq 3$.

In Figure 7.4 we provide an example of backtracking where we apply the aforementioned policy. We start branching on the root node (node 0) and then move to node 1 (one of the two children of node 0). Then, we keep branching on a rounded down decision variable until we solve node 3, which is fathomed. We then backtrace to node 4, then 5, then 6 and 7. Only now, we move back to the first layer and to node 8, which was the second child node of the root node. We repeat a similar process until node 14.

We now describe the second branching option, i.e., jumptracking. In such a case, the BB algorithm can select any active node from $\mathcal{A}$. **Usually,**

9: When it comes to the definition of branching, we show consistency with Carter et al., 2018 and mean which active node to solve (and from which, if needed, create two new children nodes). Hence, we *branch on a node* and *separate on a variable*. In some other references (see *Wikipedia: Branch & Cut* 2024), branching is used to describe the creation of the two children nodes stemming from the current solved node, hence it involves *branching on a variable*. We hope this sidenote might avoid confusion if readers are more familiar with a different terminology.

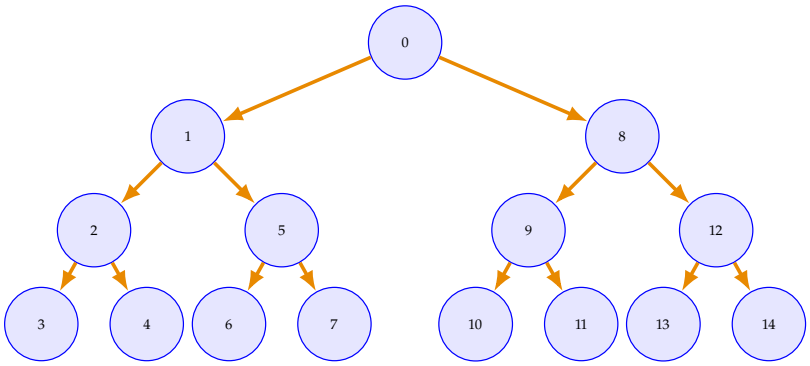


Figure 7.4: Example of backtracking strategy. The ordering of the nodes represents the sequence in which they are solved. Note: in our example, for every couple of children nodes, the left one is associated with a rounding down and the right one with a rounding up. In our policy, we always explore the rounded-down node first.

the selection is not random but follows a logic that leverages the information available. For example, branching on the node that is more likely to yield the highest objective possible should be beneficial in identifying a better integer solution and then improve $\mathbb{BI}$, albeit such a node might lead to infeasible solutions or solutions worse than the current $\mathbb{BI}$.

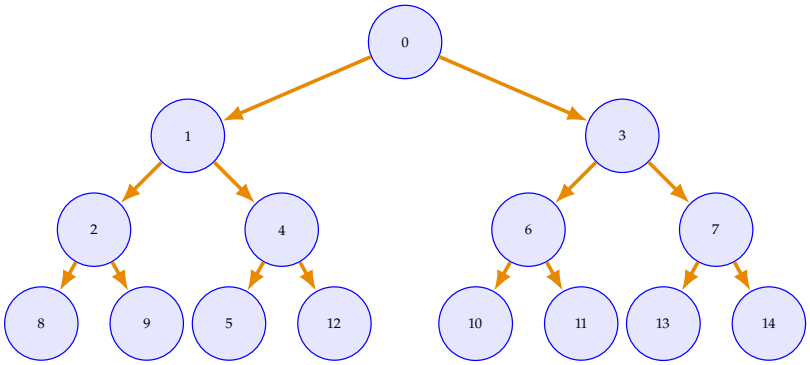


Figure 7.5: Example of jumptracking strategy. The ordering of the nodes represents the sequence in which they are solved.

In Figure 7.5 we provide an example of jumptracking. Differently from backtracking, no clear pattern can be (explicitly) recognized in the sequence of solved nodes.

7.8.1.1 Advantages and disadvantages of the different branching options

Readers may debate the effectiveness or efficiency of the two strategies and their merits. However, with no one-size-fits-all solution (refer to Section 7.10), each approach has its pros and cons.

The advantage of backtracking is that, by solving nodes that have generally a child-parent-grandparent relationship, we are solving very similar LPs. By efficiently storing the information of the final simplex tableau of a parent, a child node entails "just" the addition of a single constraint. This is algorithmically very efficient[10] and the optimal solution of the child node can be computed quickly. This is evident if we analyze again Figure 7.4. To solve node 1, we can extract and modify accordingly all the information from the final simplex tableau of node 0 by adding one additional single constraint. The same logic applies when we branch on node 2 from 1, etc. **This algorithmic efficiency does not necessarily result in a fast convergence, because of the quite rigid sequence in which nodes are solved**. Let us assume that, in Figure 7.4, a node on the far right side is

10: We refer readers to Hillier and Lieberman (2015) for a thorough explanation of this process.

associated with the optimal solution With our backtracking policy, we will have to explore every node on the left side, most of which might be unnecessary explorations.

Conversely, if we apply jumptracking with the aforementioned policy of branching on the node from $\mathcal{S}$ associated with the current $\mathbb{BB}$, we might avoid going very deep in a part of the tree which can instead be fathomed at a shallower level, hence reducing the overall number of nodes to explore. Because of the well-known "no free-lunch theorem", the disadvantage of this approach is correlated to the advantage of backtracking. Because of the potential sudden "jumps" from one side to the other of the BB decision tree, the algorithm might have to solve a completely different LP, whereas in the backtracking case every new LP inherits one or just a few constraints with respect to its predecessor.

In conclusion, the memory storage and computational time of each individual LP may pose challenges for jumptracking (and serve as strengths for backtracking). However, the intrinsic advantage of jumptracking (and drawback of backtracking) lies in enabling a more efficient exploration of the BB tree.

7.8.2 Bounding and separation rules

Let us assume that, in a BB decision tree, we just branched on a new node using one of the branching options from Section 7.8.1. Let us also assume that the LP associated with such a node yields a fractional solution where multiple integer decision variables are fractional. We are faced with the dilemma of choosing the fractional basic decision variable to separate so that two new subproblems (children nodes) can added. In each of them, the rounding down and up will have a similar effect in terms of direction, i.e., that the objective will worsen (decrease for a max problem). Yet, we do not know the severity of such degradation. It turns out we are not so blind, as we can leverage the information of the optimal simplex tableau of the parent node to assess the minimum loss in the objective if we were to round down or up a fractional basic variable. Because the process we are about to explain provides the minimum loss (hence, **a lower bound on the expected degradation of the solution**), it is labeled **bounding**.

Before diving into the formulas, let us start with an example. Let us take the street food company problem of Example 6.1 that has accompanied us for the entirety of Chapter 6 and consider a slight variation. Because of the surging prices of materials needed for the trucks, their price has increased from 30,000 to 35,000€ and from 40,000 to 45,000€. We can write the mathematical formulation of such a variant as:

$$\max \quad Z = 2x_1 + 5x_2 \tag{7.2}$$

s.t.:

$$x_1 \leq 8 \tag{7.3}$$
$$x_2 \leq 6 \tag{7.4}$$
$$\frac{7}{2}x_1 + \frac{9}{2}x_2 \leq 36 \tag{7.5}$$
$$x_1, x_2 \in \mathbb{N}_0 \tag{7.6}$$

where in (7.5) we updated the coefficients of x_1 and x_2 to map the increase in price. In addition, we now acknowledge we are dealing with an IP and not an LP in (7.6), where, as a reminder, $\mathbb{N}_0$ represents the set of non-negative integers.

While we will be formally describing the BB algorithm in Section 7.9, we have already mentioned that the first step when solving, in this case, an IP is to relax all the decision variables to be continuous and solve the root node that starts the decision tree. If we solve the LP associated with the root node, we obtain the optimal simplex tableau shown in Table 7.1.

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	0	0	0	2.429	0.571	35.14
(x_3)	0	0	0	1	1.286	-0.286	5.429
(x_2)	0	0	1	0	1	0	6
(x_1)	0	1	0	0	-1.296	0.286	2.571

Table 7.1: Optimal simplex tableau for the full linear relaxation (root node) of the variant of the street food company problem of Example 6.1 with increased truck cost.

It is noteworthy that, despite having the flexibility to assume continuous values, x_2 is fixed at 6. **The substantial advantage in attracting customers of trucks of the second type outweighs the price increase, prompting the (relaxed) model to prioritize acquiring as many as feasible**. Conversely, x_1 is fractional in such a solution as $x_1 = 2.57$. In the current optimal solution x_3 is also fractional, but being a slack variable and not an original decision variable it is allowed to take fractional values anyway. We expand on this important aspect in the 💡 **A note on which decision variables should be considered when separating** box.

💡 A note on which decision variables should be considered when separating

In every MILP, according to what it represents in practice, we require certain decision variables to be integer or binary. We also extensively discussed how the simplex method needs an augmented form of an LP to apply row operations. Hence, anytime the simplex method is applied to a relaxation of an MILP, only fractional decision variables that belong to the original model should be considered for separation. Augmented (slack, artificial, or surplus) decision variables are not part of the original MILP and can take continuous values as they are exploited to ensure balance between left- and right-hand sides in all constraints. As such, they should not be considered in the BB decision-making process.
This being said, if an MILP is characterized by integer-only coefficients in the objective function and all functional constraints (both left- and right-hand sides), then in the optimal solution also augmented variables will be integer.

Going back to our example, because x_2 is integer-valued, we are already guaranteed we will separate on x_1 and introduce two children nodes, one where $x_1 \leq 2$ and the other where $x_1 \geq 3$. In other situations, we might have several fractional decision variables, and hence getting some indication on which one is more promising for the separation might expedite the convergence of the BB process. We now show how the previously introduced bounding process works using the information contained in Table 7.1. Let us consider fractional basic variable x_i[11], and let f_i be the fractional residual that makes such a variable non-integer. In our example, we only consider x_1 where $f_1 = 0.57$. In addition, let us define a_{ij} the coefficient in the tableau in the row mapping basic variable x_i and in the column mapping variable x_j, and c_j the coefficient in the objective row mapping variable x_j. We define the **down penalty** D_i, i.e., the minimum reduction in the objective if we round down fractional basic variable x_i as

11: We inherit our notation from Carter et al., 2018.

$$D_i = \min_j \left\{ \frac{c_j f_i}{a_{ij}} \quad \forall j \text{ s.t. } a_{i,j} > 0 \right\} \tag{7.7}$$

while we define the **up penalty** U_i, i.e., the minimum reduction in the objective if we round up fractional basic variable x_i as

$$U_i = \min_j \left\{ \frac{c_j(f_i - 1)}{a_{ij}} \quad \forall j \text{ s.t. } a_{ij} < 0 \right\} \tag{7.8}$$

In our case, we only need to consider x_5 for D_1 and x_4 for U_1. We compute $D_1 = \frac{0.571 \times 0.571}{0.286} = 1.14$ and $U_1 = \frac{2.429 \times (-0.429)}{(-1.286)} = 0.81$ (for more technical details pertaining (7.7) and (7.8) we refer readers to Carter et al., 2018 or Salkin et al., 1989).

In cases where numerous integer decision variables are fractional, Carter et al., 2018 proposes selecting the decision variable with the highest penalty (either down or up) to generate two children nodes. The objective is to establish a branch with potential and another with less potential, aiming to explore the promising branch for a potential integer solution while swiftly fathoming the less promising one. Hence, the idea is to branch on the child node more likely to yield a better objective.

Our example can be depicted by the partial BB decision tree depicted in Figure 7.6. Note that in the two children nodes, we are using an inequality to express the maximum value that Z can take based on the D_1 and U_1 values determined before. This is a key concept. If we wanted, given the current parent node, to be sure to branch on the child node yielding the best objective value, we could try to separate using all fractional variables, solve the LPs of both children nodes, analyze all the results, and assess which decision variable we should use to separate and which child node to branch on. This approach entails applying the full simplex method to several LPs, which might be computationally heavy. **The approach proposed here used the optimal tableau of the root node (which is already available anyway) to compute a set of D_is and U_is by inspecting some values from the tableau and applying (7.7)-(7.8) (which is a much less demanding computation) to get an insight into which fractional variable we should separate and which child node to**

solve. The trade-off is that we only obtain an estimated objective value. Consequently, the decision variable and branch chosen for separation might not be as optimal as in the approach with complete information.

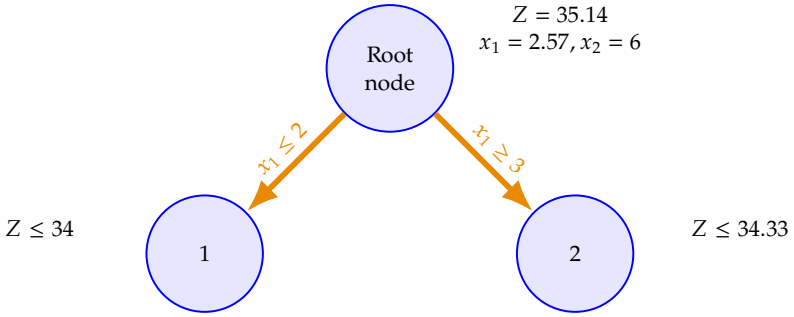


Figure 7.6: Root node and bounds on the objective value of the two children nodes of the variant of the street food company problem of Example 6.1 with increased truck cost.

In our example, there is no ambiguity regarding which fractional decision variable to separate, as only x_1 is fractional. When it comes to which child node to explore, our insight based on D_1 and U_1 suggests that the branch associated with the round up should be prioritized. For the sake of the example, let us solve both LPs and report the results in Figure 7.7.

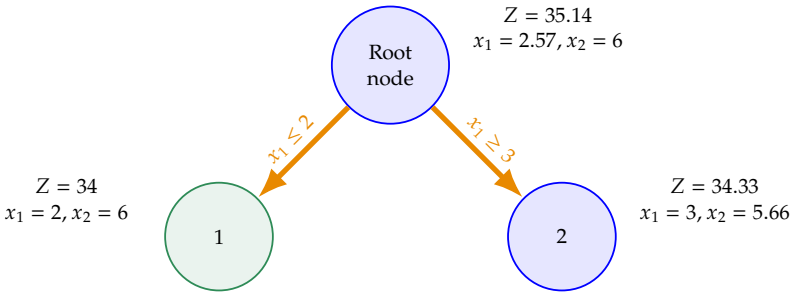


Figure 7.7: Root node and solved children nodes for the variant of the street food company problem of Example 6.1 with increased truck cost. Node 1 is colored in green as it resulted in an integer solution.

We notice that, in both cases, the bound on the objective turned out to be accurate, as both objective values matched the bound. In addition, we colored node 1 in green because it yielded an integer solution. Given what we discussed in Section 7.7, $\mathbb{BI} = 34$ and node 1 is fathomed. In addition, $\mathbb{BB} = 34.33$ because the root node is now dominated by the two children. For the sake of completeness, let us branch on node 2 (the only node we can branch on) and separate on x_2 (the only decision variable we can separate). We report the updated BB tree in Figure 7.8.

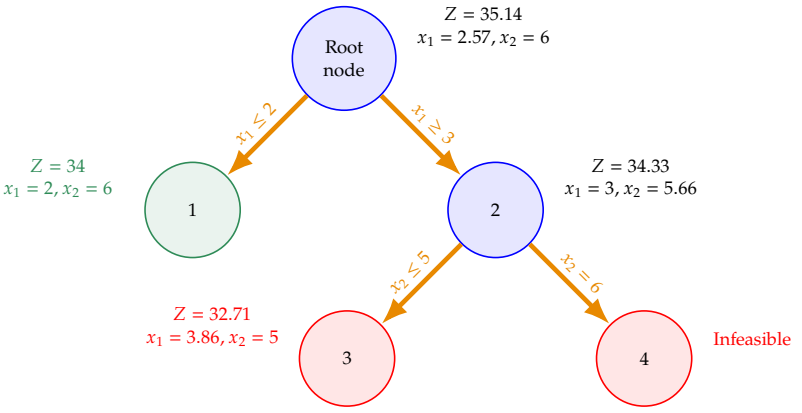


Figure 7.8: Complete BB tree for the variant of the street food company problem of Example 6.1 with increased truck cost. Nodes 3 and 4 are colored in red as they are fathomed.

We can notice that the solution $Z = 34$ is optimal, because node 4 is infeasible (and is hence fathomed), while node 3 is fathomed because it yielded a fractional solution worse than $\mathbb{BI}$ (fathoming of third type).

Crucially, node 3 was fathomed due to the prior resolution of node 1. In the absence of an integer solution, branching on node 3 with children nodes defined by $x_1 = 3$ and $x_1 \geq 4$ would lead to the same solution but extend the exploration process. This example underscores that diverse branching and separation strategies do not alter the final BB solution but can significantly impact the efficiency of the algorithm.

7.9 The BB algorithm in a nutshell

Having covered all the different features of the BB procedure, we can now focus on a more formal description of its algorithm.

The algorithm follows quite strictly the rules that we discussed so far in an integrated fashion. Given an MILP, the algorithm first relaxes all integer decision variables to create the root node, add it to $\mathcal{A}$, and solve it. If the root node is infeasible, this means the original MILP is infeasible as well. If not, the root node is moved to $\mathcal{S}$, the $\mathbb{BB}$ is initialized, and, given the adopted separation rule, two children nodes are created from the root node and added to $\mathcal{A}$. Then, according to the branching rule, a node from $\mathcal{A}$ to branch on is selected and solved. Then, the fathoming rules are checked to assess if the node should be fathomed or not, and the process is repeated. As soon as a node in $\mathcal{S}$ yields an integer solution, then the $\mathbb{BI}$ is initialized as well. As more nodes are explored, $\mathbb{BB}$ is updated, and if a node in $\mathcal{S}$ with an integer solution yielding a better (for a max problem, this means higher) objective is solved, then $\mathbb{BI}$ is updated as well. The process continues until $\mathbb{GO}$ (recall (7.1)) falls below a pre-determined threshold or if a time limit is reached. We summarize the algorithm in the **BB algorithm (for a** max **problem)** box.

If optimality must be proven, then $\epsilon = 0$ should hold so that $\mathbb{GO} = 0 \implies \mathbb{BB} = \mathbb{BI}$. **In practice, state-of-the-art solvers use values such as $\epsilon = 0.01\%$ as it is algorithmically hard, and unnecessary, to converge to 0%. Note that, in some other references (see Carter et al., 2018 for example), the stopping criterion is when the set of active nodes $\mathcal{A}$ is empty. The two conditions $\mathbb{GO} = 0$ and $\mathcal{A} = \emptyset$ are in fact equivalent.**

BB algorithm (for a max problem)

- **Inputs**:
  - original MILP;
  - ϵ: threshold on $\mathbb{GO}$ (e.g., 5%);
  - T_M: threshold on maximum computational time (e.g., 3,600 s).
- initialize elapsed time $t_e = 0$;
- initialize $\mathbb{GO} = \infty$;
- initialize $\mathbb{BB} = \infty$, $\mathbb{BI} = -\infty$;
- initialize $\mathcal{A} = \emptyset$, $\mathcal{S} = \emptyset$;
- **create the root node** by relaxing all integer decision variables of the MILP. Add the root node to $\mathcal{A}$;
- solve the root note, initialize $\mathbb{BB}$ with the root node objective;
- **decide which fractional variable to separate on** (according to the separation rule as shown in Section 7.8) and generate the two children nodes. Add the two children nodes to $\mathcal{A}$;
- remove the root node from $\mathcal{A}$ and add it to $\mathcal{S}$;
- WHILE $t_e \leq T_M \vee \mathbb{GO} \geq \epsilon$:
  - **select which node from $\mathcal{A}$ to branch on** and solve (according to the branching rule as shown in Section 7.8)
    - IF the node satisfies one of the fathoming conditions (see Section 7.7), **fathom it**. If the node provides an integer solution $Z \geq \mathbb{BI}$, then add it to $\mathcal{S}$ and update the best incumbent value; $\mathbb{BI} \leftarrow Z$
    - ELSE the node provides a fractional solution $Z \geq \mathbb{BI}$. **Add the node to $\mathcal{S}$, then decide which fractional variable to separate on** (according to the separation rule as shown in Section 7.8) and generate the two children nodes. Add the two children nodes to $\mathcal{A}$.
  - remove the solved node from $\mathcal{A}$ and add it to $\mathcal{S}$;
  - **if both children of a parent node are in $\mathcal{S}$, then remove the parent node from $\mathcal{S}$** (as discussed in Section 7.8);
  - **update $\mathbb{BB}$** as the highest objective among all nodes in $\mathcal{S}$ with fractional solutions;
  - **Update t_e and $\mathbb{GO}$**.
- **Outputs**: $\mathbb{BI}$ and values of decision variables associated with that solution

7.10 Considerations on the algorithmic complexity of the BB algorithm

In the previous sections, we advertised the BB algorithm as being very efficient in exploring only the parts of the solution space of an MILP that are deemed worthy of exploration. In particular, the fathoming rules play an important role in preventing the algorithm from exploring solutions that would not lead to any improvement in our objective. **This being said, there is still a plethora of parameters and tweaks that can affect the computational efficiency of BB**.

As a divide-and-conquer approach, BB involves breaking down a complex problem into smaller and easier subproblems through branching and systematically exploring the solution space. The algorithm's efficiency depends on the quality of the bounding mechanism used to discard subproblems that cannot lead to an optimal solution (see Section 7.7). Additionally, the branching strategy (see Section 7.8), which dictates the order in which subproblems are explored, plays a crucial role in determining the algorithm's performance. The overall complexity is influenced by the nature of the problem being solved, the problem size, and the specific characteristics of the objective function and constraints. **While BB offers a systematic and theoretically sound method for solving optimization problems, the efficiency of its practical implementation relies on fine-tuning these various components to suit the specific problem at hand**.

Additionally, for large-scale problems the curse of dimensionality is unavoidable. This means that such large problems are very seldom solvable to optimality (or within reasonable values of $\mathbb{GO}$) in a reasonable time-frame. This issue calls for algorithmic advancements. A little help can arrive from valid inequalities and cuts that we will briefly explain in Chapter 8, but, usually, this is not enough. Oftentimes, alternative solution approaches such as heuristics are employed to solve such large-scale problems. The advantage of such solution methods (some examples are Genetic Algorithm (GA), Large Neighborhood Search (LNS), or Tabu Search (TS) just to cite a few examples) is that they are generally faster than the BB process. The disadvantage, being non-exact methods, is that no proof of convergence is applicable and, hence, solution quality is hard to assess. Here, the exact BB formulation becomes crucial. **When achieving the optimal solution for the original problem is impractical in a reasonable time, the BB process can yield a $\mathbb{BB}$ and a $\mathbb{BI}$ within the allotted computational time. These serve as benchmarks against our heuristic solution**. For an effective heuristic in a max problem, its solution should surpass $\mathbb{BI}$ (outperforming the BB solution method). The gap between the heuristic solution and $\mathbb{BB}$ is akin to the optimality gap, indicating how much better our solution could theoretically be.

7.11 An illustrative example

We wrap up this chapter with an illustrative example where the BB algorithm is showcased in its entirety. Because the scope of the example is to help the reader familiarize with the concepts covered in the previous

sections, we will analyze the process step-by-step reporting the most important sets and parameters such as $\mathcal{A}$, $\mathcal{S}$, $\mathbb{BB}$, and $\mathbb{BI}$.

Furthermore, the example aims to acquaint the reader with the BB process, not to evaluate or compare algorithmic performance across strategies. For this purpose, we opted for a custom branching strategy, resembling backtracking, but solving sequentially the two children nodes generated from a parent node at the same depth level, rather than delving deeply along a single branch.

Example 7.1 *We have spent a long day in the library trying to get the hang of the BB algorithm. To reward ourselves, we decided to go our for dinner with our friends in a pizzeria that allows customers to build their own pizza by providing a menu with extra toppings (with prices) to be added to a regular margherita (that costs 6€). While being happy because BB does not seem so daunting any longer, we are not spendthrift and set our maximum budget to 13€.*

Each topping has a different satisfaction value for us, turning our meal into an optimization problem: maximize satisfaction without surpassing the budget. Being OR enthusiasts, what better time to apply our newfound BB knowledge?

The menu, detailed in Table 7.2, categorizes toppings into cheeses, meats, and vegetables. Mentally assigning satisfaction scores S_t to each topping, higher values indicate stronger preferences. Recognizing the problem as a 0-1 KP (see Section 10.1.1), we associate each topping with a binary decision variable x_t (1 if added to our pizza). The extended menu, containing all inputs for solving the 0-1 KP, is presented in Table 7.3, where we also translated prices into a topping-specific parameter P_t.

Cheese		Meat		Vegetable	
Type	**Price**	**Type**	**Price**	**Type**	**Price**
Buffalo mozzarella	2€	Parma ham	3€	Zucchini	1€
Gorgonzola	1.5€	Pancetta	2€	Fried eggplant	2€
Ricotta	1€	Salame	2€	Cherry tomatoes	0.5€
Burrata	3€	'Nduja	0.5€	Roasted peppers	1.5€

Table 7.2: Menu with additional toppings and prices of Example 7.1.

Cheese				Meat				Vegetable			
Type	P_t	S_t	x_t	**Type**	P_t	S_t	x_t	**Type**	P_t	S_t	x_t
Buffalo mozzarella	2€	5.0	x_1	Parma ham	3€	7.2	x_5	Zucchini	1€	3.5	x_9
Gorgonzola	1.5€	3.1	x_2	Pancetta	2€	4.2	x_6	Fried eggplant	2€	5.2	x_{10}
Ricotta	1€	4.2	x_3	Salame	2€	8.3	x_7	Cherry tomatoes	0.5€	3.7	x_{11}
Burrata	3€	4.7	x_4	'Nduja	0.5€	4.8	x_8	Roasted peppers	1.5€	4.1	x_{12}

Table 7.3: Menu with additional toppings and prices of Example 7.1, plus satisfaction S_t and binary decision variable x_t for every topping t.

In this BP, we need only one set, i.e., the set of toppings $\mathcal{T}$ containing 12 elements: $\mathcal{T} = \{\text{buffalo mozzarella} = 1, \cdots, \text{roasted peppers} = 12\}$. In addition, because a margherita pizza costs 6€ and our overall budget is 13€ this leaves us with a remaining budget B of 7€ for our toppings. We can formulate the BP as:

$$\max \sum_{t \in \mathcal{T}} S_t x_t \tag{7.9}$$

s.t.:

$$\sum_{t \in \mathcal{T}} P_t x_t \leq B \tag{7.10}$$

$$x_t \in \{0, 1\} \tag{7.11}$$

where (7.9) defines the objective, i.e., maximizing our satisfaction, (7.10) is the budget constraint, and (7.11) defines the binary nature of the decision variables. Given the small size of the problem at hand, we can even expand all the terms as:

$$\begin{aligned} \max \quad & 5.0x_1 + 3.1x_2 + 4.2x_3 + 4.7x_4 + \\ & 7.2x_5 + 4.2x_6 + 8.3x_7 + 4.8x_8 + \\ & 3.5x_9 + 5.2x_{10} + 3.7x_{11} + 4.1x_{12} \end{aligned} \tag{7.12}$$

s.t.:

$$\begin{aligned} & 2.0x_1 + 1.5x_2 + 1.0x_3 + 3.0x_4 + 3.0x_5 + 2.0x_6 + \\ & 2.0x_7 + 0.5x_8 + 1.0x_9 + 2.0x_{10} + 0.5x_{11} + 1.5x_{12} \leq 7 \end{aligned} \tag{7.13}$$

$$x_1, \cdots, x_{12} \in \{0, 1\} \tag{7.14}$$

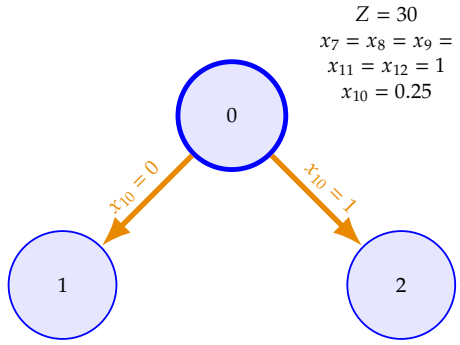
where (7.12), (7.13), and (7.14) are the expanded counterparts of (7.9), (7.10), and (7.11), respectively.

While we could use an off-the-shelf BB solver and directly solve the BP, we want to cement the knowledge we acquired at the library and go over the BB decision tree with a "manual" process. We hence start with the root node, where all 12 decision variables are relaxed to be continuous $\rightarrow x_t \in [0, 1] \ \forall t \in \mathcal{T}$.

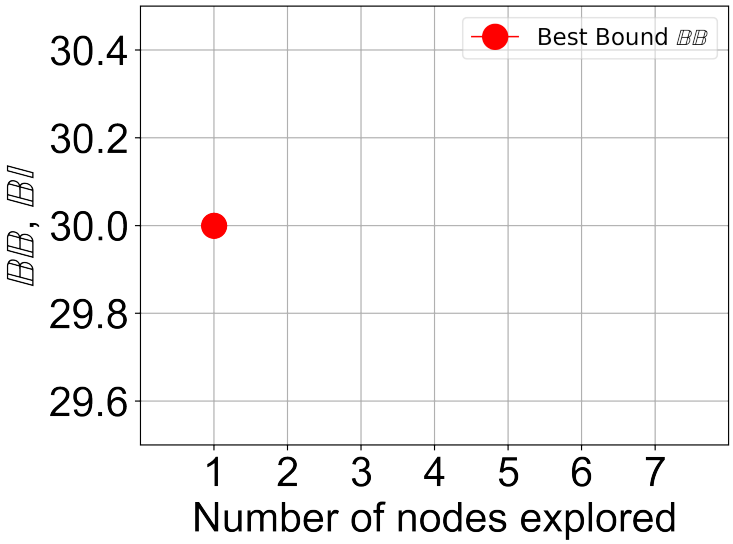
Implementing insights from Section 7.9, we initialize $\mathbb{GO} = \infty$, $\mathbb{BB} = \infty$, $\mathbb{BI} = -\infty$, $\mathcal{A} = 0$, and $\mathcal{S} = \emptyset$. Solving the root node (as illustrated in Figure 7.9a), we obtain a fractional solution recommending salame, 'nduja, zucchini, cherry tomatoes, roasted peppers, and $\frac{1}{4}$ of fried eggplant. Despite a possible plea for a minimal serving of eggplant, the pizzeria's policy prohibits such modifications. As x_{10} is the sole fractional variable, we perform separation on it, creating children nodes 1 and 2. We update the BB values as follows: $\mathbb{GO} = \infty$, $\mathbb{BB} = 30$, $\mathbb{BI} = -\infty$, $\mathcal{A} = \{1, 2\}$, and $\mathcal{S} = \{0\}$. In addition, in Figure 7.9 and all the following ones we will use the following color-scale: **red** for fathomed nodes, **green** for the node featuring the current best incumbent (albeit being formally fathomed as well), and thicker contours for solved[12] nodes.

12: We will be using the terms *solved* and *explored* interchangeably in the description.

Opting for node 1 (with the added constraint $x_{10} = 0$), we relinquish the fried eggplant. The solution of this modified LP remains fractional, recommending ricotta, salame, 'nduja, zucchini, cherry tomatoes, roasted peppers, and $\frac{1}{4}$ of buffalo mozzarella. Faced with another unsuccessful attempt at a partial portion of a topping, we perform separation on x_1, leading to the creation of children nodes 3 and 4. We depict this new scenario in Figure 7.10. The main values are updated as follows: $\mathbb{GO} = \infty$,



(a) Tree structure.



(b) Evolution of $\mathbb{BB}$ and $\mathbb{BI}$.

Figure 7.9: BB decision tree for the build your own pizza problem of Example 7.1: one node explored.

$\mathbb{BB} = 30$, $\mathbb{BI} = -\infty$, $\mathcal{A} = \{2, 3, 4\}$, and $\mathcal{S} = \{0, 1\}$. Note that the best bound is still equal to 30, as we have not solved yet node 2.

Recalling our branching policy to always explore both children nodes of a parent node at the shallowest level possible, we then branch on node 2, where the additional constraint $x_{10} = 1$ embodies our desire not to give up on the fried eggplant. Surprisingly, this time we achieve an integer solution: a step closer to satisfying our hunger. The solution we obtain entails a margherita with the addition of ricotta, salame, 'nduja, zucchini, fried eggplant, and cherry tomatoes, for an overall satisfaction $Z = 29.7$. This implies we have now a $\mathbb{BI}$. In addition, because the root node is now "dominated" by nodes 1 and 2, we eliminate it from $\mathcal{S}$ and update $\mathbb{BB} = 29.85$ (the objective value of node 1). We depict this new scenario in Figure 7.11. We modify the main BB as follows: $\mathbb{GO} = \left|\frac{29.85 - 29.7}{29.7}\right| \times 100 = 0.5\%$, $\mathbb{BB} = 29.85$, $\mathbb{BI} = 29.7$, $\mathcal{A} = \{3, 4\}$, and $\mathcal{S} = \{1, 2\}$.

Because node 1 is still characterized by a fractional, yet slightly better ($29.85 > 29.7$) solution than our current feasible option, we want to investigate if a different combination of toppings can yield better satisfaction. We then explore node 3. Solving the LP, we obtain a solution suggesting ricotta, salame, 'nduja, zucchini, cherry tomatoes, roasted peppers, and a fraction of Parma ham (x_5=0.17). Having given up on the possibility of

convincing the waiter, we separate using decision variable x_5 creating nodes 5 and 6. The only changes in the BB values pertain to the sets $\mathcal{A}$ and $\mathcal{S}$: $\mathbb{GO} = 0.5\%$, $\mathbb{BB} = 29.85$, $\mathbb{BI} = 29.7$, $\mathcal{A} = \{4, 5, 6\}$, and $\mathcal{S} = \{1, 2, 3\}$. See Figure 7.12 for the updated BB tree.

We now pick node 4 from the set of active nodes $\mathcal{A}$ as the next one to be solved. We obtain another integer solution consisting of buffalo mozzarella, ricotta, salame, 'nduja, zucchini, and cherry tomatoes. Despite the enticing mix of cheeses, meats, and vegetables, **its satisfaction value of 29.5 falls short of our current $\mathbb{BI}$ and is promptly fathomed (second type)**. Because parent node 1 is dominated by children nodes 3 and 4, $\mathbb{BB}$ is updated (and hence $\mathbb{GO}$). In addition, having fathomed the solved node, no additional children nodes are produced as part of the current iteration. Hence: $\mathbb{GO} = 0.34\%$, $\mathbb{BB} = 29.8$, $\mathbb{BI} = 29.7$, $\mathcal{A} = \{5, 6\}$, and $\mathcal{S} = \{2, 3, 4\}$. The revised BB tree is displayed in Figure 7.13.

We continue the exploration of our BB decision tree with node 5 (see Figure 7.14). Because its combination of ricotta, salame, 'nduja, zucchini, cherry tomatoes, roasted peppers, and $\frac{1}{4}$ of pancetta yields a fractional solution whose objective ($Z = 29.65$) is already lower than our $\mathbb{BI}$, then the node is fathomed (fathoming of third type). We depict the updated decision tree in Figure 7.14 and update the BB values as follows: $\mathbb{GO} = 0.34\%$, $\mathbb{BB} = 29.8$, $\mathbb{BI} = 29.7$, $\mathcal{A} = \{6\}$, and $\mathcal{S} = \{2, 3, 4, 5\}$.

Left with just one unexplored node (node 6), we uncover a solution featuring ricotta, Parma ham, salame, 'nduja, and roasted peppers. Despite the intimidating trio of meats, this node yields a feasible but inferior topping solution, $\rightarrow Z = 28.2 < \mathbb{BI}$. Consequently, the node is promptly fathomed (second type). With no remaining active nodes, our BB algorithm concludes. The optimal value is $\mathbb{BI} = 29.7$, and $\mathbb{BB}$ is updated as well to reflect convergence to optimality. Final BB values are $\mathbb{GO} = 0\%$, $\mathbb{BB} = 29.7$, $\mathbb{BI} = 29.7$, $\mathcal{A} = \emptyset$, and $\mathcal{S} = \{2, 5, 6\}$. Notably, in $\mathcal{S}$, node 3 is excluded, being dominated by nodes 5 and 6. The final BB tree is shown in Figure 7.15.

Having finally computed the optimal solution, we realize our day at the library was well-spent and we are positive we are one step closer to mastering the basics of OR. We happily build our pizza with the selected additional toppings ricotta, salame, 'nduja, zucchini, fried eggplant, and cherry tomatoes. A nice blend of cheese, meats, and vegetables to celebrate the achievement!

The model we "manually" solved using the BB process was intentionally kept exceptionally simple for concise step-by-step illustration. It, being a BP, exclusively incorporated binary variables, simplifying the separation process (one branch with $x_i = 0$ and the other with $x_i = 1$). However, despite its simplicity, it encompassed nearly every aspect of BB. We witnessed no infeasible node, hence we did not get the chance to fathom nodes using the fathoming of the first type. Conversely, we used fathoming of the second and third types. **We assessed how to update KPIs such as $\mathbb{BB}$, $\mathbb{BI}$, and $\mathbb{GO}$ and how to update sets $\mathcal{A}$ and $\mathcal{S}$. We also assessed how, in this case, the process was stopped because $\mathcal{A} = \emptyset$ (as described in Carter et al., 2018) and, as we had a $\mathbb{BI}$ at our disposal, such solution was labeled as optimal**.

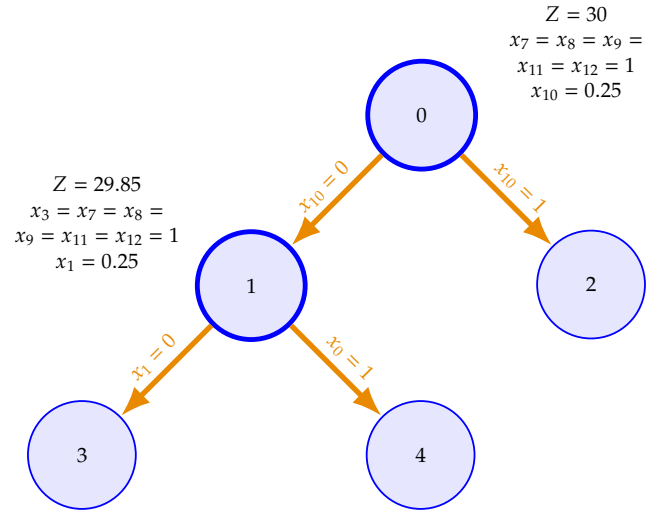
Coded example

A coded version of Example 7.1 is available here.

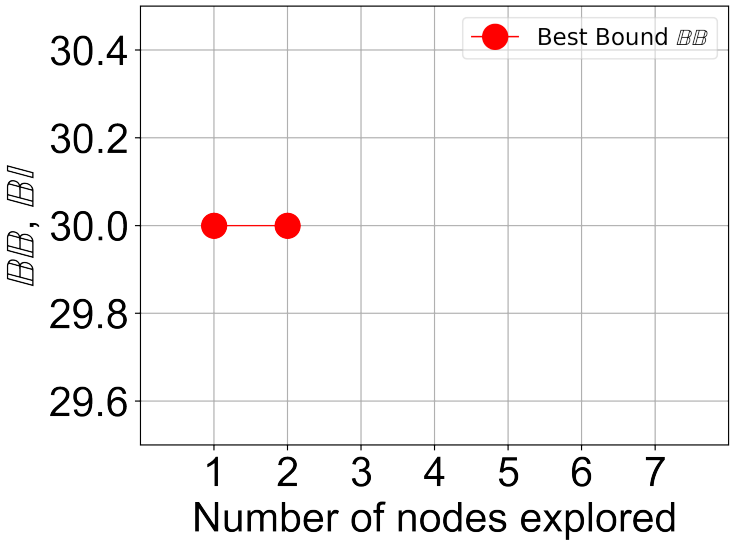
In the **An extension to Example 7.1** box, we challenge readers with a variation to the presented Example 7.1.

An extension to Example 7.1

Let us assume that, in the menu, we missed an asterisk forwarding us to the following footnote: *"Only one meat selection is allowed as extra topping"*. How would the original BP would change? Try to formulate this new model and solve it again using an BB solver. Can you already foresee what could/will happen to the quality of the final optimal solution?

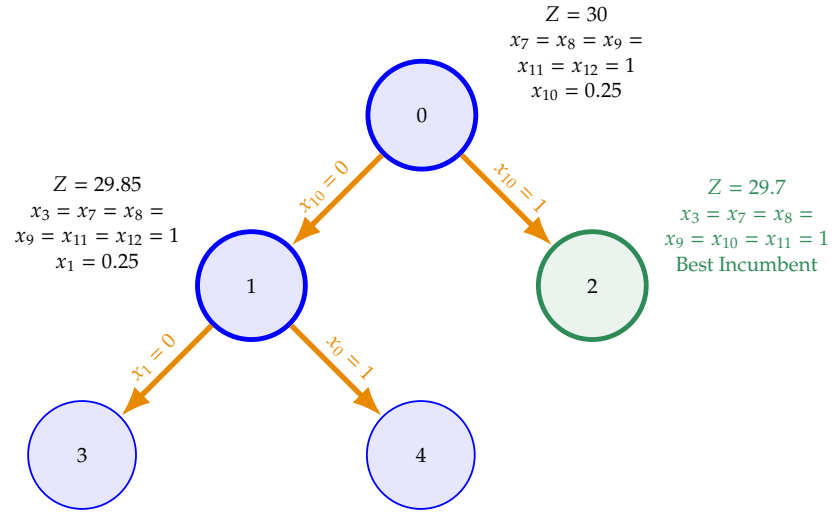


(a) Tree structure.

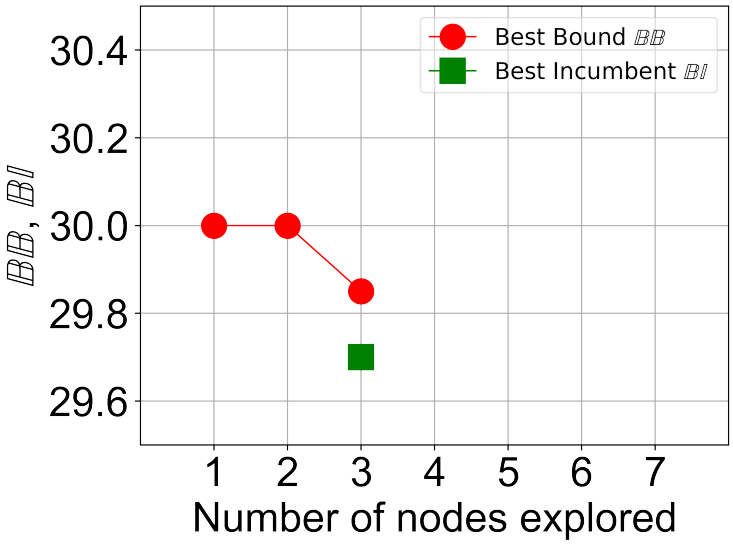


(b) Evolution of $\mathbb{BB}$ and $\mathbb{BI}$.

Figure 7.10: BB decision tree for the build your own pizza problem of Example 7.1: two nodes explored.

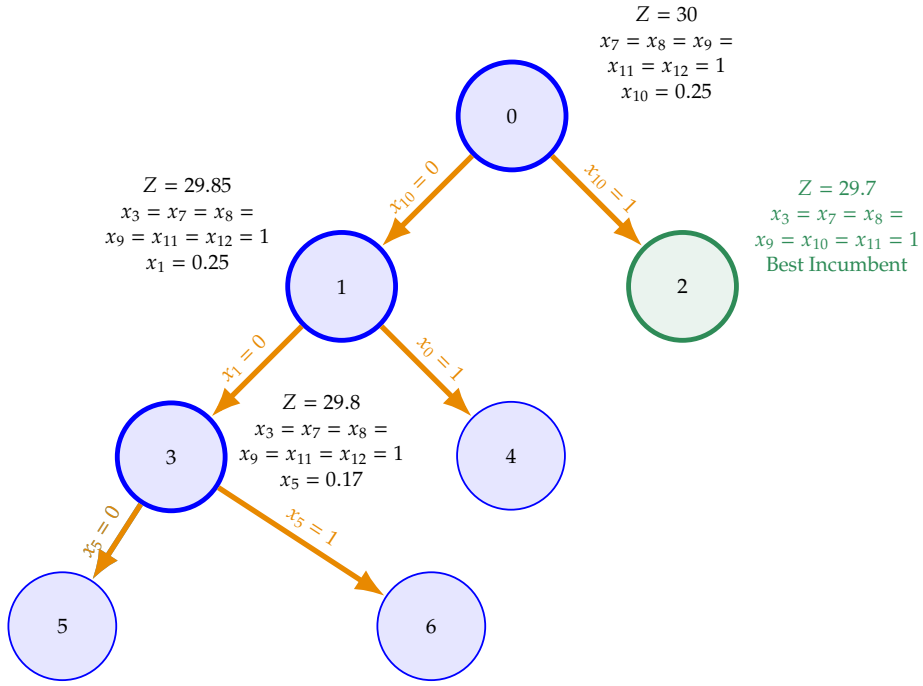


(a) Tree structure.

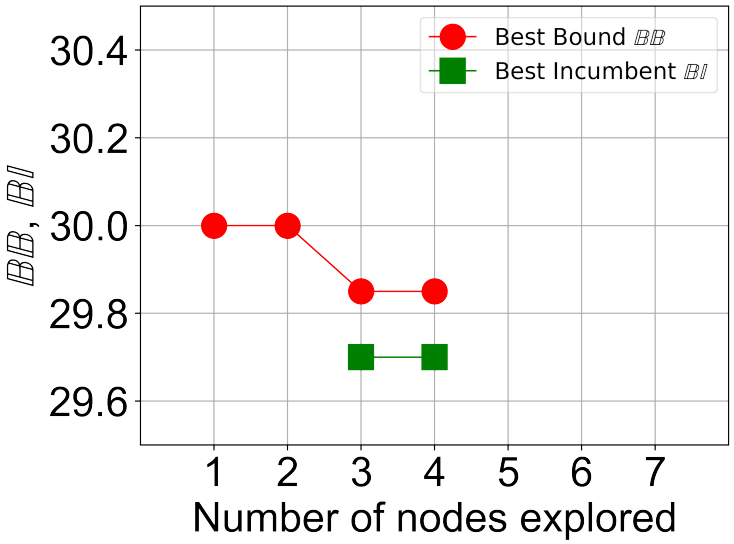


(b) Evolution of $\mathbb{BB}$ and $\mathbb{BI}$.

Figure 7.11: BB decision tree for the build your own pizza problem of Example 7.1: three nodes explored.

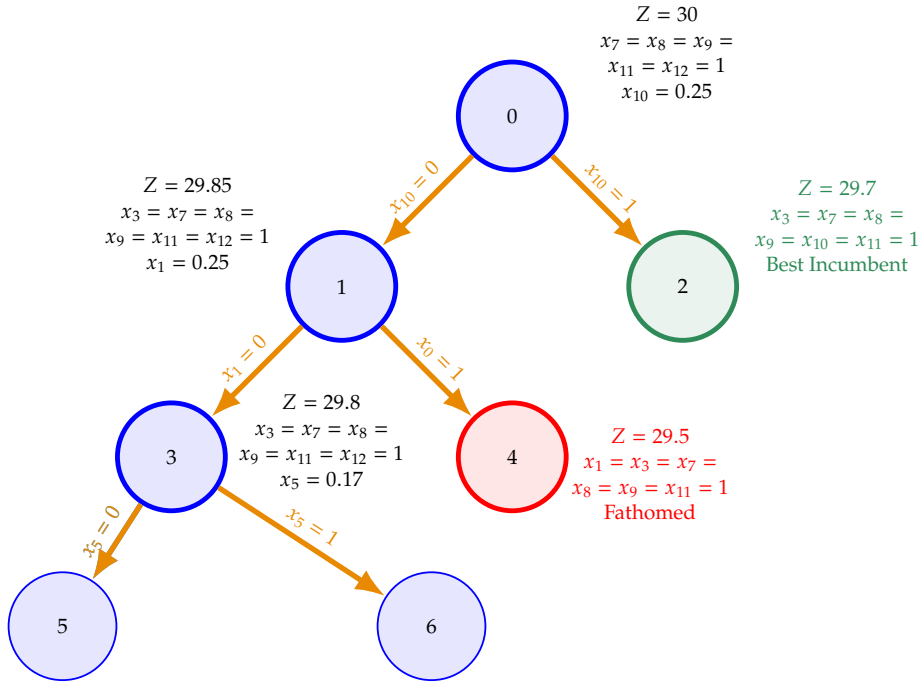


(a) Tree structure.

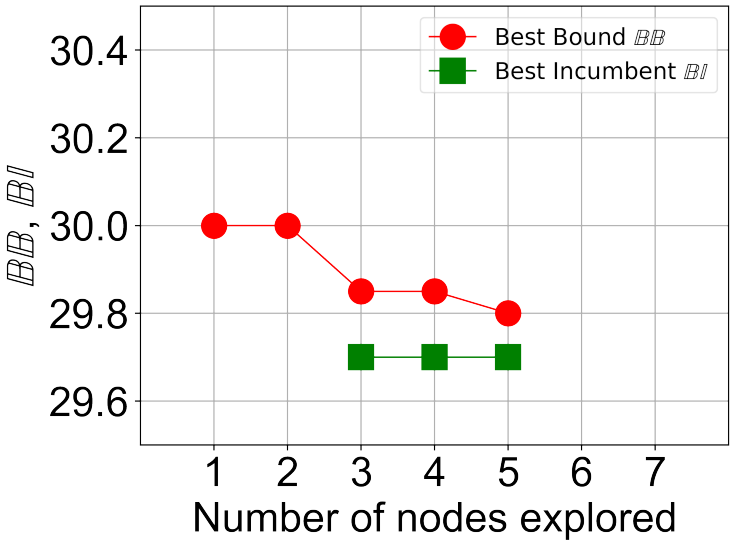


(b) Evolution of $\mathbb{BB}$ and $\mathbb{BI}$.

Figure 7.12: BB decision tree for the build your own pizza problem of Example 7.1: four nodes explored.

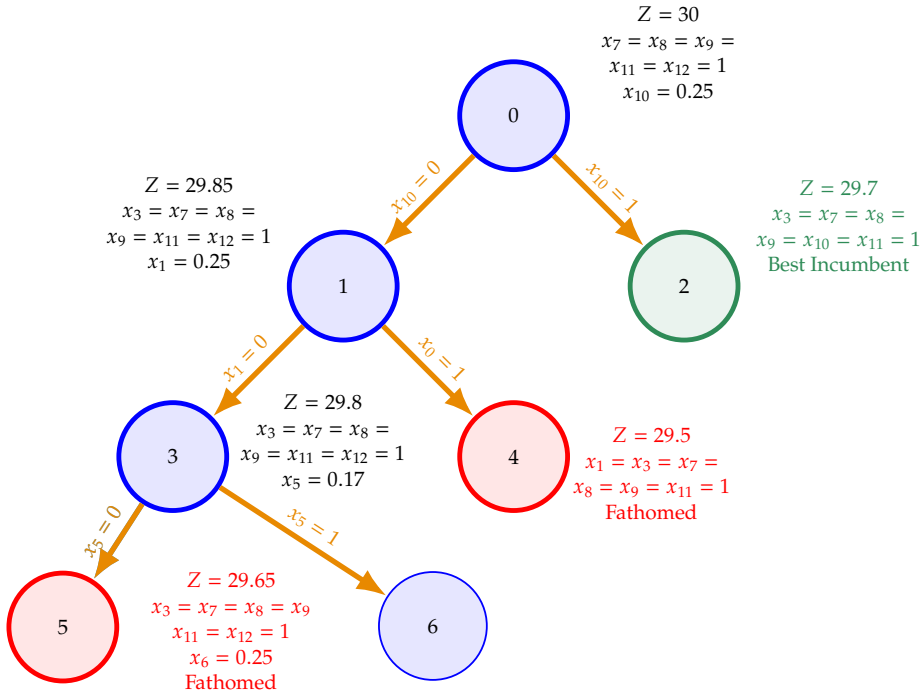


(a) Tree structure.

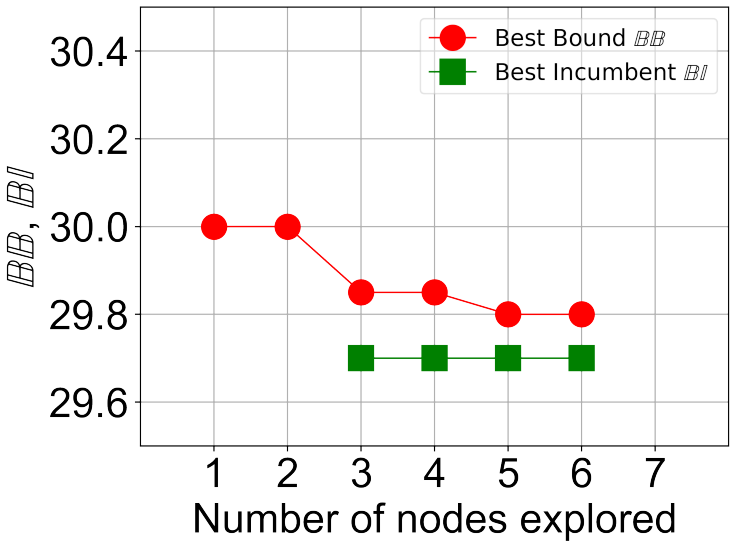


(b) Evolution of $\mathbb{BB}$ and $\mathbb{BI}$.

Figure 7.13: BB decision tree for the build your own pizza problem of Example 7.1: five nodes explored.

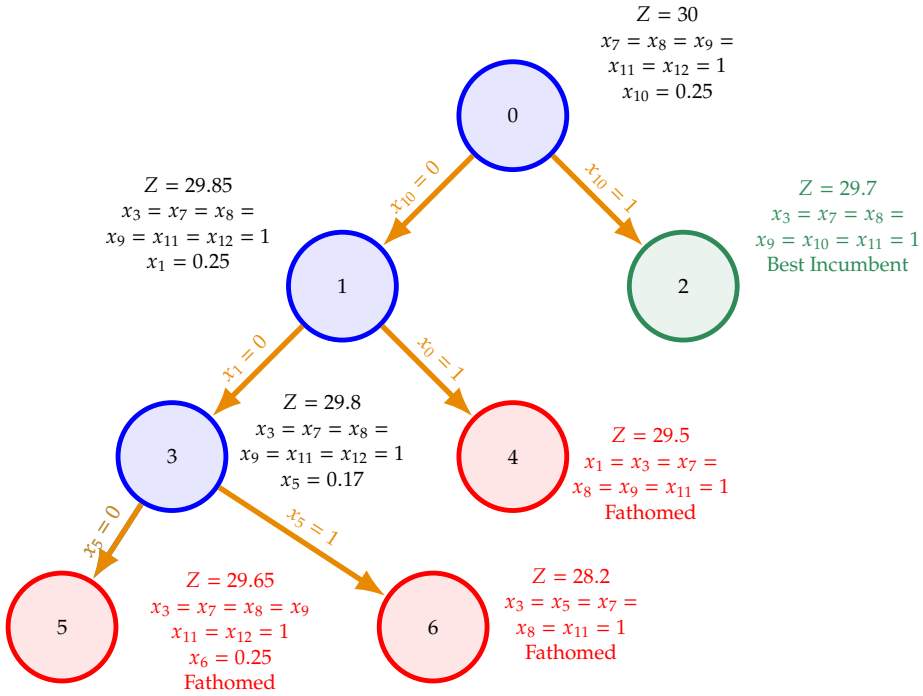


(a) Tree structure.

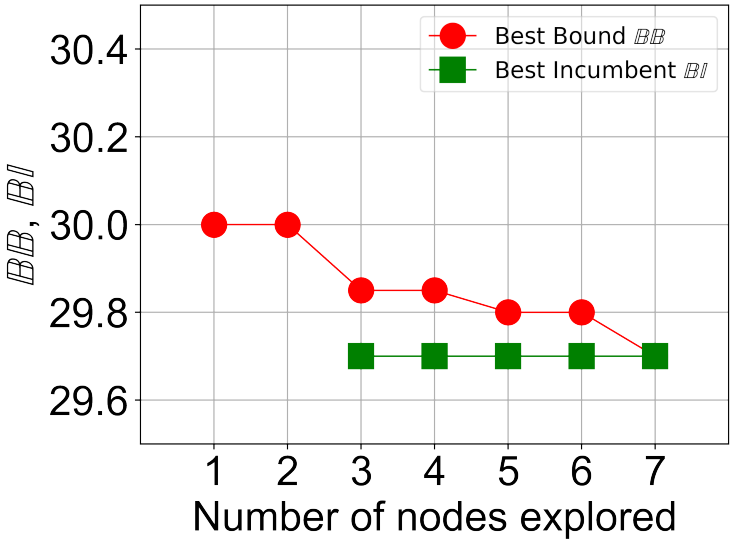


(b) Evolution of $\mathbb{BB}$ and $\mathbb{BI}$.

Figure 7.14: BB decision tree for the build your own pizza problem of Example 7.1: six nodes explored.



(a) Tree structure.



(b) Evolution of $\mathbb{BB}$ and $\mathbb{BI}$.

Figure 7.15: BB decision tree for the build your own pizza problem of Example 7.1: seven nodes explored.

Branch & Cut (BC) 8

The first cut is the deepest.

Cat Stevens

8.1 Motivation for BC . . . 133
8.2 Examples of cutting planes . . . 136
8.2.1 Gomory fractional cuts 136
8.2.2 Cover inequalities . . . 141
8.2.3 Zero-half cuts . . . 145
8.2.4 List of other cutting planes . . . 145
8.3 Combining BB and cutting planes for an efficient BC . . . 146

8.1 Motivation for BC

In Chapter 6, we explored how the solution of an LP resides at one of its corner points and how the simplex method efficiently navigates these points to find the optimal one. In Chapter 7, we extended this idea, emphasizing that when a model includes at least one integer decision variable, the method must be integrated into a decision tree. This tree must account for the correct treatment of each decision variable type, ensuring both mathematical accuracy and practical relevance. While delving into BB basics, we briefly introduced linear relaxation, highlighting its necessity for algorithm application but acknowledging its tendency to explore impractical regions of the solution space.

To this avail, **general BB solvers are equipped with automated ways of analyzing the MILP being solved and adding constraints that cut off parts of the feasible region that a linear relaxation would explore but are recognized not to lead to any integer solution (without affecting the optimal integer solution). Because additional constraints can be interpreted as cuts along the feasible region, this extension of BB is named Branch & Cut (BC)**.

We back up this intuition with an explicative example. Let us consider the following IP:

$$\max \quad x_1 + x_2 \tag{8.1}$$

s.t.:

$$x_1 \leq 2 \tag{8.2}$$
$$x_1 + 2x_2 \leq 4 \tag{8.3}$$
$$-x_1 + 2x_2 \leq 2 \tag{8.4}$$
$$x_1, x_2 \in \mathbb{N}_0 \tag{8.5}$$

We can visualize the feasible region and the integer (x_1, x_2) pairs inside it in Figure 8.1.

Considering our earlier discussion, astute readers might assert that the portion of the feasible region above $x_2 = 2$ exclusively pertains to the linear relaxation of the original IP. No integer (x_1, x_2) pair exists within it for $x_2 > 2$. A more refined version of Figure 8.1 is proposed in Figure 8.2,

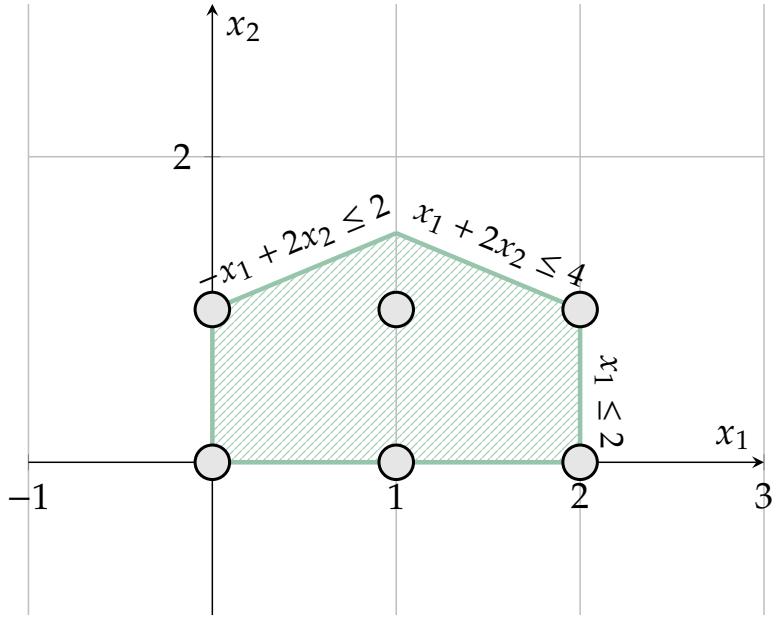


Figure 8.1: Example of integer feasible points and LP feasible region.

where the red region is “cut" from the feasible region. Removing this redundant space makes our BB solver explore a more compact solution space. This intuitive enhancement precisely characterizes what BC contributes to the BB routine detailed in Chapter 7.

The primary aim of BC is to eliminate all the red regions (as highlighted in Figure 8.2), streamlining the exploration of the BB decision tree by avoiding undesirable areas. In practical terms, to achieve this, one straightforward method is computing the **convex hull** of the set of feasible integer solutions within the original set of functional constraints. For those unfamiliar with the concept of convex hull, envision stretching an elastic band to enclose the entire green region in Figure 8.1. The gray feasible integer points act as pins, and upon releasing the band, the resulting shape, bounded by the pins, forms the green region in Figure 8.2. A more formal definition of convex hull is **the smallest convex set that contains a given shape in an n-dimensional space**.

Successfully computing the relevant convex hull for an MILP allows us to define it as an **integer polytope**. A polytope is essentially an extension of a polygon into an n-dimensional space. In this context, each side of the polytope is termed a **facet**, with the term “side" applicable specifically to 2-dimensional spaces. In a 3-dimensional space, a facet becomes a surface, and in general, a facet extends to an (n-1)-dimensional set of points that are part of the convex hull in an n-dimensional space.

While the elastic band and pins analogy offered an intuitive approach to grasp the concept of convex hull (and thus, an integer polytope) for an MILP, the algorithmic and mathematically rigorous computation of all necessary facets becomes a formidable task for larger problems. Ensuring that all facets are determined guarantees integer corner points in the resulting MILP, thus optimizing the BB process. However, one could argue that any cut diminishing the red region in Figure 8.2 is beneficial for enhancing exploration efficiency. Fortunately, adding "good" cuts for this purpose is a less challenging task, and modern BB solvers incorporate an extensive set of cuts. These cuts are scrutinized and added to eliminate

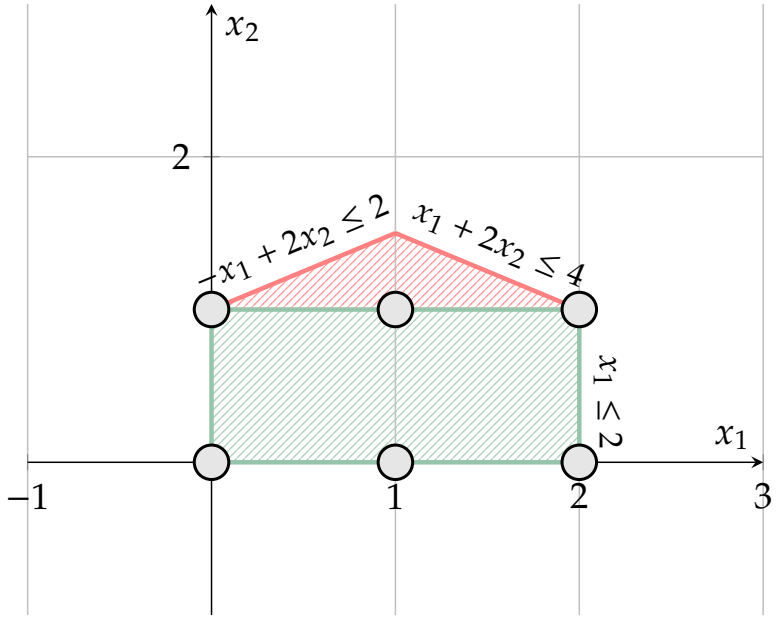


Figure 8.2: Example of integer feasible points and LP feasible region reduced to an integer polytope.

the superfluous portion of the linear solution space without impacting the integer one. In practice, all state-of-the-art BB solvers effectively function as BC solvers. We provide a clarification on the term “cut" in the **A note on the term “cut"** box.

A note on the term “cut"

Given our definition of convex hull and integer polytope, it follows that facets in a 2-dimensional space are lines. Hence, every constraint added to tighten the feasible solution space is a cut in the strict sense (i.e., a 1-dimensional line). In a 3-dimensional space, cuts are cutting planes (2-dimensional surfaces), **with the term cutting plane generally applied as an umbrella term for every constraint tightening the feasible region in higher dimensions**.

As outlined in Chapter 7, when managing an MILP, the BB tree structure employs only fractional original variables that should be integers for separation. Augmented variables, on the other hand, can assume feasible fractional values to secure the optimal corner point and its associated optimal basic solution. For the sake of clarity, let us consider the following IP:

$$\max \quad x_1 + x_2 \tag{8.6}$$

s.t.:

$$x_1 \leq \frac{5}{2} \tag{8.7}$$

$$x_2 \leq \frac{7}{2} \tag{8.8}$$

$$x_1, x_2 \in \mathbb{N}_0 \tag{8.9}$$

As x_1 and x_2 are not simultaneously present in any constraint and given that (8.6) seeks to maximize their sum, we can assign each the maximum integer value allowed by Equation 8.7 and (8.8), respectively. This yields the optimal solution, $(x_1, x_2) = (2, 3)$, with an optimal objective of $Z = 5$, obviating the need for any BB algorithm here. In an augmented form, converting the model reveals $x_3 = \frac{1}{2}$ and $x_4 = \frac{1}{2}$, where x_3 and x_4 serve as the slack variables for (8.7) and (8.8). While these variables are fractional due to the nature of the right-hand sides, it does not impact the integer nature of the solution concerning the original decision variables. We provide an additional consideration about this example in the **A note on model (8.6)-(8.9)** box.

A note on model (8.6)-(8.9)

Some readers might question that, given that $x_1, x_2 \in \mathbb{N}_0$, then x_1 could never assume the $\frac{5}{2}$ value, and hence constraint (8.7) could be tightened as $x_1 \leq 2$, being 2 the largest integer value that x_1 can take. The same would apply to constraint (8.8) which can be tightened as $x_2 \leq 3$. This is of course correct and displays a good understanding of the problem and its mathematical features by the modeler. **We kept the original fractional right-hand sides to substantiate our claim that in an IP the optimal solution might feature fractional augmented variables if some coefficients are fractional**.

Conversely, if an MILP includes only integer coefficients, it can be proven that all decision variables, both original and augmented, must be integers in the optimal corner point. For a more in-depth exploration of this concept, readers can refer to Hillier and Lieberman, 2015 or Carter et al., 2018. Many cutting plane techniques hinge on this assumption, grounded in the observation that numerous parameters from real-life operations (subsequently mapped into these coefficients) inherently possess integer characteristics. Consequently, in the set of example cutting planes outlined in Section 8.2, we will presume that every coefficient of the original MILP, unless explicitly stated otherwise, is an integer.

8.2 Examples of cutting planes

8.2.1 Gomory fractional cuts

1: Named after Ralph E.Gomory, applied mathematician and executive. See this Wikipedia page.

Gomory[1] cuts rank among the most renowned types of cutting planes. These cuts leverage the property that, **within any integer solution, fractional values may persist in certain constraints of the tableau. However, when these fractional values cancel each other out, the ultimate outcome becomes integer-valued**.

Leveraging this insight, the optimal tableau of the linear relaxation of the original MILP can be analyzed. When confronted with at least one fractional decision variable, a Gomory cut is introduced. This cut excludes the current fractional solution without eliminating any integer-valued solutions from the revised solution space. The problem is then solved iteratively until an integer solution is attained. Readers may ponder whether this iterative process could entirely replace BB. **In theory, continuously solving LPs while incrementally adding a new**

constraint (the latest Gomory cut) might eventually converge to the optimal integer solution without resorting to a decision tree. While this is true, particularly for large instances, the number of Gomory cutting planes required for convergence can escalate, diminishing the process's efficiency compared to BB. As previously suggested, modern solvers enhance the efficiency of a pure BB algorithm by rapidly generating a subset (rather than all) of the cutting planes (see Section 8.3).

Before delving into the formal definition of a Gomory cutting plane, let us underscore its utility with an illustrative example. Considering the IP:

$$\max \quad 8x_1 + 5x_2 \tag{8.10}$$

s.t.:

$$x_1 + x_2 \leq 6 \tag{8.11}$$
$$9x_1 + 5x_2 \leq 45 \tag{8.12}$$
$$x_1, x_2 \in \mathbb{N}_0 \tag{8.13}$$

The optimal tableau of its LP relaxation is shown in Table 8.1, where x_3 and x_4 are, respectively, the slack variables of (8.11) and (8.12).

	Z	x_1	x_2	x_3	x_4	R.H.S.
	1	0	0	1.25	0.75	41.25
(x_2)	0	0	1	2.25	-0.25	2.25
(x_1)	0	1	0	-1.25	0.25	3.75

Table 8.1: Optimal tableau of the LP relaxation of (8.10)-(8.13) before the addition of a Gomory cut.

Because of the fractional nature of the optimal basic solution $(x_1, x_2) = (3.75, 2.25)$, the current LP needs further enrichment to lead us to an integer-valued solution. This is where a Gomory fractional cutting plane comes in handy. Let us consider the second constraint row in Table 8.1, i.e.,

$$1x_1 - 1\frac{1}{4}x_3 + \frac{1}{4}x_4 = 3\frac{3}{4} \tag{8.14}$$

where we wrote each coefficient highlighting the integer and fractional part. We can now separate those coefficients into an integer and **positive fractional part**, e.g., $3\frac{3}{4} = 3 + \frac{3}{4}$, but $-1\frac{1}{4} = -2 + \frac{3}{4}$, and rewrite (8.14) by moving all the integer parts to the right-hand side together with the original right-hand side. We hence obtain

$$\frac{3}{4}x_3 + \frac{1}{4}x_4 = (-x_1 + 2x_3 + 3) + \frac{3}{4} \tag{8.15}$$

Analyzing (8.15), the following takeaways emerge:

- fractional coefficients are split to feature an integer value and a positive fractional value, and terms containing such positive fractional values are isolated on the left-hand side. **As such, the left-hand side of (8.15) or of any equivalent constraint should be non-negative**;
- in any integer solution, given that assumptions on the integrality of every coefficient of the original MILP, then **the term inside the brackets on the right-hand side (i.e., $-x_1 + 2x_3 + 3$) should be integer**.

Combining the two insights, we can write that in every integer solution

$$\frac{3}{4}x_3 + \frac{1}{4}x_4 = \left\{\frac{3}{4}, 1\frac{3}{4}, 2\frac{3}{4}, \cdots\right\} \tag{8.16}$$

meaning in (8.16) that the left-hand side can take one of the values within the brackets because the right-hand side is the summation of a non-negative integer number $(-x_1 + 2x_3 + 3)$ and $\frac{3}{4}$. Finally, we can rewrite (8.16) as

$$\frac{3}{4}x_3 + \frac{1}{4}x_4 \geq \frac{3}{4} \tag{8.17}$$

which is the Gomory cutting plane we were looking for. **Note that in the current optimal solution both x_3 and x_4 are non-basic. Hence, this cutting plane makes the current optimal solution of the LP relaxation infeasible as $\frac{3}{4} \times 0 + \frac{1}{4} \times 0 \geq \frac{3}{4}$ is not satisfied**.

We can add this inequality to the optimal tableau of Table 8.1 by first rewriting it as $-\frac{3}{4}x_3 - \frac{1}{4}x_4 \leq -\frac{3}{4}$ and then putting it in augmented form (adding the additional slack variable x_5) as $-\frac{3}{4}x_3 - \frac{1}{4}x_4 + x_5 = -\frac{3}{4}$. The new tableau is depicted in Table 8.2.

Table 8.2: Optimal tableau of the LP relaxation of (8.10)-(8.13) after the addition of a Gomory cut. Because x_5 features a negative value, the addition of the cut makes the current fractional solution infeasible.

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	0	0	1.25	0.75	0	41.25
(x_2)	0	0	1	2.25	-0.25	0	2.25
(x_1)	0	1	0	-1.25	0.25	0	3.75
(x_5)	0	0	0	-0.75	-0.25	1	-0.75

As both x_3 and x_4 are non-basic, introducing the cut results in $x_5 = -\frac{3}{4}$, **emphasizing that this addition renders the current solution infeasible[2]: a correct outcome as the cutting plane is designed to eliminate such fractional solutions**. To determine the "revised" optimal solution after incorporating the Gomory cut, two options exist. The first involves resolving the problem entirely, incorporating the additional constraint from the outset. The second approach leverages the fact that we already know the optimal solution to a very similar problem ("just" the additional Gomory cut differentiates the two), and hence we could tamper with the current infeasible tableau and with some row operations to find the new optimal solution. This second approach relies on a variant of the method called the **dual simplex method**.

While we leave out the full description of such an algorithm (we refer interested readers to Hillier and Lieberman, 2015), we share here the

2: Here, we are employing a specific aspect of sensitivity analysis. In this scenario, we are appending a constraint in tabular format to the optimal solution of an optimization problem to confirm its current validity. We are not delving into the theoretical intricacies, such as the effects on the optimal solution when altering coefficients of basic or non-basic variables, but directing curious readers to Hillier and Lieberman (2015), which dedicates a comprehensive chapter to sensitivity analysis.

main features of the method. **One of the main applications of the dual simplex method entails dealing with the addition to an optimal tableau of a functional constraint in augmented form that makes the current optimal solution infeasible**. As discussed in Chapter 6, adding a constraint either retains the feasibility of the current optimal solution or renders it infeasible. In the latter case, for a max problem, it signifies that the new optimal solution will be lower than the original, if an optimal solution to the revised problem exists.

In the simplex method, we look for the entering basic variable that has the potential to increase the objective the most by selecting the non-basic variable with the most negative coefficient in the objective row. Then, we determine the exiting basic variable via the minimum ratio test. Conversely, here the reverse process is followed. The exiting basic variable is identified first as the one with the negative value (x_5 in our example), i.e., the one highlighting infeasibility. **The entering basic variable is then selected as the one that minimally reduces the objective value: in our case, either x_3 or x_4**. Opting for x_3 involves transforming the 1.25 coefficient in the objective row to 0. This is achieved by replacing the objective row with a linear combination of itself plus $\frac{1.25}{-0.75}$ times the (x_5) row, resulting in a revised objective value of 40. Choosing x_4 would require a linear combination involving $\frac{0.75}{-0.25}$ times the (x_5) row, yielding a revised objective value of 39. The preference for x_3 is evident. Notably, if x_4 were chosen, negative coefficients for the non-basic variables in the objective row would highlight the necessity of additional iterations, as explained in Chapter 6. Conversely, selecting x_3 as the entering basic variable and performing all row operations leads to an optimal tableau (shown in Table 8.3) with all coefficients in the objective row being positive, confirming the optimality of the new solution.

	Z	x_1	x_2	x_3	x_4	x_5	R.H.S.
	1	0	0	0	-0.33	-1.67	40
(x_2)	0	0	1	0	-1	3	0
(x_1)	0	1	0	-0	0.67	-1.67	5
(x_3)	0	0	0	1	0.33	-1.33	1

Table 8.3: Optimal tableau of the LP relaxation of (8.10)-(8.13) after the addition of a Gomory cut and after the dual simplex method has been applied to restore the feasibility of the solution.

After going through an application example, we formalize the expression of a Gomory cutting plane as follows. Let us consider the optimal tableau of the LP relaxation of an MILP, and let us assume x_r to be a fractional basic variable, with b indicating its associated row in the tableau. Let us also define $\mathfrak{X}_{NB}$ the set of non-basic variables of the optimal solution and index them with j. Hence, with A_{bj} we represent the coefficient in position (b, j) of the tableau. Row r can be expressed as

$$x_r + \sum_{j \in \mathfrak{X}_{NB}} A_{bj} x_j = b_r \tag{8.18}$$

where $\sum_{j \in \mathfrak{X}_{NB}} A_{bj} x_j = 0$ because all x_js are non-basic and $x_r = b_r$ is fractional as per our assumption. The Gomory cutting plane can be defined as

$$\sum_{j \in \mathfrak{X}_{NB}} (A_{bj} - \lfloor A_{bj} \rfloor) x_j \geq (b_r - \lfloor b_r \rfloor) \tag{8.19}$$

Let us verify that (8.19) eliminates the original fractional solution while preserving all integer solutions: a crucial criterion for any cutting plane. **Concerning the first requirement, we should remember that all x_js are 0 as they are the non-basic variables of the fractional solution**. In addition, because we assumed x_r to be fractional, we have that $(b_r - \lfloor b_r \rfloor) > 0$. Hence (8.19) applied to the original LP relaxation becomes $0 \geq (b_r - \lfloor b_r \rfloor)$ **which is not satisfied and hence renders the current fractional solution infeasible**. We now need to verify that, on the other hand, such a cutting plane is harmless for integer solutions. Here, the process is slightly more complicated. First, let us realize that for any LP solution it holds that

$$x_r + \sum_{j \in \mathfrak{X}_{NB}} \lfloor A_{bj} \rfloor x_j \leq x_r + \sum_j A_{bj} x_j = b_r \tag{8.20}$$

which, for the particular case of an integer solution, becomes

$$x_r + \sum_{j \in \mathfrak{X}_{NB}} \lfloor A_{bj} \rfloor x_j = \lfloor b_r \rfloor \tag{8.21}$$

because for an integer solution $b_r = \lfloor b_r \rfloor$. If we now subtract (8.21) from (8.18) we obtain $\sum_j (A_{bj} - \lfloor A_{bj} \rfloor) x_j \geq (b_r - \lfloor b_r \rfloor)$, which is the definition of the Gomory cutting plane of (8.19).

Going back to our example, let us leverage the fact that it is a two-dimensional example and analyze the graphical interpretation of the Gomory cut we added. Because the feasible region is defined in the (x_1, x_2) space, we need to rewrite the Gomory cut $\frac{3}{4}x_3 + \frac{1}{4}x_4 \geq \frac{3}{4}$ as a function of x_1 and x_2. To this avail, we can use (8.11) and (8.12) in augmented form, respectively, $x_1 + x_2 + x_3 = 6$ and $9x_1 + 5x_2 + x_4 = 45$ and plug them in the Gomory cut: $\frac{3}{4}(6 - x_1 - x_2) + \frac{1}{4}(45 - 9x_1 - 5x_2) \rightarrow 3x_1 + 2x_2 \leq 15$.

We can now plot the original feasible region and the feasible region "trimmed" by the Gomory cut. The first situation is depicted in Figure 8.3. In green it is reported the feasible region, in gray the integer points inside such a region, and in dark orange the optimal solution of the LP: $(x_1, x_2) = (3.75, 2.25)$. We can notice how the bottom right corner of the feasible region does not belong to the integer polytope we defined before.

In Figure 8.4, the added value of the Gomory cutting plane is highlighted. In this specific case, the cut passes through integer points $(x_1, x_2) = (3, 3)$ and $(x_1, x_2) = (5, 0)$, hence eliminating the portion of the original feasible region highlighted in red. **By doing so, the "trimmed" feasible region is an integer polytope (i.e., the convex hull of all integer points) as the cutting plane coincided with the missing facet running from $(x_1, x_2) = (3, 3)$ to $(x_1, x_2) = (5, 0)$**. Thanks to this improvement, now the solution to the revised LP is integer: $(x_1, x_2) = (5, 0)$ as highlighted by the dark orange circle.

A final reminder: the Gomory cutting plane method's convergence can generally be slow. The process is conceptually straightforward: solve the relaxed LP of an MILP, halt if the solution is integer, or else add

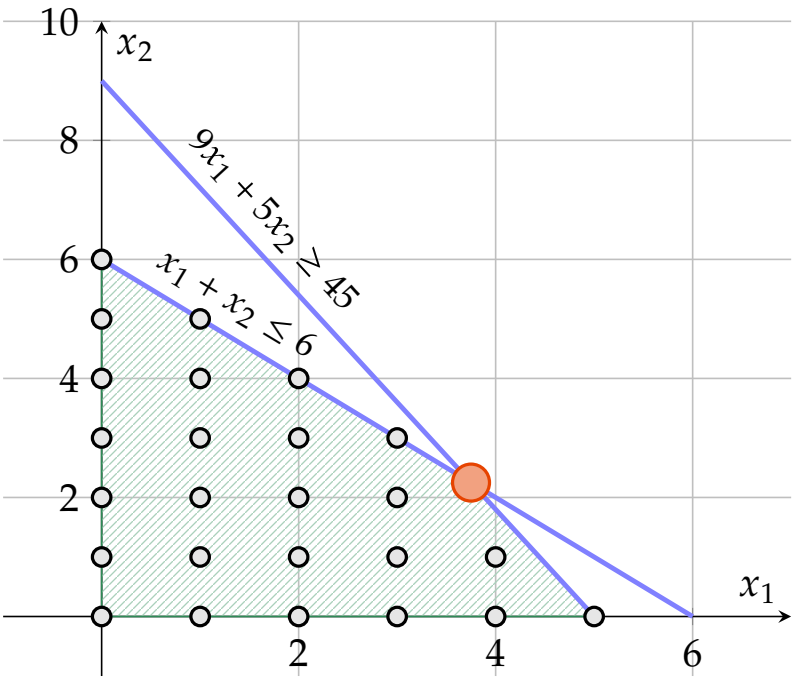


Figure 8.3: Feasible region, integer points (in gray), and optimal solution (in dark orange) of the LP relaxation of (8.10)-(8.13) for the Gomory cutting plane example (before the addition of the cutting plane).

a Gomory cut in the form of (8.19) to the tableau based on a selected fractional basic variable. Subsequently, run one iteration of the dual simplex method to compute the new (lower) objective while restoring feasibility. **In our example, a single Gomory cutting plane sufficed, but for many large-scale problems, the substantial number of required planes for convergence makes this approach unsuitable as a stand-alone process.**

We want to tease interested readers with the following IP:

$$\max \quad x_2 \tag{8.22}$$

s.t.:

$$3x_1 + 2x_2 \leq 6 \tag{8.23}$$
$$-3x_1 + 2x_2 \leq 0 \tag{8.24}$$
$$x_1, x_2 \in \mathbb{N}_0 \tag{8.25}$$

Despite being similar to the previous example in size, in this case one single Gomory cut will not suffice to converge to the optimal integer solution.

8.2.2 Cover inequalities

Gomory cutting planes, detailed in Section 8.2.1, primarily target fractional integer decision variables. In contrast, cover inequalities are cutting planes tailored specifically for binary decision variables. Let us consider a generic constraint

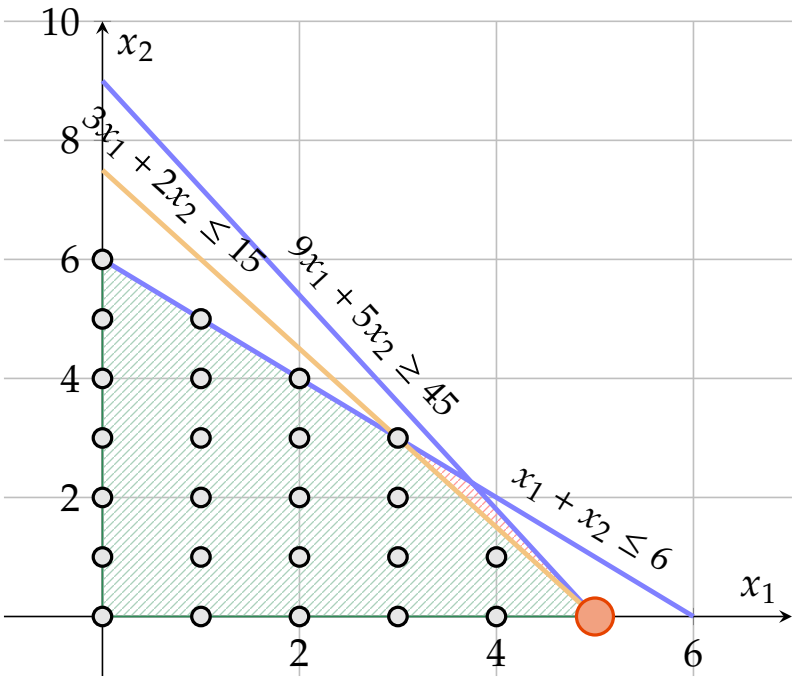


Figure 8.4: Feasible region, integer points (in gray), and optimal solution (in dark orange) of the LP relaxation of (8.10)-(8.13) for the Gomory cutting plane example (after the addition of the cutting plane, which is depicted in orange). Note that the Gomory cutting plane reduces the feasible region by eliminating the small portion highlighted in red and without cutting off any integer solution. Additionally, the cutting plane passes through integer points $(x_1, x_2) = (3, 3)$ and $(x_1, x_2) = (5, 0)$ and hence defines the missing facet of the integer polytope.

$$\sum_i C_i x_i \leq b \tag{8.26}$$

where every $x_i \in \{0, 1\}$. Because of the binary nature of each decision variable, the associated coefficient C_i is activated and contributes positively to the left-hand side if $x_i = 1$. If we identify a subset $\mathcal{S}$ of decision variables in (8.26) where

$$\sum_{i \in \mathcal{S}} C_i > b \tag{8.27}$$

we can infer that those binary decision variables cannot all simultaneously take a value of 1, as doing so would violate (8.26). **In principle, if they were all to take a unitary value, the left-hand side would "cover" the right-hand side coefficient, hence the origin of the name for this cutting plane technique**. Let us consider the following example:

$$4x_1 + 3x_2 + 6x_3 \leq 7 \tag{8.28}$$

While x_1 and x_2 can be simultaneously unitary in (8.28), index sets $\{1, 3\} \rightarrow 4x_1 + 6x_3 = 4 \times 1 + 6 \times 1 = 10 > 7$ and $\{2, 3\} \rightarrow 3x_2 + 6x_3 = 3 \times 1 + 6 \times 1 = 9 > 7$ define combinations of decision variables that cannot be simultaneously unitary because they would violate (8.28). We can hence define the cover inequalities as

$$x_1 + x_3 \leq 1 \tag{8.29}$$
$$x_2 + x_3 \leq 1 \tag{8.30}$$

that can be added to the original BP model to strengthen the formulation. The general definition of a cover constraint is

$$\sum_{i \in \mathcal{S}} x_i \leq |\mathcal{S}| - 1 \tag{8.31}$$

This implies that, given a cover set $\mathcal{S}$, at most $|\mathcal{S}| - 1$ binary variables (or possibly fewer) can be activated to satisfy the constraint from which the set was derived. Furthermore, there are cases where removing an element from the cover still maintains its validity as a "smaller" cover. For instance, consider the following constraint

$$4x_1 + 5x_2 + 3x_3 + 6x_4 + 7x_5 + 2x_6 \leq 18 \tag{8.32}$$

and let us assume we are given the following cover set $\mathcal{S} = \{1, 2, 3, 4, 5, 6\}$. The set satisfies the condition for being a cover set, as $4 \times 1 + 5 \times 1 + 3 \times 1 + 6 \times 1 + 7 \times 1 + 2 \times 1 = 27 > 18$. Note that, according to (8.31), we could select no more than $|\mathcal{S}| - 1 = 5$ variables. However, in (8.32), we observe that there is no subset of 5 decision variables that fulfills the constraint. Setting the decision variable with the largest coefficient (x_5) to 0 and activating the other 5 variables results in a left-hand side sum of 20, which is larger than 18. Thus, we can argue that the new set $\mathcal{S} = \{1, 2, 3, 4, 6\}$ also forms a cover set. This example illustrates the concept of a **minimal cover**: in a minimal cover set $\mathcal{S}$, if we remove any element from it, the associated constraint is now satisfied. For example, considering again (8.32), a minimal cover set is $\mathcal{S} = \{1, 2, 3, 5\}$ and the associated cover inequality (applying (8.31)) is

$$x_1 + x_2 + x_3 + x_5 \leq 3 \tag{8.33}$$

and we can verify that any combination of 3 out of the 4 decision variables being active (and all the others set to 0) results in a satisfied (8.32). For example, $x_1 = x_2 = x_3 = 1, x_5 = 0 \rightarrow 4 \times 1 + 5 \times 1 + 3 \times 1 = 12 \leq 18$, $x_1 = x_2 = x_5 = 1, x_3 = 0 \rightarrow 4 \times 1 + 5 \times 1 + 7 \times 1 = 16 \leq 18$, etc. This intuition paves the road for a formal definition of a minimal cover set $\mathcal{S}_{MC}$:

$$\sum_{j \in \mathcal{S}_{MC}} C_j - C_k < b \qquad \forall k \in \mathcal{S}_{MC} \tag{8.34}$$

which highlights that a minimum cover set $\mathcal{S}_{MC}$ ceases to be such a set as soon as one single element is removed from it.

Cover inequalities are typically applied to constraints in the form of (8.26), which are known as **knapsack constraints**[3]. On the left-hand side, we have potential combinations of items, each with its own weight C_i. We can select any combination as long as the total weight does not surpass the capacity b of our knapsack. Notwithstanding, they can also be applied to $\geq$ constraints by reshuffling them into an $\leq$ form and equality constraints. For an equality constraint, the trick is to "duplicate" it with a $\leq$ and $\geq$ version. We clarify this with an example. Let us consider the following constraint:

3: We will talk a lot about this type of constraints in Chapter 10.

$$4x_1 - 5x_2 + 6x_3 - 2x_4 = 1 \tag{8.35}$$

We can rewrite (8.35) as

$$4x_1 - 5x_2 + 6x_3 - 2x_4 \leq 1 \tag{8.36}$$
$$4x_1 - 5x_2 + 6x_3 - 2x_4 \geq 1 \tag{8.37}$$

where the only case where both (8.36) and (8.37) can be satisfied is when the two left-hand sides are both equal to 1, hence providing the same information as (8.35). To satisfy the requirements of a knapsack constraint, an inequality should be in the $\leq$ form and all coefficients of the decision variables should be positive. In our example, this is not yet the case. We can transform (8.37) into $-4x_1 + 5x_2 - 6x_3 + 2x_4 \leq -1$. Then, we replace all decision variables x_i characterized by a negative coefficient with the auxiliary term $1 - x_i^{'}$, where $x_i = 1 - x_i^{'}$. After making these adjustments and separating all terms involving decision variables on the left-hand side and constants on the right-hand side, we arrive at the revised set of inequalities

$$4x_1 + 5x_2^{'} + 6x_3 + 2x_4^{'} \leq 8 \tag{8.38}$$
$$4x_1^{'} + 5x_2 + 6x_3^{'} + 2x_4 \leq 9 \tag{8.39}$$

where (8.38) is the revised version of 8.36 and (8.39) of (8.37). (8.38)-(8.39) are now knapsack constraints and can be used separately to add cover inequalities. Focusing on (8.38), a couple of minimal cover sets are $\{1, 2^{'}\}$ and $\{1, 3\}$. The associated cover inequalities (remember to convert back to the original variables if needed) are:

$$x_1 + x_2^{'} \leq 1 \rightarrow x_1 - x_2 \leq 0 \tag{8.40}$$
$$x_1 + x_3 \leq 1 \tag{8.41}$$

While the process of computing cover sets is straightforward, involving simple algebraic operations, large-scale optimization problems can lead to an overwhelming number of such sets. Even for a single knapsack constraint with numerous decision variables, generating all combinations of variables that would violate the constraint if all were active can be algorithmically challenging. Moreover, many cover inequalities may remain inactive unless they effectively narrow the feasible region to make an integer (binary, in this case) point a corner point. Therefore, instead of blindly adding numerous cuts, it is more advantageous to focus on incorporating “good" cover inequalities. For a comprehensive discussion on algorithms addressing this objective, readers are directed to Carter et al. (2018).

8.2.3 Zero-half cuts

Zero-half cuts are based on the intuition that, when the left-hand side of an inequality features integer coefficients and decision variables, then the right-hand side can be rounded down. The act of rounding down is the zero-half cut. As usual, we clarify this with an example. Let us start with the following two inequalities:

$$x_1 + 2x_2 + x_3 + x_4 \leq 8 \tag{8.42}$$
$$x_1 + 3x_3 + x_4 + 2x_5 \leq 5 \tag{8.43}$$

We can sum them and obtain

$$2x_1 + 2x_2 + 4x_3 + 2x_4 + 2x_5 \leq 13 \tag{8.44}$$

which can be divided by 2 becoming:

$$x_1 + x_2 + 2x_3 + x_4 + x_5 \leq \frac{13}{2} \tag{8.45}$$

Note that in (8.44) all the coefficients of the decision variables are multiple of 2, hence they retain their integer nature in (8.45). Because the left-hand side of (8.45) cannot be fractional, we can round down the right-hand side $\lfloor \frac{13}{2} \rfloor = 6$ and obtain the zero-half cut:

$$x_1 + x_2 + 2x_3 + x_4 + x_5 \leq 6 \tag{8.46}$$

As a side note, in the final consideration of Section 8.1 related to model (8.6)-(8.9), readers might recognize the suggestion to translate $x_1 \leq \frac{5}{2}$ into $x_1 \leq 2$ and $x_2 \leq \frac{7}{2}$ into $x_1 \leq 3$ as two examples of zero-half cuts.

8.2.4 List of other cutting planes

In this section, we only covered a subset of cutting planes that BC leverages to improve the exploration of an BB decision tree. Both commercial solvers such as Gurobi (*Gurobi Optimization* 2023) and CPLEX (*IBM ILOG CPLEX Optimization Studio* 2023) and open-source ones adopt a wider variety of cutting planes. An extract of the output of an optimization model solved in Gurobi might look like this:

```
Cutting planes:
  Gomory: 278
  Cover: 411
  Implied bound: 932
  Clique: 19
  MIR: 1569
```

```
StrongCG: 557
Flow cover: 1068
GUB cover: 103
Zero half: 345
RLT: 87
```

Readers might recognize Gomory cutting planes, cover inequalities, and zero-half cuts being employed by the solver. **Additional (yet, the full list is even more exhaustive) cutting planes shown in the verbatim that solvers can leverage are implied bounds, clique cuts, Mixed Integer Rounding (MIR) cuts, Strong Chvatal–Gomory (SCG) cuts, flow cover, Generalized Upper Bound (GUB) covers, and Reformulation Linearization Technique (RLT) cuts**. We refer readers to Johnson et al., 2000 for an exhaustive analysis of cutting planes.

8.3 Combining BB and cutting planes for an efficient BC

In Section 8.2, we touched upon the computational complexity involved in identifying *all* cuts of a particular type for large-scale problems. Consequently, integrating cut generation into an BB algorithm may prolong the overall computational time until convergence compared to a conventional BB solver. **State-of-the-art solvers employ a trade-off between exploration (generating as many cuts as possible) and exploitation (limiting such generation to save computational time to explore more nodes of the decision tree)**. For every node of the BB tree, the associated LP is tightened with some, and not all, potential cuts. How to smartly select which cuts are the best to add is, to some extent, an optimization problem on its own, and every solver employs different techniques and rules to effectively tackle this choice. Parameters can also be set that turn on or off the addition of specific types of cuts (*Gurobi Optimizer Reference Manual: Cuts* 2023). These parameters can be leveraged by (experienced) modelers when realizing that the problem at hand might particularly benefit if the addition of specific cutting planes is enforced.

Part IV

Assignment Problems

Assignment and scheduling problems

9

> The key is not to prioritize what's on your schedule, but to schedule your priorities.
>
> Stephen Covey

9.1 Assignment problems . 149
9.1.1 The Hungarian algorithm 151
9.2 Preliminaries of scheduling 158
9.3 Single Machine Scheduling Problem (SMSP) . . 159
9.4 Parallel Machine Scheduling Problem (PMSP) . . 163
9.5 p-median problem . . . 166
9.6 Facility location problem 169

In OR, the concept of *assignment* plays a role of paramount importance. In essence, the goal of every OR model is to unravel the intricacies of a mathematical formulation and all the possible feasible combinations of the decision variables on our way to the optimal solution. Hence, finding the optimal solution entails *assigning* an optimal value to each decision variable.

Aside from this semantic interpretation, there exists in OR a category of problems called *assignment problems* that we discuss in this chapter. They revolve around the **optimal assignment of tasks to people (or any equivalent set of items to be assigned and recipients of such items) such that every person receives exactly one task and every task is assigned exactly once**. Assignment problems can be interpreted as a special case of transportation problems (see Carter et al., 2018) that we discuss in Chapter 12. We preferred to assign (no pun intended) them to a dedicated chapter and refer readers to Section 12.1.1 for a description of how assignment problems can be reinterpreted as transportation problems.

After covering the classic assignment problem and some variants, we move to an extension of the assignment problem that deal with one of the most important drivers of human's life, i.e., time. Such extension pertains to *scheduling problems*, which involve assigning tasks to processing units (workers, machines, etc.) while adhering to time constraints and precedence relationships.

9.1 Assignment problems

An assignment problem revolves around two main sets, namely a set of tasks $\mathcal{T}$ indexed by i and a set of recipients of those tasks $\mathcal{R}$ indexed by j. The nature and specific context of the two sets might severely change according to the specific application, but a requirement of classic assignment problems is that $|\mathcal{T}| = |\mathcal{R}|$: the number of tasks and recipients is the same. Additionally, we map with C_{ij} the cost of assigning task i to recipient j. Given the nature of the problem, a set of binary variables x_{ij} taking unitary value if task $i \in \mathcal{T}$ is assigned to recipient $j \in \mathcal{R}$ seems appropriate. **The goal of the assignment problem is to assign tasks to recipients in the most cost-effective way (hence, minimizing the cost)**.[1] We define all the needed notation for an assignment problem in Table 9.1.

We define the assignment problem as:

1: This is the case when each assignment entails a "negative" cost. Sometimes, assignment problems aim at maximizing the objective. This is the case, for example, when recipients $j \in \mathcal{R}$ assign a score to each task and the goal is to maximize the cumulative score.

Table 9.1: Notation for the assignment problem.

Sets and indices	
$\mathcal{T}$	set of tasks $i \in \mathcal{T}$
$\mathcal{R}$	set of recipients $j \in \mathcal{R}$
Parameters	
C_{ij}	cost of assigning task $i \in \mathcal{T}$ to recipient $j \in \mathcal{R}$
Variables	
$x_{ij} \in \{0,1\}$	unitary if task i is assigned to recipient j

$$\min \sum_{i \in \mathcal{T}} \sum_{j \in \mathcal{R}} C_{ij} x_{ij} \tag{9.1}$$

s.t.:

$$\sum_{j \in \mathcal{R}} x_{ij} = 1 \qquad \forall i \in \mathcal{I} \tag{9.2}$$

$$\sum_{i \in \mathcal{T}} x_{ij} = 1 \qquad \forall j \in \mathcal{R} \tag{9.3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in \mathcal{I}, j \in \mathcal{R} \tag{9.4}$$

where (9.1) aims at minimizing the overall assignment cost, constraint (9.3) ensures that every task is assigned exactly once, constraint (9.3) that every recipient gets assigned exactly one task, and (9.4) defines the binary nature of the decision variables.

The BP defined by (9.1)-(9.4) can be solved with the BB algorithm showcased in Chapter 7. Furthermore, all the coefficients in the models are integer and, hence, all the corner points of the problem are also integer-valued (recall Chapter 6). **We could relax the binary requirement on the decision variables $x_{ij} \in \{0,1\} \rightarrow x_{ij} \in [0,1]$ and allow every decision variable to be continuous so that just the simplex method can be used instead of the full BB algorithm.**

Notwithstanding, this simplification that allows us to leverage the algorithmic efficiency of the simplex method comes with a caveat. Recalling that $|\mathcal{T}| = |\mathcal{R}|$, our assignment problem has $2|\mathcal{T}|$ functional constraints. Because of that was discussed in Section 6.3.1, at every iteration of the simplex method we then require $2|\mathcal{T}|$ variables to be basic and obtain their values via row operations. This clashes with constraints (9.2)-(9.3) which allow exactly $|\mathcal{T}|$ decision variables to take a value that is greater than zero. Hence, solving an assignment problem with the simplex method, at every iteration we would get many basic decision variables with a value of 0. This situation entails having a **degenerate solution** and, while not a problem per se, generally affects negatively the algorithmic performance of the simplex algorithm. While we refrain from diving into the details of degeneracy, we refer readers to Hillier and Lieberman, 2015 or Carter et al., 2018 for more details.

Fortunately, there exists an ad-hoc algorithm known as the **Hungarian algorithm**[2], which exploits the special properties of the assignment problem to solve it efficiently.

2: The story behind the origin and actual invention of such an algorithm is quite interesting. We refer readers to this Wikipedia page.

9.1.1 The Hungarian algorithm

The Hungarian algorithm is based on an efficient manipulation of the $(|\mathcal{T}|, |\mathcal{T}|)$ cost matrix C that stores all the C_{ij} cost coefficients. **An important note to be made regards the fact that all costs are non-negative**. The manipulation leverages the insight that adding or removing a constant to a row or column of C does not affect the optimal solution. Recalling our objective

$$Z = \sum_{i \in \mathcal{T}} \sum_{j \in \mathcal{R}} C_{ij} x_{ij} \tag{9.5}$$

let us add a constant P to all the C_{rj} coefficients in row r. The objective becomes:

$$Z' = \sum_{j \in \mathcal{R}} (C_{rj} + P) x_{rj} \sum_{i \in \mathcal{T} \setminus \{r\}} \sum_{j \in \mathcal{R}} C_{ij} x_{ij} \tag{9.6}$$

(9.6) can be rewritten as

$$Z' = \sum_{i \in \mathcal{T}} \sum_{j \in \mathcal{R}} C_{ij} x_{ij} + P \sum_{j \in \mathcal{R}} x_{ij} = Z + P \sum_{j \in \mathcal{R}} x_{ij} \tag{9.7}$$

where the second term in (9.7) can be interpreted as a constant. The same result can be achieved if the constant P is added to a column.

We use this intuition to qualitatively explain the Hungarian algorithm as follows. **If, given a cost matrix C, a set of row and column operations can be devised that, by subtracting specific constants from the targeted rows and columns, yields a revised cost matrix with at least a single 0 in every row and column, then the location of those 0s identifies the optimal assignment of tasks to recipients**. The optimal objective value is not zero (it is in the "revised" model subject to the row and column operations), but can be easily retrieved by summing all the C_{ij} cost coefficients in the original cost matrix located in the positions where the 0s appear in the revised cost matrix. We showcase an introductory exercise in Example 9.1.

Example 9.1 *Three students have provided their own scores for three individual projects. The scores range from 1 to 10, with 1 meaning the student really likes the project and 10 meaning the student does not like the project at all. Such scores are stored in matrix C, where element (i, j) represents the score given by student j to project i[3]. C is reported in (9.8). The objective is to devise an assignment strategy that minimizes the total assignment cost, which is equivalent to maximizing the overall likelihood that students will be satisfied with their assigned projects.*

$$C = \begin{pmatrix} 7 & 4 & 7 \\ 5 & 8 & 2 \\ 3 & 10 & 9 \end{pmatrix} \tag{9.8}$$

We recognize that, given the unconventional, yet ad-hoc choice of assuming a lower score implies a higher preference, we can model this problem as an assignment problem with element C_{ij} in C mapping the cost of assigning project i to student j.

3: Note that, in Example 9.1, the definition of the cost matrix C is consistent with the original notation we introduced in (9.1)-(9.4) where rows define tasks (projects here) and the columns define recipients (students here). Oftentimes, we could be given cost matrices where element (i, j) defines the cost of assigning recipient i to task j, hence reversing the role of rows and columns. In such a case, we have two options. In the first case we transpose the cost matrix, so that C^T is consistent with the notation of (9.1)-(9.4). In the second case, we keep C but we need to be careful with the definition of x_{ij}. Because rows represent recipients and columns tasks, $x_{ij} = 1$ entails assigning recipient i to task j and not vice versa.

We recognize that the first project is liked the most by the second student (the minimum score in the first row is a 4 in location $(1,2)$). The second project is liked the most by the third student ($C_{2,3} = 2$) and the third by the first student ($C_{3,1} = 3$). Hence we can modify C by subtracting 4 from the first row (to "pretend" the second student assigned a zero cost to the first project without making any other cost in the row negative). Similarly, we subtract 2 and 3 from the second and third row respectively. This yields the following revised cost matrix (for the sake of simplicity, we do not rename it):

$$C = \begin{pmatrix} 3 & 0 & 3 \\ 3 & 6 & 0 \\ 0 & 7 & 6 \end{pmatrix} \tag{9.9}$$

In matrix (9.9) we have a very special case where every row and column feature *exactly* a single 0. We can then set $x_{1,2} = 1$, $x_{2,3} = 1$, and $x_{3,1} = 1$, hence assigning the first project to the second student, the second project to the third student, and the third project to the first student. This might not be surprising, given that the first student rated the third, the second student rated the first, and the third student rated the second project as their favorite. Hence, our solution makes everyone happy as no conflicts or ties are generated. **The assignment cost is $4 + 2 + 3 = 9$ (recall that we need to check the original C values to compute the objective value)**.

We can interpret the final assignment as a row and column reduction of C where we select a 0 value in position (i, j) and cross out the entire row i and column j. This means that we have decided to assign task i to recipient j. **Hence such a row and column do not play any role in later assignments as both the task i and recipient j have been selected already**. We keep crossing out rows and columns starting from a new 0 until the whole C is crossed out. In our case, we could do:

$$\begin{pmatrix} 3 & 0 & 3 \\ 3 & 6 & 0 \\ 0 & 7 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 3 & 0 & 3 \\ 3 & 6 & 0 \\ 0 & 7 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 3 & 0 & 3 \\ 3 & 6 & 0 \\ 0 & 7 & 6 \end{pmatrix} \tag{9.10}$$

The sequence of perpendicular lines in (9.10) reflects again our choice $x_{1,2} = 1$, $x_{2,3} = 1$, and $x_{3,1} = 1$. **Note that we chose to start with project 1, then project 2, and finally project 3, but any other sequence would have yielded the same result**.

In Example 9.1, we only needed row operations because each student had a different preferred project. This might (and usually will not) be the case. Let us consider the following cost matrix

$$C = \begin{pmatrix} 3 & 5 & 8 \\ 3 & 9 & 6 \\ 8 & 4 & 5 \end{pmatrix} \tag{9.11}$$

which, after applying row operations, becomes:

$$C = \begin{pmatrix} 0 & 2 & 5 \\ 0 & 6 & 3 \\ 4 & 0 & 1 \end{pmatrix} \tag{9.12}$$

We realize that there is not a single 0 in the third column, and hence reduce each element there by 1 unit as $\min\{5, 3, 1\} = 1$, obtaining:

$$C = \begin{pmatrix} 0 & 2 & 4 \\ 0 & 6 & 2 \\ 4 & 0 & 0 \end{pmatrix} \tag{9.13}$$

This revised cost matrix proves troublesome, as we cannot repeat the same process of row and column elimination we employed in Example 9.1. For example, assuming we start with the 0 in position $(1, 1)$, hence assuming task 1 is assigned to recipient 1, and then we move to the 0 in position $(3, 2)$, hence assuming task 3 is assigned to recipient 2, we obtain

$$\begin{pmatrix} 0 & 2 & 4 \\ 0 & 6 & 2 \\ 4 & 0 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 2 & 4 \\ 0 & 6 & 2 \\ 4 & 0 & 0 \end{pmatrix} \tag{9.14}$$

which leaves us with only the non-zero coefficient 2. This does not mean our problem is infeasible, as we were left with the forced option of assigning task 2 to recipient 3 ($x_{2,3} = 1$). **The fact that this assignment is mapped by a coefficient greater than zero implies that our solution is not optimal**. We shed more light on a related property of the Hungarian algorithm in the **An important property of the Hungarian algorithm** box.

An important property of the Hungarian algorithm

Let us assume we have a cost matrix C where row and column operations have been carried out so that in every row and column there is *at least* a 0. **The minimum number of horizontal and vertical lines needed to cross out all the 0s in C is equivalent to the number of optimal assignments (in zero-cost elements) of tasks to recipients**. Considering cost matrix (9.9), we need 3 lines (although more than one configuration is possible as shown in (9.15)):

$$\begin{pmatrix} 4 & 2 & 0 \\ 0 & 4 & 6 \\ 5 & 0 & 7 \end{pmatrix}, \begin{pmatrix} 4 & 2 & 0 \\ 0 & 4 & 6 \\ 5 & 0 & 7 \end{pmatrix}, \begin{pmatrix} 4 & 2 & 0 \\ 0 & 4 & 6 \\ 5 & 0 & 7 \end{pmatrix}, \cdots \tag{9.15}$$

This implies that an optimal assignment can be computed. On the other hand, considering cost matrix (9.13) only 2 lines are needed (see matrix (9.16)):

$$\begin{pmatrix} 0 & 2 & 4 \\ 0 & 6 & 2 \\ 4 & 0 & 0 \end{pmatrix} \tag{9.16}$$

This implies that with the current cost matrix, only two assignments at zero cost (in the revised cost matrix) can be carried out.

Whenever we can cross all the current 0s in the cost matrix with fewer than $|\mathcal{T}|$ lines, we cannot identify an optimal assignment. **Hence, we need to generate additional 0s somewhere else in the matrix without**

deleting the existing ones or creating negative numbers. We first provide a couple of definitions, then explain a procedure to generate new 0s without affecting the existing ones by modifying cost matrix (9.13), and then formalize the Hungarian algorithm.

When we cover all the 0s in a cost matrix with the minimum number of lines necessary, elements not covered by any line are **uncovered**, elements covered by a single line are **covered**, and elements covered by two lines are **double covered**. Considering cost matrix (9.16), the 4 in position $(3, 1)$ is double covered, the four 0s are covered, and the remaining numbers are uncovered.

To generate at least one additional 0, a potential strategy could be to determine the smallest uncovered coefficient and subtract such a value from the row (or column) where it appears. In our case, such a value is 2 (appearing twice), and we decide to subtract it from the first row (not the only option possible). We obtain

$$C = \begin{pmatrix} -2 & 0 & 2 \\ 0 & 6 & 2 \\ 4 & 0 & 0 \end{pmatrix} \tag{9.17}$$

which does not solve our problem, as the 0 in position $(1, 1)$ has now become a negative number. Hence, our next step to fix the problem is to add 2 to the whole first column, leading to:

$$C = \begin{pmatrix} 0 & 0 & 2 \\ 2 & 6 & 2 \\ 6 & 0 & 0 \end{pmatrix} \tag{9.18}$$

While we restored the 0 in position $(1, 1)$ in cost matrix (9.18), we transformed the 0 in position $(2, 1)$ into a 2. To fix this new issue, we deduct 2 from the whole second row, leading to:

$$C = \begin{pmatrix} 0 & 0 & 2 \\ 0 & 4 & 0 \\ 6 & 0 & 0 \end{pmatrix} \tag{9.19}$$

In cost matrix (9.19) we now need 3 lines to cover all the 0s. This allows us to complete all three assignments using zero-cost elements. **Note that being able to complete all assignments using zero-cost elements, which ensures optimality, does not entail that the optimal solution is unique**. For example, there are two distinct ways to perform an optimal assignment given cost matrix (9.19) as reported below:

$$\begin{pmatrix} 0 & 0 & 2 \\ 0 & 4 & 0 \\ 6 & 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 2 \\ 0 & 4 & 0 \\ 6 & 0 & 0 \end{pmatrix} \tag{9.20}$$

Both the leftmost solution $x_{1,1}, x_{2,3}, x_{3,2}$ and the rightmost solution $x_{1,2}, x_{2,1}, x_{3,3}$ yield an overall cost of 13 if we consider the original cost matrix (9.11). Concerning how to select the sequence of assignments, it is suggested to start with the row or column containing the fewest 0s and cross out both the row and column associated with the current selected 0

(as previously shown, e.g., in (9.10)) and to repeat the process following the same logic until all assignments have been performed.

Before formalizing the algorithm, let us compare the first version of the modified cost matrix (9.16) with the 2 crossing lines (which did not allow us to perform an optimal assignment) and the revised version (9.18) that allowed us to compute an optimal assignment. **Uncovered coefficients have been reduced by a value equal to the smallest uncovered coefficient (to create new 0s without generating negative numbers). Covered coefficients remained unchanged. As shown above, they remained unchanged because the same coefficient was first removed and then added (or vice versa) to the associated row or column. Finally, double covered coefficients were increased by a value equal to the smallest uncovered coefficient**. We can now formalize the Hungarian algorithm in the **The Hungarian algorithm** box.

The Hungarian algorithm

- **Inputs**: original cost matrix C of dimension $(|\mathcal{T}|, |\mathcal{T}|)$
- subtract the smallest number from every row in C (**row reduction**);
- subtract the smallest number from every column in C (**column reduction**);
- compute minimum number of lines N_l needed to cover all the 0s in C;
- WHILE $N_l < |\mathcal{T}|$:
  - determine smallest value ϵ among uncovered coefficients;
  - reduce all uncovered coefficients by ϵ;
  - increase all double covered coefficient by ϵ;
  - recompute N_l;
- select the 0s associated with the optimal assignment by **starting with the row or column with fewer 0s, selecting one and crossing out its associated row and column**. Repeat until the required $|\mathcal{T}|$ 0s have been selected;
- compute the objective value identifying the proper cost coefficients in the original cost matrix C.
- **Outputs**: (task,recipient) assignment and overall assignment cost

A final consideration regarding the Hungarian algorithm and the assignment problem in general is the rather restrictive assumption that the number of tasks must match the number of recipients. While this requirement is met in some practical examples, it may not hold true for others. Let us consider a portfolio that group projects that students can individually score so that a final assignment can be made with students assigned to projects they scored positively. Because the number of projects is generally (much) smaller than the number of students, we have $|\mathcal{T}| \ll |\mathcal{R}|$ (we assume projects are tasks and students are the recipients). Hence, we do not satisfy the main assumption of an assignment problem. To fix the issue, **dummy tasks or dummy recipients can be added to restore the symmetry**. For example, let us assume the case of six students and two projects. Using the convention, consistent with the original model (9.1)-(9.4) that rows represent tasks (projects) and columns are recipients (students), the cost matrix C is a $(2, 6)$ matrix in this case. Because there is an implicit assumption that groups should be as

homogeneous as possible in terms of students, the final assignment will entail two groups of three students each. To this avail, we can vertically concatenate C with two copies of itself (highlighted in shaded red), as shown in (9.21).

$$\begin{pmatrix} 4 & 3 & 2 & 3 & 3 & 5 \\ 2 & 5 & 6 & 1 & 2 & 4 \\ 4 & 3 & 2 & 3 & 3 & 5 \\ 2 & 5 & 6 & 1 & 2 & 4 \\ 4 & 3 & 2 & 3 & 3 & 5 \\ 2 & 5 & 6 & 1 & 2 & 4 \end{pmatrix} \tag{9.21}$$

Such a cost matrix can be used to perform the optimal assignment and, as a post-processing step, the "real" projects can be assigned to students. For example, let use assume one optimal assignment is $x_{3,6} = 1$, i.e., student 6 is assigned to project 3. Because project 3 is a copy of project 1, then student 6 is, in reality, assigned to project 1. On a similar note, if $x_{6,1} = 1$, then student 1 is assigned to project 2 (being project 6 a copy of project 2).

A slight complication arises when the original values of $|\mathcal{T}|$ are not multiple of each other. In such a case, we need to concatenate both vertically (with copies of the original C cost matrix) and horizontally (with dummy recipients) the original C. Because dummy recipients are added to ensure symmetry, all their cost coefficients can be set at 0. Let us assume we now have 2 projects and 3 students and the original cost matrix C is

$$C = \begin{pmatrix} 2 & 1 & 4 \\ 1 & 4 & 2 \end{pmatrix} \tag{9.22}$$

By horizontally padding a copy of itself, we can achieve a $(4, 3)$ cost matrix. To restore symmetry, we need to vertically pad it with a zero-valued cost coefficient vector highlighted in shaded orange representing the introduction of a dummy student. The resulting C is shown in matrix (9.23)

$$C = \begin{pmatrix} 2 & 1 & 4 & 0 \\ 1 & 4 & 2 & 0 \\ 2 & 1 & 4 & 0 \\ 1 & 4 & 2 & 0 \end{pmatrix} \tag{9.23}$$

We leave it to readers to verify that, by using the Hungarian algorithm, the optimal assignment entails assigning students 1 and 2 to project 1, and student 3 to project 2 for an overall cost of 5. It is worth noting that student 4, being a dummy student, is not mentioned in the final assignment.

We showcase more thoroughly a non-symmetric assignment problem in Example 9.2

Example 9.2 *6 teaching assistants are needed to assist students who are divided into 3 groups and are working on their final year projects. Each teaching assistant has assigned scores to each project based on personal preferences, with lower*

scores indicating better matches for the teaching assistant. The resulting score matrix C is highlighted in (9.24). The goal is to assign teaching assistants to projects to maximize the quality of the matching.

$$C = \begin{pmatrix} 2 & 1 & 4 & 3 & 7 & 5 \\ 1 & 4 & 2 & 1 & 4 & 9 \\ 3 & 2 & 5 & 3 & 6 & 4 \end{pmatrix} \tag{9.24}$$

We realize this is an assignment problem in nature, where the original cost coefficient matrix is a $(3, 6)$ matrix and we hence need to duplicate it vertically before applying the Hungarian algorithm. In doing so, we assume that dummy projects 4, 5, and 6 are, respectively, projects 1, 2, and 3. Hence, two teaching assistants will be assigned to each project. The revised cost coefficient matrix is

$$C = \begin{pmatrix} 2 & 1 & 4 & 3 & 7 & 5 \\ 1 & 4 & 2 & 1 & 4 & 9 \\ 3 & 2 & 5 & 3 & 6 & 4 \\ 2 & 1 & 4 & 3 & 7 & 5 \\ 1 & 4 & 2 & 1 & 4 & 9 \\ 3 & 2 & 5 & 3 & 6 & 4 \end{pmatrix} \tag{9.25}$$

and, after performing row and column reduction, we obtain

$$C = \begin{pmatrix} 1 & 0 & 2 & 2 & 3 & 2 \\ 0 & 3 & 0 & 0 & 0 & 6 \\ 1 & 0 & 2 & 1 & 1 & 0 \\ 1 & 0 & 2 & 2 & 3 & 2 \\ 0 & 3 & 0 & 0 & 0 & 6 \\ 1 & 0 & 2 & 1 & 1 & 0 \end{pmatrix} \tag{9.26}$$

where all the 0s can be covered with just 4 lines as shown in matrix (9.25). This means we cannot yet perform all the assignments optimally.

$$C = \begin{pmatrix} 1 & 0 & 2 & 2 & 3 & 2 \\ 0 & 3 & 0 & 0 & 0 & 6 \\ 1 & 0 & 2 & 1 & 1 & 0 \\ 1 & 0 & 2 & 2 & 3 & 2 \\ 0 & 3 & 0 & 0 & 0 & 6 \\ 1 & 0 & 2 & 1 & 1 & 0 \end{pmatrix} \tag{9.27}$$

By applying the algorithm and reducing all uncovered values by 1 (smallest uncovered value) and increasing all double covered values by 1, we obtain matrix (9.28) which allows the highlighted optimal assignment. Teaching assistants 1 and 2 are assigned to project 1, teaching assistants 3 and 5 to project 2, and teaching assistants 4 and 6 to project 3.

$$C = \begin{pmatrix} 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 4 & 0 & 0 & 0 & 7 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 4 & 0 & 0 & 0 & 7 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{9.28}$$

Retrieving the scores from matrix (9.25), the optimal assignment achieves an overall score of $2 + 1 + 2 + 3 + 4 + 4 = 16$.

Coded example

The code used to model and solve Example 9.2 as a BP to verify the outcome of the Hungarian algorithm is available here.

9.2 Preliminaries of scheduling

Real-life problems invariably involve a critical component that cannot be overlooked: time. When time becomes a factor, we often encounter scheduling problems. **Scheduling involves arranging a set of items in a timeline. In transportation problems, this could mean determining the sequential visitation of nodes along a route. From a production standpoint, scheduling entails organizing the processing of items through machines**.

Consider the following generic scheduling problem. We define a set $\mathcal{N}$ indexed by i of jobs (i.e., activities) that must be performed, each with a processing time P_i. These jobs feature precedence constraints, namely for some pair i, j, the starting times t_i and t_j must be $t_i + P_i \leq t_j$, and also non-overlapping constraints (two jobs cannot be processed at the same time, even partly). The aim could be to minimize the completion time of the final executed task. In this setting, we can consider an unlimited number of resources that can perform these tasks.

Let us start by formally defining the completion time, denoted as c. The completion time represents the moment when the last job in the sequence is completed. While modeling, it is impossible to determine in advance which job will be the last. However, we can infer that the completion time must be greater than or equal to the end time of any job. Therefore, we can formulate our initial constraints and objective function as follows:

$$\min \quad c \tag{9.29}$$

s.t.:

$$c \geq t_i + P_i \qquad \forall i \in \mathcal{N} \tag{9.30}$$

$$t_i \in \mathbb{R}_0 \qquad \forall i \in \mathcal{N} \tag{9.31}$$

We can add non-overlapping constraints to enrich the model: given two jobs $i, j \in \mathcal{N}$[4], we first process i or j. Hence, it is either

$$t_i + P_i \leq t_j \tag{9.32}$$

or

4: With this expression we imply every distinct pair of jobs i, j where $i \in \mathcal{N}$ and $j \in \mathcal{N}$. An equivalent notation is $i, j \in \mathcal{N} \times \mathcal{N}$, which reflects that every permutation of set $\mathcal{N}$ is accounted for. In both cases, sometimes the expression $i \neq j$ could appear to make it explicit that the jobs should be distinct.

$$t_j + P_j \leq t_i \tag{9.33}$$

A binary variable is required to trigger one of the two constraints. Let us define y_{ij}, which is unitary if $i \prec j$[5]. The big-M construct is needed because we want one of the two constraints to be redundant when the other holds. This is an example of either-or constraint as explained in Section 4.8.2. Therefore:

5: $i \prec j$ means task i precedes task j.

$$t_i + P_i \leq t_j + M(1 - y_{ij}) \qquad \forall i, j \in \mathcal{N} \tag{9.34}$$
$$t_j + P_j \leq t_i + M y_{ij} \qquad \forall i, j \in \mathcal{N} \tag{9.35}$$

As desired, if $y_{ij} = 1$ then $i \prec j$ as constraint (9.34) is active and constraint (9.35) is redundant. While constraints (9.34)-(9.35) are general and allow the model to choose the precedence relationship that is best for the objective, the sequence of some i, j pairs of jobs could be pre-defined from the start. One example is if i represents turning on the laptop and j represents completing a piece of code. The latter cannot start if the former is not finished. Hence, we could define set $\mathcal{S}$ that stores all i, j job pairs with pre-defined precedence relationships such that $i \prec j \; \forall i, j \in \mathcal{S}$. For every i, j pair part of this set we can write:

$$t_i + P_i \leq t_j \qquad \forall i, j \in \mathcal{S} \tag{9.36}$$

In conclusion, we should use constraints (9.34)-(9.35) for those i, j job pairs $\notin \mathcal{S}$ because we have the freedom to shuffle them as needed, while constraint (9.36) is preferred for i, j job pairs $\in \mathcal{S}$. **Note that for this second category, we could still employ constraints (9.34)-(9.35) paired with forcing $y_{ij} = 1$, which would indeed yield constraint (9.36), but we are unnecessarily adding to the model some decision variables y_{ij} that we force to be unitary (hence, they are not decision variables in the first place) and constraints (every dummy constraint (9.35) which is not needed at all for i, j job pairs $\in \mathcal{S}$).**

9.3 Single Machine Scheduling Problem (SMSP)

The Single Machine Scheduling Problem (SMSP) is the problem of scheduling a set of jobs or tasks in a single machine.[6] The goal is to minimize, for example, the sum of the completion times or delays (sometimes referred to as tardiness).

6: Note that the concept of machine is quite general, as it does not have to be a physical piece of machinery, but a worker, a lecture hall, etc.

Formulating a mathematical model for a SMSP can be done in at least two ways. The first, explained in this section, works with basic time variables. The second, not presented in this book follows a routing-based approach (see Chapter 13) where jobs are modeled as nodes and our goal is to

connect them via a path such that going from node i to node j implies $i \prec j$ and proper time precedence constraints must be activated.

The mathematical formulation of the SMSP presented here features two related sets. The first one is the set of jobs $\mathcal{N}$ indexed by i. The second one is set $\mathcal{S} \subseteq \mathcal{N} \times \mathcal{N}$[7] containing all the i, j job pairs that must be executed following the precedence $i \prec j$. Each job features a processing time P_i and a deadline D_i, representing the latest time when job i can be completed. In terms of decision variables, t_i represents the start time of job i, c_i its completion time, and $y_{ij} \in \{0, 1\}$ is unitary if job i precedes job j. We already introduce two additional decision variables that will be needed for extensions to this first model. We define c as the latest completion time across all c_is, i.e., $c = \max\{c_1, \cdots, c_{|\mathcal{N}|}\}$, and define d_i as the delay of job i with respect to its deadline D_i. We report all the needed notation for the SMSP in Table 9.2.

7: As mentioned in a previous note, with the expression $\mathcal{N} \times \mathcal{N}$ we mean the set of i, j index pairs where $i \in \mathcal{N}$ and $j \in \mathcal{N}$.

Table 9.2: Notation for the SMSP.

Sets and indices	
$\mathcal{N}$	set of jobs $i \in \mathcal{N}$
$\mathcal{S}$	set of job pairs $i, j \in \mathcal{S} \subseteq \mathcal{N} \times \mathcal{N}$ such that $i \prec j$
Parameters	
P_i	processing time of job $i \in \mathcal{N}$
D_i	deadline of job $i \in \mathcal{N}$
Variables	
$t_i \in \mathbb{R}_0$	start time of job i
$c_i \in \mathbb{R}_0$	completion time of job i
$y_{ij} \in \{0, 1\}$	unitary if job i precedes job j
$c \in \mathbb{R}_0$	latest completion time across all jobs $i \in \mathcal{N}$
$d_i \in \mathbb{R}_0$	delay of job i

An LP model for the SMSP minimizing completion times is:

$$\min \sum_{i \in \mathcal{N}} c_i \tag{9.37}$$

s.t.:

$$c_i \geq t_i + P_i \qquad \forall i \in \mathcal{N} \tag{9.38}$$
$$t_i + P_i \leq t_j + M(1 - y_{ij}) \qquad \forall i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j \tag{9.39}$$
$$t_j + P_j \leq t_i + M y_{ij} \qquad \forall i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j \tag{9.40}$$
$$t_i + P_i \leq t_j \qquad \forall i, j \in \mathcal{S} \tag{9.41}$$
$$t_i \leq D_i - P_i \qquad \forall i \in \mathcal{N} \tag{9.42}$$
$$c_i \leq D_i \qquad \forall i \in \mathcal{N} \tag{9.43}$$
$$y_{ij} \in \{0, 1\} \qquad \forall i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j \tag{9.44}$$

(9.37) aims at minimizing the summation of all completion times. Constraint (9.38) imposes that the completion time of a job should be greater or equal to its start time plus its completion time. Formally, it might be argued that constraint (9.38) should be in an equality form, but as the model aims at minimizing the objective (where all c_is appear), the equality will be automatically satisfied. Constraints (9.39)-(9.40) define

precedence relations between job pairs where a pre-defined sequence is not imposed, while constraint (9.41) ensures that every job i is finished before every job j for every (i, j) job pair with pre-defined precedence relationships. Constraints (9.42)-(9.44) define the nature and range of the decision variables. We elaborate a bit more on the meaning of expression $\forall\, i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j$ in constraints (9.39)-(9.40) and (9.44) in the **A note on the definition of the y_{ij} decision variables in scheduling problems** box.

A note on the definition of the y_{ij} decision variables in scheduling problems

Let us consider constraints (9.39)-(9.40) and (9.44). On top of considering only job pairs for which no pre-defined order is given ($\forall\, i, j \in \mathcal{N} \setminus \{\mathcal{S}\}$), we also want to define such constraints for job pairs where the index i of the first job is smaller than the index j of the second job ($\wedge\, i < j$). While omitting this second condition does not imply a formal mistake, it unnecessarily defines additional decision variables. Because every y_{ij} defines a *precedence relationship* between two jobs, if a specific y_{ij} is assigned a unitary value, this implies that $y_{ji} = 0$. In words: *if job i precedes job j, then job j comes after job i*. The aforementioned intuition can be translated into $y_{ij} + y_{ji} = 1\ \forall\, (i, j) \in \mathcal{N}$. Instead of forcing such a constraint, we only define y_{ij} for potential job pairs where $i < j$. For example, $y_{1,4} = 1$ automatically implies $y_{4,1} = 0$ without the need to define the latter variable. **If a specific y_{ij} is unitary, this automatically implies that j comes after i (or vice versa) with no need for the redundant y_{ji} decision variable. Defining $y_{ij}\ \forall\, i, j \in \mathcal{N} \setminus \{\mathcal{S}\}$ would not change the result, but unnecessarily increases the size of the mathematical model**.

If a single machine is responsible for processing all the required jobs, it is essential to consider the final completion time as a critical KPI. This metric signifies the moment when the entire process concludes. We can incorporate this aspect into the showcased SMSP model with the following variant:

$$\min \quad c \tag{9.45}$$

s.t.:

$$c_i \geq t_i + P_i \qquad \forall i \in \mathcal{N} \tag{9.46}$$

$$c \geq c_i \qquad \forall i \in \mathcal{N} \tag{9.47}$$

$$t_i + P_i \leq t_j + M(1 - y_{ij}) \qquad \forall i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j \tag{9.48}$$

$$t_j + P_j \leq t_i + M y_{ij} \qquad \forall i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j \tag{9.49}$$

$$t_i + P_i \leq t_j \qquad \forall i, j \in \mathcal{S} \tag{9.50}$$

$$t_i \leq D_i - P_i \qquad \forall i \in \mathcal{N} \tag{9.51}$$

$$c_i \leq D_i \qquad \forall i \in \mathcal{N} \tag{9.52}$$

$$y_{ij} \in \{0, 1\} \qquad \forall i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j \tag{9.53}$$

$$c \in \mathbb{R}_0 \tag{9.54}$$

Now the objective function (9.45) aims at minimizing c, the final completion time. The model does not know beforehand which job will be the last one, hence we need constraint set (9.47) to define c as the largest across all c_is (**this constraint set linearizes the** max **non-linear operator defined in the text above**). This approach is known as a **min-max problem**[8]. The goal is to minimize the maximum across a set of values. In this specific case, the set of values is the set of all completion times. The rest of the constraints are inherited directly from the original formulation, with (9.53)-(9.54) defining the additional decision variables.

8: There exists also the max-min counterpart, where the model aims at maximizing the minimum across a set of values

Both variants of the SMSP presented so far entail that every job can be completed within its deadline. While deadlines are set with the expectations of being met, we all know that reality is slightly different. To this avail, another variant to the SMSP might be defined where we allow jobs to be finished after their specified deadline. To limit this unwanted (yet allowed) behavior, a proper objective function aims at minimizing the summation of delays. The mathematical formulation of this variant is:

$$\min \sum_{i \in \mathcal{N}} d_i \tag{9.55}$$

s.t.:

$$t_i + P_i \leq t_j + M(1 - y_{ij}) \qquad \forall i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j \tag{9.56}$$
$$t_j + P_j \leq t_i + M y_{ij} \qquad \forall i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j \tag{9.57}$$
$$t_i + P_i \leq t_j \qquad \forall i, j \in \mathcal{S} \tag{9.58}$$
$$t_i + P_i - d_i \leq D_i \qquad \forall i \in \mathcal{N} \tag{9.59}$$
$$t_i \in \mathbb{R}_0 \qquad \forall i \in \mathcal{N} \tag{9.60}$$
$$y_{ij} \in \{0, 1\} \qquad \forall i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j \tag{9.61}$$
$$d_i \in \mathbb{R}_0 \qquad \forall i \in \mathcal{N} \tag{9.62}$$

where (9.55) aims at minimizing the summation of all delays. Most constraints are inherited from the previous models, with constraint set (9.59) being specific to this variant. **It allows the completion time of job** i**, i.e.,** $t_i + P_i$ **to exceed its deadline** D_i **pending the "activation" of delay variable** d_i**. If every job can be finished within the deadline, then** $d_i = 0 \, \forall i \in \mathcal{N}$ **which implies** $\min \sum_{i \in \mathcal{N}} d_i = 0$.

Readers might contend that minimizing the total sum of delays may not always be optimal, as a solution could potentially involve a combination of numerous small and a few significant delays. Therefore, an alternative min-max approach similar to the one showcased for the completion time can be devised.

9.4 Parallel Machine Scheduling Problem (PMSP)

The Parallel Machine Scheduling Problem (PMSP) concerns the scheduling of a set of tasks on two or more machines. There are several variants, and the one presented here assumes that a task should be entirely processed in one machine (hence, a task cannot be started on one machine and then moved and completed on another). Regarding the objective function, multiple options are possible in accordance with what was discussed in Section 9.3. In the following, we present a PMSP formulation that minimizes the maximum completion time.

In terms of modeling, most of the notation is inherited from the PMSP. An additional decision layer is now required to assign each job to a specific machine. This can be accomplished using the binary decision variable x_{im}, which is unitary if job $i \in \mathcal{N}$ is assigned to machine $m \in \mathcal{M}$. We define the notation for the SMSP in Table 9.3.

Table 9.3: Notation for the PMSP.

Sets and indices	
$\mathcal{N}$	set of jobs $i \in \mathcal{N}$
$\mathcal{S}$	set of job pairs $i, j \in \mathcal{S} \subseteq \mathcal{N} \times \mathcal{N}$ such that $i \prec j$
$\mathcal{M}$	set of machines $m \in \mathcal{M}$
Parameters	
P_i	processing time of job $i \in \mathcal{N}$
D_i	deadline of job $i \in \mathcal{N}$
Variables	
$t_i \in \mathbb{R}_0$	start time of job i
$c_i \in \mathbb{R}_0$	completion time of job i
$x_{im} \in \{0,1\}$	unitary if job m is assigned to machine m
$y_{ij} \in \{0,1\}$	unitary if job i precedes job j
$c \in \mathbb{R}_0$	latest completion time across all jobs $i \in \mathcal{N}$

The mathematical formulation for the version of the PMSP defined above is:

$$\min \; c \tag{9.63}$$

s.t.:

$$\sum_{m \in \mathcal{M}} x_{im} = 1 \qquad \forall i \in \mathcal{N} \tag{9.64}$$

$$c_i \geq t_i + P_i \qquad \forall i \in \mathcal{N} \tag{9.65}$$

$$c \geq c_i \qquad \forall i \in \mathcal{N} \tag{9.66}$$

$$t_i + P_i \leq t_j + M(3 - y_{ij} - x_{im} - x_{jm}) \qquad \forall i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j, m \in \mathcal{M} \tag{9.67}$$

$$t_j + P_j \leq t_i + M y_{ij} \qquad \forall i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j \tag{9.68}$$

$$t_i + P_i \leq t_j \qquad \forall i, j \in \mathcal{S} \tag{9.69}$$

$$t_i \leq D_i - P_i \qquad \forall i \in \mathcal{N} \tag{9.70}$$

$$c_i \leq D_i \qquad \forall i \in \mathcal{N} \tag{9.71}$$

$$x_{im} \in \{0, 1\} \qquad \forall i \in \mathcal{N}, m \in \mathcal{M} \tag{9.72}$$

$$y_{ij} \in \{0, 1\} \qquad \forall i, j \in \mathcal{N} \setminus \{\mathcal{S}\} \wedge i < j \tag{9.73}$$

$$c \in \mathbb{R}_0 \tag{9.74}$$

(9.63) minimizes the overall completion time. Constraint (9.64) ensures that each job is assigned to exactly one machine. Constraint (9.65) defines the completion time of each job, while constraint (9.66) maps the maximum completion time across all jobs. Constraints (9.67)-(9.68) ensure that time precedence constraints are satisfied for jobs assigned to the same machine. In particular, if both jobs i and j are assigned to machine m ($x_{im} = 1$ and $x_{jm} = 1$) and i precedes j ($y_{ij} = 1$), then constraint (9.67) is active ($t_i + P_i \leq t_j$) and constraint (9.68) is dummy. Constraint (9.68) implies that $t_j + P_j \leq t_i$ if $y_{ij} = 0$ regardless of the machines the two jobs are assigned to, with constraint (9.67) being redundant. Constraint (9.69) imposes time precedence for those jobs that are required by a pre-defined time hierarchy, while (9.70)-(9.74) define the nature and range of the decision variables.

We showcase an application example of the PMSP in Example 9.3.

Example 9.3 *3 university students are studying altogether for an exam. They want to study individually 6 main topics so that they can later summarize them together. Each topic $i \in \mathcal{N}$ is expected to take P_i hours to be mastered. In addition, the three students have assigned to each topic a deadline D_i such that topic i must be completed by that time. Due to the prerequisite nature of certain topics, there exist precedence relationships among the six topics. In other words, some topics cannot be started until others have been completed. All pertinent information regarding the six topics is provided in Table 9.4. The objective is to formulate a mathematical model to assist the students in distributing the workload such that each topic is covered by one of them, accounting for the required precedence, with the aim of completing the study session as expeditiously as possible.*

Table 9.4: Data pertaining to the six topics of Example 9.3.

Topic	i	P_i	D_i	Precedence
Derivatives	1	3	5	-
Integrals	2	3	3	-
Heat equation	3	4	7	1
Biot–Savart law	4	1	4	2
Kirchhoff's circuit laws	5	6	6	-
Maxwell's equations	6	3	10	5

We recognize that this is a PMSP where the 6 topics are the jobs $i \in \mathcal{N}$ and the 3 students the 3 machines $m \in \mathcal{M}$. In addition, the set of pre-defined time precedence requirements is $\mathcal{S} = \{(1,3),(2,4),(5,6)\}$. Our goal is to minimize the latest completion time as in (9.63). We do not show the full set of constraints, as it would take up too much space, but directly display the obtained solution in Figure 9.1 and critically assess it.

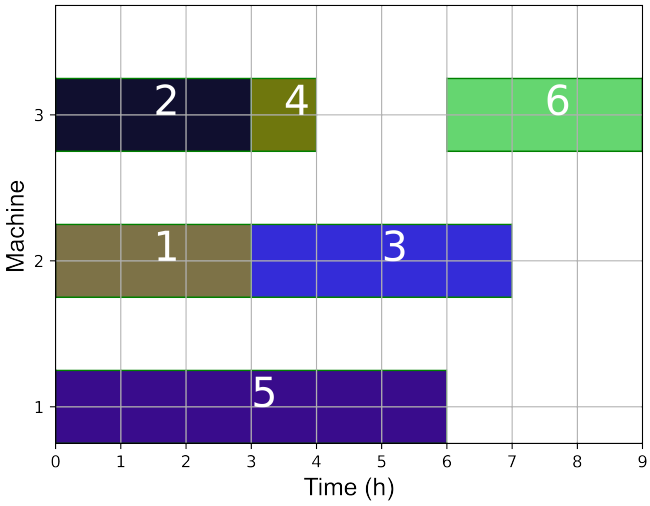


Figure 9.1: Final solution for the PMSP of Example 9.3.

The optimal solution suggests an overall completion time of 9 hours. Student 1 starts at time $t = 0$ with topic $i = 5$. Because such a topic has duration of 6 hours and must be completed within 6 hours ($D_5 = 6$), the only feasible option is that a student starts immediately with it. Because student 1 is occupied for 6 hours with that topic, the other two students split the remaining workload. Student 2 takes care of topic 1 and then 3, so that the precedence relationship is met. On a similar note, student 3 takes care of topic 2 and then 4. The only remaining topic is then 6, which is assigned by the model to student 3, who cannot tackle it immediately after being done with topic 4 as topic 5 is not completed yet. An equivalent solution would entail assigning topic 6 to student 1 right after the completion of topic 5. **Additionally, note that this solution has symmetrical equivalent solutions that can be obtained by shuffling the assigned topics to a different student. For example, if student 1 takes care of topics 1 and 3, student 2 of topics 2,4, and 6, and student 3 of topic 5, the solution will stay the same. It is worth noting that in our model, not all topics with precedence relationships are necessarily assigned to the same machine (student). For instance, while topic 5 is assigned to student 1, topic 6 is assigned to student 3. Although permitted by our model, this arrangement may not be ideal in practice, as it is typically preferable for the same student to study a topic and any prerequisite topics. Consequently, in some variants of the PMSP, we may require that jobs with specific precedence relationships must be handled by the same machine. Formulating such additional constraints is relatively straightforward.**

Coded example

The code used to model and solve Example 9.3 is available here.

PMSP as a serious game

A serious game based on the PMSP can be found here.

9.5 p-median problem

The p-median problem is a typical network design problem that entails allocating a set of nodes $\mathcal{N}$ (indexed by i) to at most p potential locations from a set $\mathcal{F}$ indexed by f. Nodes are typically referred to as *demand points* and locations as *facilities*. The goal is to minimize the sum of the distances between the nodes and their assigned facilities. **One of the main applications of the p-median problem is the opening of certain facilities when there is a limited budget. An example could be the location of warehouses in a new region. In line with the goal of the model, it is desirable that the sum of the distances between the warehouses and the demand points is minimized**.

Because distances play a crucial role in the problem, we define D_{if} as the distance between demand node $i \in \mathcal{N}$ and facility $f \in \mathcal{F}$. The other parameter is $p \leq |\mathcal{F}|$, i.e., the maximum number of facilities to be potentially opened. The variables are all binary. First, we define the assignment variable x_{if}, unitary if demand node $i \in \mathcal{N}$ is assigned to facility $f \in \mathcal{F}$. Next, the activation variable y_f, with $f \in \mathcal{F}$, unitary if facility $f \in \mathcal{F}$ is active (i.e., if at least one demand node is connected to it). The notation for the p-median problem is shown in Table 9.5

Table 9.5: Notation for the p-median problem.

Sets and indices	
$\mathcal{N}$	set of demand nodes $i \in \mathcal{N}$
$\mathcal{F}$	set of facilities $f \in \mathcal{F}$
Parameters	
D_{if}	distance between demand node $i \in \mathcal{N}$ and facility $f \in \mathcal{F}$
p	maximum number of facilities to use
Variables	
$x_{if} \in \{0,1\}$	unitary if demand node i is served by facility f
$y_f \in \{0,1\}$	unitary if facility f is used

The BP describing the p-median problem is:

$$\min \sum_{i \in \mathcal{N}} \sum_{f \in \mathcal{F}} D_{if} x_{if} \tag{9.75}$$

s.t.:

$$\sum_{f \in \mathcal{F}} x_{if} = 1 \qquad \forall i \in \mathcal{N} \tag{9.76}$$

$$\sum_{f \in \mathcal{F}} y_f \leq p \tag{9.77}$$

$$\sum_{i \in \mathcal{N}} x_{if} \leq |\mathcal{N}| y_f \qquad \forall f \in \mathcal{F} \tag{9.78}$$

$$x_{if} \in \{0, 1\} \qquad \forall i \in \mathcal{N}, j \in \mathcal{F} \tag{9.79}$$

$$y_f \in \{0, 1\} \qquad \forall f \in \mathcal{F} \tag{9.80}$$

(9.75) aims at minimizing the overall distance connecting demand nodes with facilities. Constraint (9.76) enforces that each demand node is connected to a facility, while constraint (9.77) ensures that at most p facilities are used. Constraint (9.78) does not allow any demand node $i \in \mathcal{N}$ to be connected to facility $f \in \mathcal{F}$ unless the associated y_f is unitary[9]. Finally, (9.79)-(9.80) define the nature of the decision variables.

9: In Section 10.2.1 we will see an alternative way of modeling such a constraint.

We showcase an application of the p-median problem in Example 9.4.

Example 9.4 *A logistics company has identified 100 new customers to be served in a* 20×20 *km area where 10 of its facilities are present. The location of both customers and facilities is shown in Figure 9.2. The company would like to assess the benefit of using a different number of facilities to serve the identified set of customers. The goal is to devise a mathematical model that assigns customers to a facility, given a maximum number of exploitable facilities, with the goal of minimizing the overall distance between customers and facilities.*

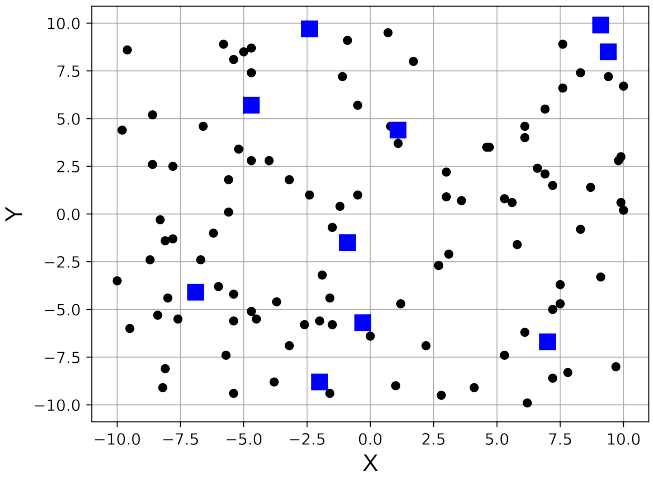
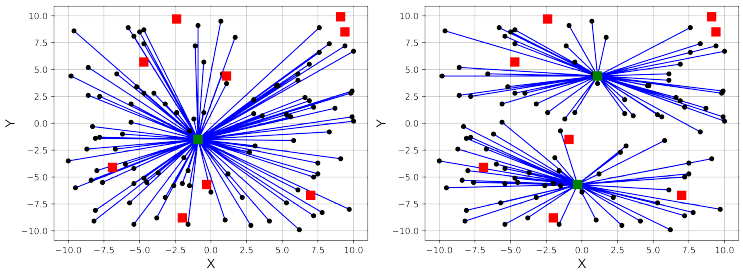


Figure 9.2: Location of the customers and facilities for Example 9.4.

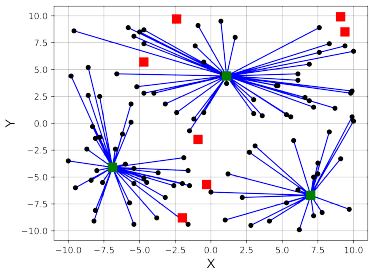
We recognize that this problem is a p-median one because we have a maximum cap on the number of facilities we can use and the goal is to minimize the distance between customers and their assigned facility. Because we are asked to assess the impact of different values of p on the solution, we solve 10 versions of the same model where the only difference is p, which we range from 1 to 10. We showcase the different final assignments in Figure 9.3, where used and unused facilities are highlighted, as well as distances between customers and the assigned facility.

The solution changes quite substantially moving from $p = 1$ (Figure 9.3a) to $p = 10$ (Figure 9.3j). In the first scenario, the model seeks to activate a

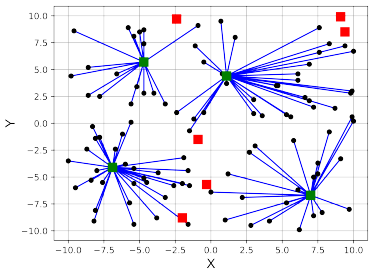


(a) $p = 1$.

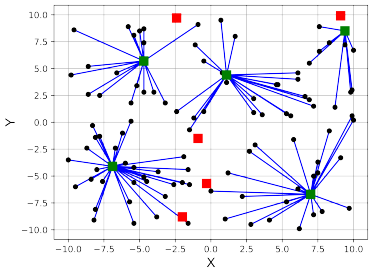
(b) $p = 2$.



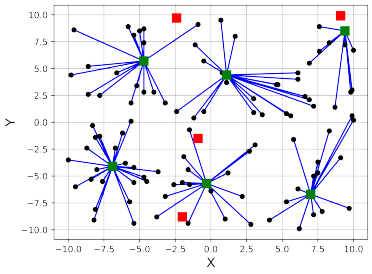
(c) $p = 3$.



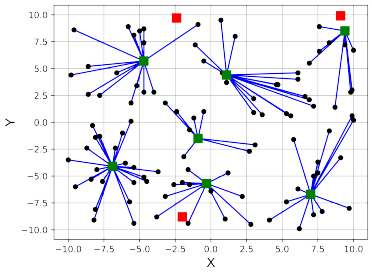
(d) $p = 4$.



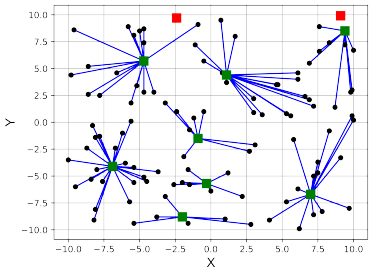
(e) $p = 5$.



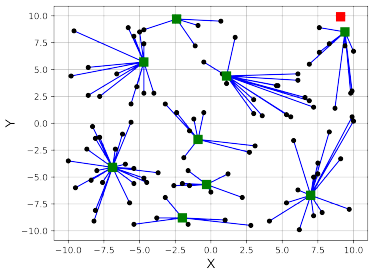
(f) $p = 6$.



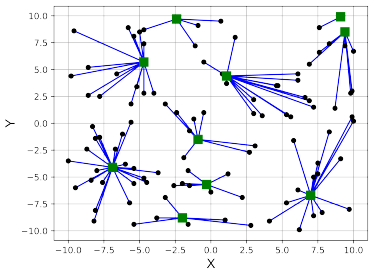
(g) $p = 7$.



(h) $p = 8$.



(i) $p = 9$.



(j) $p = 10$.

Figure 9.3: Assignment of demand nodes to facilities for different values of p in Example 9.4. The used facilities are highlighted in **green**, the others in **red**.

single facility to minimize the distances to all 100 customers. **As intuition would imply, the selected facility is typically the most centrally located within the area of interest**. Increasing p allows for more flexibility and de-centralization. Figure 9.3j showcases this as facilities are assigned to their closest customers, with a facility even serving a single customer.

The advantage in terms of objective value becomes apparent when we visualize the actual objective, namely the cumulative distance between customers and facilities, as a function of p, as demonstrated in Figure 9.4. Starting from an initial cumulative distance of roughly 800 km when $p = 1$, the model halves it if $p = 5$. Then, the additional reduction due to increasing values of p becomes less prominent. **This can be an insightful managerial consideration to report to the company, i.e., that increasing the number of active facilities beyond a certain value of p does not provide a consistent additional benefit**.

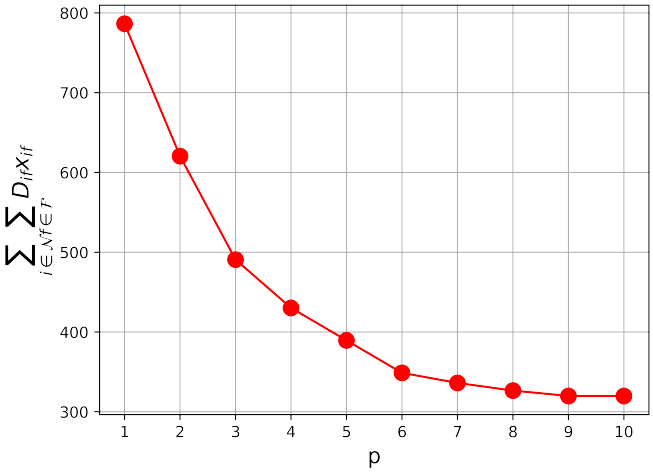


Figure 9.4: Objective value (cumulative distance between customers and facilities) as a function of p for Example 9.4.

Related to the previous point, it is important to emphasize that, in practice, activating and operating a facility typically incurs costs. In the p-median model outlined here, increasing p yields either a better or equivalent objective, as no penalty is incurred for employing additional facilities. However, if such a penalty is necessary, a different model is required, which we will introduce in Section 9.6.

Coded example

The code used to model and solve Example 9.4 is available here.

9.6 Facility location problem

A facility location problem relies on the same set of inputs as the p-median problem defined in Section 9.5, namely a set of demand nodes or customers $i \in \mathcal{N}$ and a set of potential facilities $f \in \mathcal{F}$. We would like to stress the term *potential*, as this is in general a strategic problem. A company has identified a set of potential locations $\mathcal{F}$ where to build new facilities. The construction of each facility $f \in \mathcal{F}$ is expected to cost C_f monetary units. Furthermore, serving a customer $i \in \mathcal{N}$ from facility $f \in \mathcal{F}$ incurs a cost C_{if}, which could, for instance, be proportional or at least correlated with the distance D_{if} between the customer and facility. It

is also assumed that every customer $i \in \mathcal{N}$, who generates revenue equal to R_i, must be served. **The objective of the facility location problem is to determine which facilities to construct and which customers to serve from each constructed facility, maximizing overall profit**.

To achieve the goal, the same decision variables as in the p-median problem are employed, although y_f has a slightly different practical connotation, i.e., it maps whether facility f is built (and used) rather than just used. We report the notation needed for the facility location problem in Table 9.6.

Table 9.6: Notation for the facility location problem.

Sets and indices	
$\mathcal{N}$	set of demand nodes $i \in \mathcal{N}$
$\mathcal{F}$	set of facilities $f \in \mathcal{F}$
Parameters	
C_{if}	cost of serving demand node $i \in \mathcal{N}$ from facility $f \in \mathcal{F}$
C_f	cost of building facility $f \in \mathcal{F}$
Variables	
$x_{if} \in \{0,1\}$	unitary if demand node i is served by facility f
$y_f \in \{0,1\}$	unitary if facility f is built

The BP modeling the facility location problem is:

$$\min \sum_{i \in \mathcal{N}} \sum_{f \in \mathcal{F}} C_{if} x_{if} + \sum_{f \in \mathcal{F}} C_f y_f \tag{9.81}$$

s.t.:

$$\sum_{f \in \mathcal{F}} x_{if} = 1 \qquad \forall i \in \mathcal{N} \tag{9.82}$$

$$\sum_{i \in \mathcal{N}} x_{if} \leq |\mathcal{N}| y_f \qquad \forall f \in \mathcal{F} \tag{9.83}$$

$$x_{if} \in \{0,1\} \qquad \forall i \in \mathcal{N}, f \in \mathcal{F} \tag{9.84}$$

$$y_f \in \{0,1\} \qquad \forall f \in \mathcal{F} \tag{9.85}$$

(9.81) minimizes the overall cost. Recall that we stated that our goal is to maximize profit. The formal definition of profit for the problem at hand is

$$\sum_{i \in \mathcal{N}} \sum_{f \in \mathcal{F}} R_i x_{if} - \sum_{i \in \mathcal{N}} \sum_{f \in \mathcal{F}} C_{if} x_{if} - \sum_{f \in \mathcal{F}} C_f y_f \tag{9.86}$$

but as we impose that every customer must be visited in constraint (9.82), the first term of (9.86) is a constant equal to $\sum_{i \in \mathcal{N}} R_i$. **Hence, maximizing (9.86) is equivalent to minimizing the overall costs as defined in (9.81)**. Constraint set (9.83) ensures that customers can be served from a facility if that facility is built, while constraints (9.84)-(9.85) define the binary nature of the decision variables.

We showcase an application of the facility location problem defined in the same initial setting as the p-median case of Example 9.4 in Example 9.5

Example 9.5 *The same company from Example 9.4 wants to perform a what-if analysis. Considering the same* 20×20 *km area, 100 customers, and the location of the 10 facilities, the company now assumes that no facility has been built yet and plans to strategically build some of them in the coming years. The company has computed an estimate* C_f *of how much the construction of each facility* $f \in \mathcal{F}$ *would cost (the estimate is considered to be accurate. Hence, it will not change according to inflation and other events). In addition, it has been estimated that the service cost related to servicing a specific customer* $i \in \mathcal{N}$ *from a facility* $f \in \mathcal{F}$ *is directly proportional to the distance* D_{if} *via a constant* C_D*. Such a constant will probably change between now and when customers will be served, and it has been estimated to go as low as* $200,000$€ *per unit distance (best-case scenario) and as high as* $500,000$€ *per unit distance (worst-case scenario). The goal is to assist the company in assessing the optimal selection of facilities and customers to be served by each facility in the two scenarios with the goal of minimizing overall costs. We report the location of the customers and facilities in Figure 9.5 and the construction costs* C_f *in Table 9.7 (costs are divided by* $1,000$ *for the sake of readability).*

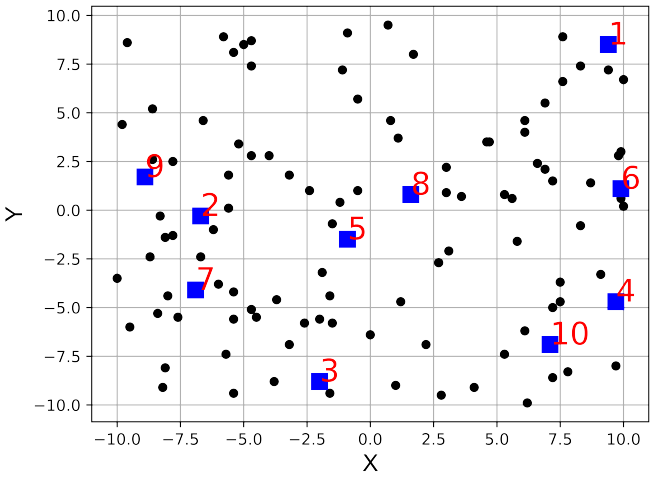


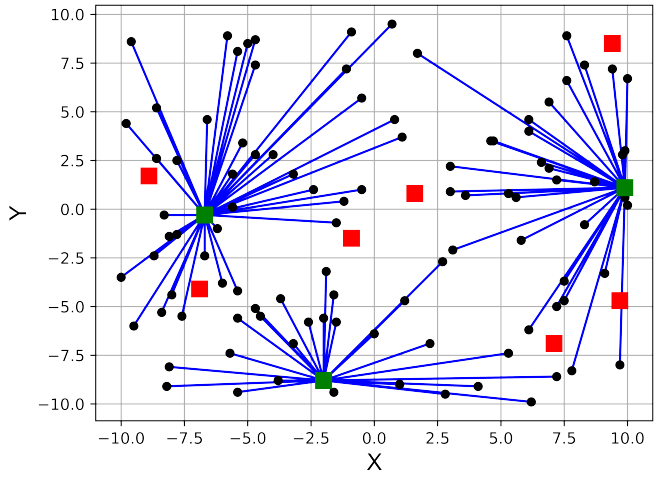
Figure 9.5: Location of the customers and facilities for Example 9.5. We also report the index f of each facility to more easily extract its construction cost from Table 9.7.

Table 9.7: Construction cost C_f for each facility $f \in \mathcal{F}$ of Example 9.5.

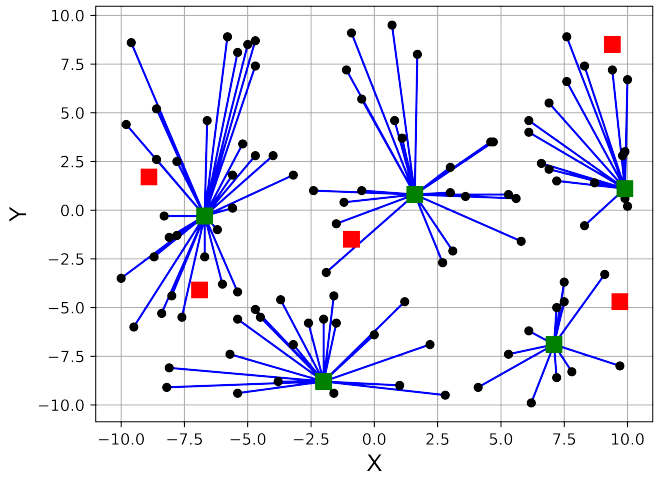
f	C_f
1	18487
2	12137
3	13790
4	17841
5	19573
6	17184
7	19687
8	17480
9	15029
10	19608

We recognize this is a facility location problem and solve it twice, using once the best-case value of $C_D = 200$ and once the worst-case value of $C_D = 500$ (these values are also divided by $1,000$ for consistency with Table 9.7) to determine the servicing costs. We directly report the two solutions in Figure 9.6.

Upon visual examination of the results (Figure 9.6a), we observe that when servicing costs, contingent on the overall distance between customers and facilities, are low, the model favors fewer facilities and longer routes to reach customers. **This behavior is anticipated, as the expense**



(a) $C_D = 200$.



(b) $C_D = 500$.

Figure 9.6: Assignment of demand nodes to facilities for different values of C_D in Example 9.5. The built facilities are highlighted in **green**, the others in **red**.

of constructing an additional facility to shorten routes may not be completely offset by the savings in servicing costs. Conversely, when the servicing costs are at their expected maximum, the model favors more facilities as the extra construction costs are fully compensated by the savings in servicing costs. This behavior is evident in Figure 9.6b.

In the new scenario, the same three facilities ($f = 2, 3, 6$) chosen in the first case, as depicted in Figure 9.6a, are selected again. However, some customers originally assigned to these facilities are now redistributed to two additional facilities ($f = 8, 10$) in order to minimize travel distances. Notably, facility 8 is positioned roughly at the center of the triangle formed by the three originally selected facilities in Figure 9.6a. This adjustment is anticipated, as customers who were previously served by facilities 2, 3, or 6 and located far from them (resulting in significantly higher costs if serviced by the same facilities with the increased value of C_D) are now reassigned to the closer facility 8.

Coded example

The code used to model and solve Example 9.5 is available here.

Packing problems 10

On a long journey even a straw weighs heavy.

Spanish Proverb

10.1 KPs 173
10.1.1 0-1 KP 173
10.1.2 Bounded KP 177
10.1.3 0-1 multiple KP 178
10.1.4 Other variants of the KP 179
10.2 BPPs 179
10.2.1 One-dimensional BPP 179
10.2.2 Two-dimensional horizontal BPP 181
10.2.3 Other variants of the BPP 188

This chapter deals with two foundational problems in OR: the Knapsack Problem (KP) and the Bin Packing Problem (BPP). The KP revolves around managing limited supply to accommodate varying demands, while the classic BPP aims to fulfill all demands while minimizing required supply. Although introduced briefly in Chapter 4 through examples, we now provide a comprehensive overview of these core problems and explore their basic versions along with extensions. Additionally, we justify their inclusion in Part IV, categorizing them as specialized assignment problems[1].

1: Such a categorization, while considered justified by the authors, might not be consistent with other references. For example, assignment problems are considered a special type of network problems in some references, such as Carter et al., 2018.

10.1 KPs

The KP derives its name from a representative analogy: envision having a knapsack (e.g., a backpack) with a fixed volume capacity, and a set of items, each with its own volume and value. Due to the limited space in the knapsack, not all items can be accommodated. The objective of the KP is to help us choose a subset of items that fit within the knapsack's volume constraints while maximizing the total value carried. Within this very general setting, several variants are possible, which we explain in the following sections.

10.1.1 0-1 KP

The 0-1 KP is the foundation of all KPs. It features a single knapsack of given capacity $\overline{W}$ and a set of items $\mathcal{I}$, where each item $i \in \mathcal{I}$ is characterized by a needed capacity W_i and a value V_i. Note that we stick with a generic definition of *capacity*, as it does not affect how the KP is defined. Capacity can be intended in terms of volume, weight, budget, etc. The goal is to map which items to select (via decision variable $x_i \in \{0, 1\}$, unitary if item i is placed in the knapsack). We report the needed notation for the 0-1 KP in Table 10.1.

We define the 0-1 KP as:

$$\max \sum_{i \in \mathcal{I}} V_i x_i \tag{10.1}$$

s.t.:

Table 10.1: Notation for the 0-1 KP.

Sets and indices	
$\mathcal{I}$	set of items $i \in \mathcal{I}$
Parameters	
$\overline{W}$	capacity of the knapsack
W_i	capacity needed by item $i \in \mathcal{I}$
V_i	value of item $i \in \mathcal{I}$
Variables	
$x_i \in \{0, 1\}$	unitary if item i is placed in the knapsack

$$\sum_{i \in \mathcal{I}} W_i x_i \leq \overline{W} \tag{10.2}$$

$$x_i \in \{0, 1\} \qquad \forall i \in \mathcal{I} \tag{10.3}$$

(10.1) maximizes the value carried in the knapsack, while (10.2) ensures that the carried items do not exceed the available capacity. (10.3) defines the nature of the decision variables.

We present the KP in a unique context compared to the typical standards of OR in Example 10.1.

Example 10.1 *A bounty hunter in the realm of Hilmor is entering an enchanted forest where they know 8 outlaws, ranging from humanoids to beasts, oozes, and undeads are hiding. Each outlaw comes with bounty money that the bounty hunter can collect if they defeat the outlaw in combat. Combats come with a price, as each outlaw takes away some life points from the hunter, who needs to save at least one life point (we leave a glorious death for another story). The hunter has at their disposal 16 life points (meaning that 15 can be spent defeating outlaws). The 8 outlaws are depicted as card games in Figure 10.1, where the top-left number inside the heart symbolizes the life points that the outlaw takes away during combat and the top-right number inside the coin represents the bounty money that is collected if such an opponent is defeated. Our task is to assist the bounty hunter so that they can maximize the money collected while remaining alive in the process.*

We acknowledge that this problem, despite its departure from the conventional framing, aligns with the KP because the value is represented by the bounty money and the capacity by the life points of the bounty hunter. A slight deviation from the original formulation is evident in (10.2). **Unlike a typical KP where the entire capacity is generally utilized, the bounty hunter must reserve one life point**. Thus, the right-hand side of Equation 10.2 in our model is $L - 1$, where L represents the total life points, considering that life points can only decrease in integer increments.

There is only one set, namely the set of outlaws $\mathcal{O}$. For every outlaw $o \in \mathcal{O}$, we define parameters C_o and L_o as, respectively, the coins received in bounty money and the life points lost if outlaw o is defeated. We report the information regarding the eight outlaws in Table 10.2.

Our KP becomes:



(a) Zorgoiln the Zombie. **(b)** Hermy the Hermit Crab. **(c)** Ghost of your past.

(d) Marion of the Haron. **(e)** Gerald the Gunk. **(f)** The Big Brown Bear.

(g) The Frog Prince. **(h)** The Mummy.

Figure 10.1: The eight outlaws of Example 10.1.

Outlaw	o	C_o	L_o
Zorgoiln the Zombie	1	5	2
Henry the Hermit Crab	2	17	5
Ghost of your past	3	15	4
Marion of the Haron	4	19	5
Gerald the Gunk	5	55	14
The Big Brown Bear	6	8	2
The Frog Prince	7	8	2
The Mummy	8	32	7

Table 10.2: Data pertaining to the eight outlaws of Example 10.1.

$$\max \sum_{o \in \mathcal{O}} C_o x_o \tag{10.4}$$

s.t.:

$$\sum_{o \in \mathcal{O}} L_o x_o \leq L - 1 \tag{10.5}$$

$$x_o \in \{0, 1\} \qquad \forall o \in \mathcal{O} \tag{10.6}$$

which can be expanded as:

$$\max \; 5x_1 + 17x_2 + 15x_3 + 15x_4 + 55x_5 + 8x_6 + 8x_7 + 32x_8 \tag{10.7}$$

s.t.:

$$2x_1 + 5x_2 + 4x_3 + 5x_4 + 14x_5 + 2x_6 + 2x_7 + 7x_8 \leq 15 \tag{10.8}$$

$$x_1, \cdots, x_8 \in \{0, 1\} \tag{10.9}$$

Solving the model results in $x_3 = x_6 = x_7 = x_8 = 1$, hence the bounty hunter should defeat the Ghost of your past, the Big Brown Bear, the Frog Prince, and the Mummy resulting in 15 life point lost and 63 gold coins accrued as bounty money. **Note that the bounty hunter uses all the allowed $L - 1$ life points in this case. In KPs it is usually the case that the whole capacity is used, if this helps increasing the objective value, but there might be occurrences where some capacity is left unused.**

We now proceed to showcase an alternative solution method which, while not proven to yield the optimal solution, does not rely on an BP mathematical model but on a more intuitive approach[2]. **This solution method is based on the intuition that an "ideal" outlaw provides a substantial bounty in gold coins while inflicting minimal damage to the bounty hunter's life points**. Even more, the ideal outlaw is the one characterized by the highest $B_o = \frac{C_o}{L_o}$ possible, as this KPI defines how much bounty money is obtained per each life point lost[3]. Hence, the bounty hunter could sort the outlaws by decreasing values of B_o. Then, the outlaw with the highest remaining B_o value is defeated if the remaining life points allow that or they are skipped. The process is continued until all available life points have been used or all outlaws have been "processed". This heuristic solution is showcased in Table 10.3, where outlaws are sorted by decreasing value of B_o. In **red** are marked outlaws who should have been defeated according to the B_o KPI but were skipped due to not enough life points available.

In the context of this example, we realize this sorting heuristic provides the optimal solution, as it suggests to defeat, in sequence, the Mummy, the Big Brown Bear, the Frog Prince, and the Ghost of your past. Note that in Table 10.3 we reported the outlaws after the Ghost of your past just for the sake of completeness. As the bounty hunter is left with just one

2: This solution method is a *heuristic*, i.e., an approach to solve an optimization problem that is not characterized by convergence proofs or optimality criteria, as opposed to the simplex method and the BB methods shown in Chapter 6 and Chapter 7.

3: In some literature (e.g., Carter et al., 2018) such a heuristic is labeled *bang for the buck*.

Outlaw	$\frac{C_o}{L_o}$	C_o	L_o	Life points remaining
The Mummy	4.57	32	7	9
The Big Brown Bear	4	8	2	7
The Frog Prince	4	8	2	5
Gerald the Gunk	**3.93**	**55**	**14**	**-9**
Marion of the Haron	**3.8**	**19**	**5**	**0**
Ghost of your past	3.75	15	4	1
Hermy the Hermit Crab	**3.4**	**17**	**5**	**-**
Zorgoiln the Zombie	**2.5**	**5**	**2**	**-**

Table 10.3: Solution to Example 10.1 using the sorting heuristic based on B_o values.

life point after having defeated the Ghost of your past and the minimum loss when defeating any outlaw is one life point, the bounty hunter is done after defeating the Ghost of your past anyway. In addition, as the Big Brown Bear and the Frog Prince are characterized by the same B_o, we arbitrarily chose to defeat the former first.

Coded example

The code used to model and solve Example 10.1 is available here.

0-1 KP as a serious game

A serious game based on the 0-1 KP can be found here. It entails three levels of increasing complexity and a fourth level based on the theory that will be unraveled in Chapter 14.

10.1.2 Bounded KP

The bounded KP provides a small twist to the original 0-1 KP introduced in Section 10.1.1 by not restricting each item $i \in \mathcal{I}$ to be "unique", but assuming N_i copies of it. In Example 10.1, this would be the case for the Big Brown Bear and the Frog Prince. Due to their identical values for L_o and C_o, both entities are equivalent within the scope of the KP. While readers might point out numerous distinctions between a brown bear and a frog prince, we defer this discussion for another occasion. Therefore, Example 10.1 can be reconsidered as a bounded KP comprising seven types of items, with six appearing individually and one appearing in duplicate. Aside from new parameters N_i, the nature of the decision variables changes as well, as now $x_i \in \{0, 1, \cdots, N_i\}$ is integer-valued. We report the needed notation for the bounded KP in Table 10.4.

Sets and indices	
$\mathcal{I}$	set of items $i \in \mathcal{I}$
Parameters	
$\overline{W}$	capacity of the knapsack
W_i	capacity needed by item $i \in \mathcal{I}$
V_i	value of item $i \in \mathcal{I}$
N_i	copies available of item $i \in \mathcal{I}$
Variables	
$x_i \in \{0, 1, \cdots, N_i\}$	number of copies of item i is placed in the knapsack

Table 10.4: Notation for the bounded KP.

We define the bounded KP as:

$$\max \sum_{i \in \mathcal{I}} V_i x_i \tag{10.10}$$

s.t.:

$$\sum_{i \in \mathcal{I}} W_i x_i \leq \overline{W} \tag{10.11}$$

$$x_i \in \{0, 1, \cdots, N_i\} \qquad \forall i \in \mathcal{I} \tag{10.12}$$

(10.10) maximizes the value carried in the knapsack, while (10.11) ensures that the carried items do not exceed the available capacity. (10.12) defines the integer nature of the decision variables and is the only tangible difference model-wise with respect to the original 0-1 KP of Section 10.1.1.

10.1.3 0-1 multiple KP

The 0-1 multiple KP provides a different twist to the original 0-1 KP by allowing multiple knapsacks to be used. Differently from the bounded KP from Section 10.1.2, each item is "unique", but we now need a second set $\mathcal{K}$ representing the knapsacks available, each with its own capacity $\overline{W}_k$. Because now there is more than a single knapsack, the type of decision changes as well. On top of deciding if an item should be transported or not, we also need to assign it to a knapsack. Hence, we define $x_{ik} \in \{0, 1\}$ to be unitary if item i is assigned to knapsack k. We report the needed notation for the 0-1 multiple KP in Table 10.5.

Table 10.5: Notation for the 0-1 multiple KP.

Sets and indices	
$\mathcal{I}$	set of items $i \in \mathcal{I}$
$\mathcal{K}$	set of knapsacks $k \in \mathcal{K}$
Parameters	
$\overline{W}_k$	capacity of knapsack $k \in \mathcal{K}$
W_i	capacity needed by item $i \in \mathcal{I}$
V_i	value of item $i \in \mathcal{I}$
Variables	
$x_{ik} \in \{0, 1\}$	unitary if item i is placed in knapsack k

We define the 0-1 multiple KP as:

$$\max \sum_{i \in \mathcal{I}} \sum_{k \in \mathcal{K}} V_i x_{ik} \tag{10.13}$$

s.t.:

$$\sum_{k \in \mathcal{K}} x_{ik} \leq 1 \qquad \forall i \in \mathcal{I} \qquad (10.14)$$

$$\sum_{i \in \mathcal{I}} W_i x_{ik} \leq \overline{W}_k \qquad \forall k \in \mathcal{K} \qquad (10.15)$$

$$x_{ik} \in \{0,1\} \qquad \forall i \in \mathcal{I}, k \in \mathcal{K} \qquad (10.16)$$

(10.13) maximizes the value carried across all knapsacks. (10.14) is a constraint set that was not needed in the previous KPs as they relied on a single knapsack. Now, we limit the assignment of each item to at most one knapsack. (10.15) ensures that the carried items in each knapsack $k \in \mathcal{K}$ do not exceed the available capacity $\overline{W}_k$. (10.16) defines the binary nature of the decision variables.

10.1.4 Other variants of the KP

Several variants of the KP exist in addition to the variants we presented in Section 10.1.2 and Section 10.1.3. We report in the following a couple of them and refer interested readers to Cacchiani et al., 2022a and Cacchiani et al., 2022b for extensive literature reviews on the topic.

- **multi-dimensional KP**. In this case, the capacity of the knapsack is not a scalar value but an n-dimensional vector $(\overline{W}_1, \cdots, \overline{W}_n)$. Following the same logic, each item requires n capacities, one per dimension. The goal is to maximize the value of the items transported while ensuring that the used capacity per dimension is not exceeded;
- **geometric KP**. In this case, the knapsack and each item are either a rectangle (two-dimensional case) or a parallelepiped (three-dimensional case). The goal is to maximize the value of the items that can inserted into the knapsack without exceeding its bounds.

10.2 BPPs

In a BPP, the foremost objective is to guarantee the packing of all considered items. This stands in stark contrast to the KP, where the central decisions revolve around item selection or exclusion. Conversely, in the BPP, the pivotal decision revolves around selecting the type and quantity of bins required to accommodate all items while minimizing associated packing costs. **Because of such a requirement, an important consideration is that there should always be a bin "large" enough to be able to contain the "largest" item in our set, otherwise our problem cannot be solved**. In analogy to the KP case, we present a couple of foundational models in the following sections. Both of them revolve around a set of items $\mathcal{I}$ and bins $\mathcal{B}$ as the main inputs.

10.2.1 One-dimensional BPP

In this version of the BPP, the capacity of both items $i \in \mathcal{I}$ and bins $b \in \mathcal{B}$ is one-dimensional (it could be thought of as weight or volume,

for example). We could even take one step further and consider different bin types (e.g., larger vs. smaller ones) by defining $\mathcal{T}$ the set of bin types. For each $t \in \mathcal{T}$, $\mathcal{B}_t$ is the subset of bins $b \in \mathcal{B}$ that are of type t. Having defined $\mathcal{T}$, we can define type-specific parameters such as $\overline{W}_t$ and C_t: the available capacity and cost of a bin of type t. Hence, every bin $b \in \mathcal{B}_t$ inherits the same parameters $\overline{W}_b = \overline{W}_t$ and $C_b = C_t$. It also follows that $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_2 \cup \cdots \cup \mathcal{B}_{|\mathcal{T}|}$, i.e., the set of bins is the union of the subsets containing bins of a specific type.

Each item must be packed into a single bin, yet the optimal assignment of items to bins is unknown beforehand. Therefore, adopting a highly risk-averse approach involves assuming that $|\mathcal{I}|$ bins of type t are available for each bin type $t \in \mathcal{T}$. With such an approach, we ensure the feasibility of the worst-case scenario where each item can only fit into the same bin type $\hat{t}$ and just by itself because of capacity restrictions. Therefore, all bins of type $\hat{t}$ are utilized, each containing a single item, while all other bins remain unused. Although uncommon, this risk-averse approach is the only method guaranteeing the feasibility of such a scenario. Before diving into the formulation, we need to define the decision variables needed. As our primary goal is to minimize the cost of the used bins, a decision variable mapping which bins are used is needed: $z_b \in \{0, 1\}$ which takes a unitary value if bin $b \in \mathcal{B}$ is used. In addition, the assignment of items to bins must also be tracked via $x_{ib} \in \{0, 1\}$ which takes a unitary value if item i is assigned to bin b. We report the needed notation for the one-dimensional BPP in Table 10.6.

Table 10.6: Notation for the one-dimensional BPP.

Sets and indices	
$\mathcal{I}$	set of items $i \in \mathcal{I}$
$\mathcal{T}$	set of bin types $t \in \mathcal{T}$
$\mathcal{B}$	set of bins $b \in \mathcal{B}$
$\mathcal{B}_t \subseteq \mathcal{B}$	set of bins of type $t \in \mathcal{T}$
Parameters	
$\overline{W}_b$	capacity of bin $b \in \mathcal{B}$
C_b	cost of bin $b \in \mathcal{B}$
W_i	capacity needed by item $i \in \mathcal{I}$
Variables	
$x_{ib} \in \{0, 1\}$	unitary if item i is placed in bin b
$z_b \in \{0, 1\}$	unitary if bin b is used

We define the one-dimensional BPP as:

$$\min \sum_{b \in \mathcal{B}} C_b z_b \tag{10.17}$$

s.t.:

$$\sum_{b \in \mathcal{B}} x_{ib} = 1 \qquad \forall i \in \mathcal{I} \qquad (10.18)$$

$$\sum_{i \in \mathcal{I}} W_i x_{ib} \leq \overline{W}_b \qquad \forall b \in \mathcal{B} \qquad (10.19)$$

$$\sum_{i \in \mathcal{I}} x_{ib} \leq |\mathcal{I}| z_b \qquad \forall b \in \mathcal{B} \qquad (10.20)$$

$$x_{ib} \in \{0,1\} \qquad \forall i \in \mathcal{I}, b \in \mathcal{B} \qquad (10.21)$$

$$z_b \in \{0,1\} \qquad \forall b \in \mathcal{B} \qquad (10.22)$$

(10.17) minimizes the cost of the used bins. (10.18) ensures that each item is assigned to exactly one bin, while (10.19) ensures that each bin is used within its capacity. (10.20) is a fixed-charge constraint (recall Section 4.8.4) that allows assigning items to bin b (potentially all the $|\mathcal{I}|$ items pending (10.19)) if z_b is unitary. A variant of such a constraint that achieves the same goal is discussed in the **An alternative version of constraint (10.20)** box. Finally, (10.21)-(10.22) define the nature of the decision variables.

Some readers might have noticed that set $\mathcal{T}$ does not appear anywhere in the formulation. While it appears implicitly in the definition of the properties of each bin $b \in \mathcal{B}$ (which inherits the properties of its bin type $t \in \mathcal{T}$), there is no explicit mention to it. This is not an oversight, because in the shown formulation we assume every item can be assigned to every bin. On the other hand, let us assume that every item i can only be packed in a subset of bin types $\mathcal{T}_i$. In such a case, (10.18) can be rewritten as

$$\sum_{t \in \mathcal{T}_i} \sum_{b \in \mathcal{B}_t} x_{ib} = 1 \qquad \forall i \in \mathcal{I} \qquad (10.23)$$

so that the contribution of $\mathcal{T}$ is apparent. Constraints (10.19)-(10.21) should be updated as well.

10.2.2 Two-dimensional horizontal BPP

This variant[4] of the BPP is defined on a horizontal (x, y) plane where both items $i \in \mathcal{I}$ and bins $b \in \mathcal{B}$ are rectangular-shaped. Each item i is defined by an original length L_i and width W_i: we assume that, originally, L_i is aligned with the x- and W_i with the y-direction. Items can generally be rotated by $\frac{\pi}{2}$ (hence aligning W_i with the x-direction and L_i with the y-direction), but for some of them such a rotation might not be allowed. Additionally, some items might only be packed in specific bin types, with set $\mathcal{T}$ defining the different bin types. Each bin type $t \in \mathcal{T}$ features a length $\overline{L}_t$ and a width $\overline{W}_t$: every bin $b \in \mathcal{B}_t \subseteq \mathcal{B}$ (subset of bins of type t) inherits such geometric properties. **Finally, some items might not be packed in the same bin. This last constraint has some practical implications. In air cargo operations, for example, packing in the same container vegetables and chemicals is generally not allowed as the latter might contaminate the former**. We define $\mathcal{B}_{inc}$ as the set of all item pairs (i, j) that are incompatible, i.e., that cannot be assigned to the same bin.

4: For the variant presented here, we took inspiration and adapted the three-dimensional BPP formulation from Paquay et al., 2016.

An alternative version of constraint (10.20)

Let us explain constraint (10.20) again. If $z_b = 1$, which implies an increment of C_b in the objective function, then $\sum_{i \in \mathcal{I}} x_{ib} \leq |\mathcal{I}|$. This means that, potentially, every item could be packed there, pending that the bin has enough capacity and every item can be packed there. If $z_b = 0$, then $\sum_{i \in \mathcal{I}} x_{ib} \leq 0$ prevents any item from being assigned to bin b (basically, bin b is unused).
An alternative way of expressing the same concept is the following

$$x_{ib} \leq z_b \qquad \forall i \in \mathcal{I}, b \in \mathcal{B} \tag{10.24}$$

which is defined for every item i and bin b combination and achieves the same goal, albeit with a slightly different nuance. (10.24) implies that bin b is labeled as used as soon as one item i is assigned to it and, conversely, no single item i can be assigned to bin b unless it is flagged as used. Constraint set (10.24) entails more constraints than the set (10.20) (namely $|\mathcal{I}| \times |\mathcal{B}|$ instead of $|\mathcal{B}|$). Notwithstanding, according to Postek et al., 2024 it provides a tighter linear relaxation (recall Chapters 7-8) and hence might be preferred for larger-scale problems.

To map the position of each item i, we define two continuous decision variables x_i and y_i that map the (x, y) location of the left-bottom vertex of the item. We then express the location of the right-front vertex $(x_i^{'}, y_i^{'})$ as

$$x_i^{'} = x_i + L_i(1 - r_i) + W_i r_i \tag{10.25}$$
$$y_i^{'} = y_i + L_i r_i + W_i(1 - r_i) \tag{10.26}$$

where $r_i \in \{0, 1\}$ is unitary if item $i \in \mathcal{I}$ is rotated by $\frac{\pi}{2}$ and zero if it retains its original orientation. We provide a visual interpretation of how such a rotation decision variable operates in Figure 10.2. We decide to place item i, with $L_i = 6$ and $W_i = 2$, with its left-bottom vertex in position $(x, y) = (0, 0)$. If we assume that no rotation occurs $(r_i = 0)$, then $x_i^{'} = x_i + L_i(1 - r_i) + W_i r_i = 0 + 6 \times 1 + 2 \times 0 = 6$ and $y_i^{'} = y_i + L_i r_i + W_i(1 - r_i) = 0 + 6 \times 0 + 2 \times 1 = 2$. This results in box i being located as the **green** box in Figure 10.2. Conversely, if we assume a $\frac{\pi}{2}$ rotation $(r_i = 1)$, we have $x_i^{'} = 0 + 6 \times 0 + 2 \times 1 = 2$ and $y_i^{'} = 0 + 6 \times 1 + 2 \times 0 = 6$, resulting in the **red** box in Figure 10.2.

Before diving into the notation and formulation, we highlight that this version of the BPP is based on specific geometric properties, one of which implies that items cannot overlap with each other. In fact, the available capacity of each bin $b \in \mathcal{B}$ is the available surface $L_b \times W_b$ it offers. We ensure that such a capacity is not exceeded by:

- ensuring each item i placed in bin b **does not exceed the boundary of the bin itself**;
- ensuring that all the items placed in the same bin b **do not overlap**.

To enforce the second requirement, we need decision variables $l_{ij} \in \{0, 1\}$ and $b_{ij} \in \{0, 1\}$ defined $\forall\, i, j \in \mathcal{I}$ (i.e., for every distinct pair or items).

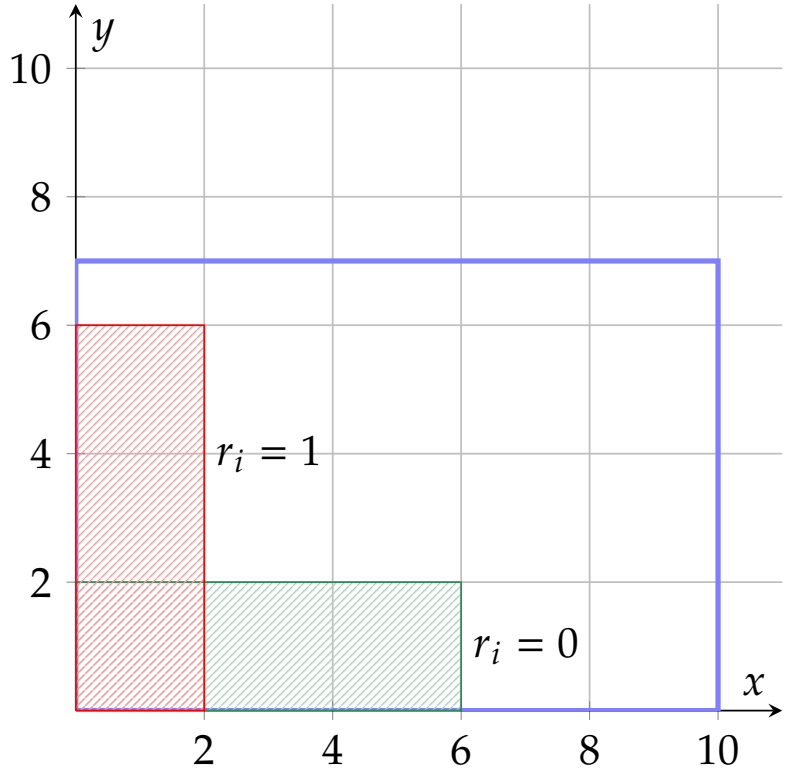


Figure 10.2: Example of the role of rotation decision variable r_i for an item with original length $L_i = 6$ and width $W_i = 2$, whose left-bottom vertex is placed in the origin $(0, 0)$. If $r_i = 0$, no rotation occurs, resulting in the **green** configuration. If $r_i = 1$, a rotation of $\frac{\pi}{2}$ occurs, resulting in the **red** configuration.

They are unitary if item i is completely to the left to (resp. behind) item j. We showcase a visual interpretation of the l_{ij} and b_{ij} decision variables in Figure 10.3. Focusing on item 1, we can write $l_{1,2} = 0, b_{1,2} = 1$ (item 1 is not to the left but behind item 2), $l_{1,3} = 1, b_{1,3} = 1$ (item 1 is to the left and behind item 3), and $l_{1,4} = 1, b_{1,4} = 0$ (item 1 is to the left but not behind item 4). Note that the term *completely* is key. For example, item 1 is slightly behind item 4, but not completely, as $y_1^{'} = 3 > y_4 = \frac{3}{2}$: basically, if we were to prolong the horizontal side of item 1 with $y = 3$, we would intersect item 4.

We report the needed notation for the two-dimensional horizontal BPP in Table 10.7. The mathematical formulation of the two-dimensional horizontal BPP is:

$$\min \sum_{b \in \mathcal{B}} C_b z_b \tag{10.27}$$

s.t.:

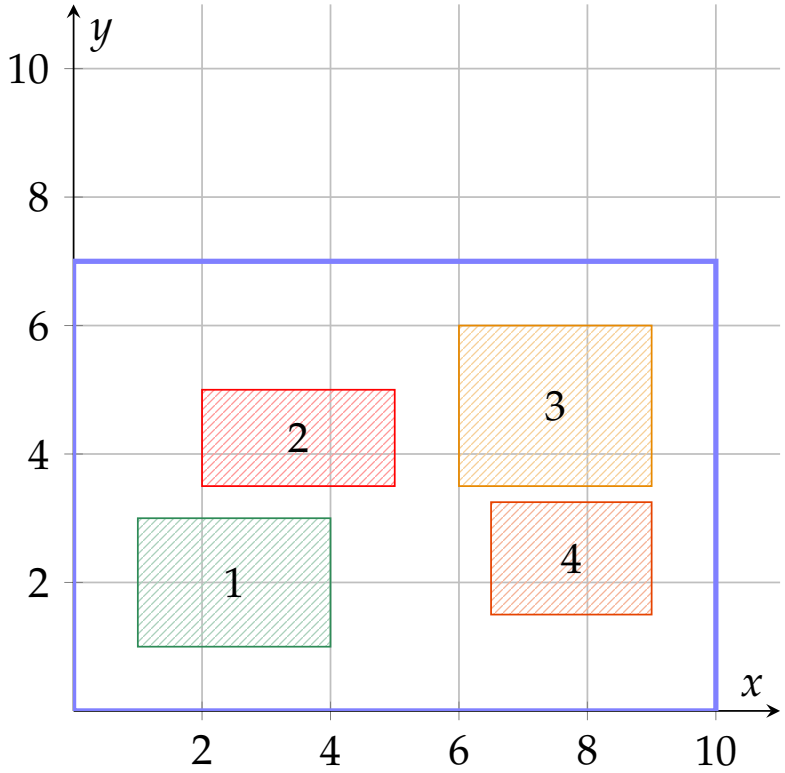


Figure 10.3: Example of the role of decision variables l_{ij} and b_{ij} in mapping the relative position between items i and j.

Table 10.7: Notation for the two-dimensional horizontal BPP.

Sets and indices	
$\mathcal{I}$	set of items $i, j \in \mathcal{I}$
$\mathcal{I}_{inc}$	set of incompatible items $i, j \in \mathcal{I}$
$\mathcal{T}$	set of bin types $t \in \mathcal{T}$
$\mathcal{B}$	set of bins $b \in \mathcal{B}$
$\overline{\mathcal{B}}_t$	subset of bins $b \in \mathcal{B}$ of type $t \in \mathcal{T}$
$\mathcal{T}_i$	subset of bin types $t \in \mathcal{T}$ where item $i \in \mathcal{I}$ can be placed
$\mathcal{B}_i$	subset of bins $b \in \mathcal{B}$ that can accommodate item $i \in \mathcal{I}$
Parameters	
C_b	cost of bin $b \in \mathcal{B}$
$\overline{L}_b$	length of bin $b \in \mathcal{B}$
$\overline{W}_b$	width of bin $b \in \mathcal{B}$
$\overline{L}_{max}$	$\max_{b \in \mathcal{B}} \left\{\overline{L}_b\right\}$: maximum length across all bins
$\overline{W}_{max}$	$\max_{b \in \mathcal{B}} \left\{\overline{W}_b\right\}$: maximum width across all bins
L_i	length of item $i \in \mathcal{I}$
W_i	width of item $i \in \mathcal{I}$
Variables	
$x_i \in \mathbb{R}_0$	x-position of the left-bottom vertex of item i
$y_i \in \mathbb{R}_0$	y-position of the left-bottom vertex of item i
$r_i \in \{0,1\}$	unitary if item $i \in \mathcal{I}$ is rotated by $\frac{\pi}{2}$
$l_{ij} \in \{0,1\}$	unitary if item i is to the left of item j
$b_{ij} \in \{0,1\}$	unitary if item i is behind item j
$p_{ib} \in \{0,1\}$	unitary if item i is placed in bin b
$z_b \in \{0,1\}$	unitary if bin b is used

$$l_{ij} + l_{ji} + b_{ij} + b_{ji} \geq p_{ib} + p_{jb} - 1 \qquad \forall i, j \in \mathcal{I}, b \in \mathcal{B}_i \cap \mathcal{B}_j \qquad (10.28)$$

$$x_j \geq x_i + L_i(1 - r_i) + W_i r_i - \overline{L}_{max}(1 - l_{ij}) \qquad \forall i, j \in \mathcal{I} \qquad (10.29)$$

$$x_j + \epsilon \leq x_i + L_i(1 - r_i) + W_i r_i + \overline{L}_{max} l_{ij} \qquad \forall i, j \in \mathcal{I} \qquad (10.30)$$

$$y_j \geq y_i + L_i r_i + W_i(1 - r_i) - \overline{H}_{max}(1 - b_{ij}) \qquad \forall i, j \in \mathcal{I} \qquad (10.31)$$

$$y_j + \epsilon \leq y_i + L_i r_i + W_i(1 - r_i) + \overline{L}_{max} b_{ij} \qquad \forall i, j \in \mathcal{I} \qquad (10.32)$$

$$x_i + L_i(1 - r_i) + W_i r_i \leq \sum_{b \in \mathcal{B}_i} \overline{L}_b p_{ib} \qquad \forall i \in \mathcal{I} \qquad (10.33)$$

$$y_i + L_i r_i + W_i(1 - r_i) \leq \sum_{b \in \mathcal{B}_i} \overline{W}_b p_{ib} \qquad \forall i \in \mathcal{I} \qquad (10.34)$$

$$\sum_{b \in \mathcal{B}_i} p_{ib} = 1 \qquad \forall i \in \mathcal{I} \qquad (10.35)$$

$$p_{ib} \leq z_b \qquad \forall i \in \mathcal{I}, b \in \mathcal{B}_i \qquad (10.36)$$

$$p_{ib} + p_{jb} \leq 1 \qquad \forall i, j \in \mathcal{I}_{inc}, b \in \mathcal{B}_i \cap \mathcal{B}_j \qquad (10.37)$$

$$z_b \leq z_{b-1} \qquad \forall t \in \mathcal{T}, b \in \overline{\mathcal{B}}_t \setminus \{1\} \qquad (10.38)$$

$$x_i \in \mathbb{R}_0 \qquad \forall i \in \mathcal{I} \qquad (10.39)$$

$$y_i \in \mathbb{R}_0 \qquad \forall i \in \mathcal{I} \qquad (10.40)$$

$$r_i \in \{0, 1\} \qquad \forall i \in \mathcal{I} \qquad (10.41)$$

$$l_{ij}, b_{ij} \in \{0, 1\} \qquad \forall i, j \in \mathcal{I} \qquad (10.42)$$

$$p_{ib} \in \{0, 1\} \qquad \forall i \in \mathcal{I}, b \in \mathcal{B}_i \qquad (10.43)$$

$$z_b \in \{0, 1\} \qquad \forall b \in \mathcal{B} \qquad (10.44)$$

(10.27) aims at minimizing the overall cost associated with the used bins. (10.28) enforces that if two items are placed in the same bin (the right-hand side is unitary), then at least one of the four decision variables on the left-hand side should be unitary. This ensures that the two items cannot overlap. Constraints (10.29)-(10.30) are either-or constraints that force $x_i^{'}$ (recall (10.25)) to be smaller or equal to x_j if $l_{ij} = 1$, hence correctly enforcing the left-right relationship between items i and j. ϵ is a small number used to avoid ambiguity when $x_j = x_i^{'}$, as in such a case both $l_{ij} = 0$ and $l_{ij} = 1$ would satisfy the inequality. By introducing ϵ, we force $l_{ij} = 1$ when $x_j = x_i^{'}$. Constraints (10.31)-(10.32) are the equivalent counterpart along the y-direction. (10.33) ensures that the right side of item i is within the horizontal extension of the bin where it is placed, and (10.34) is the counterpart for the y-direction. (10.35) ensures that every item i is assigned to one of the bins $b \in \mathcal{B}_i$ that can allocate such an item. Constraints (10.36) ensure that as soon as an item is assigned to a bin that bin is active (here we used the version of the constraint explained in the **An alternative version of constraint (10.20)** box from Section 10.2.1), while constraints (10.37) ensure that incompatible items are not assigned to the same bin. With $b \in \mathcal{B}_i \cap \mathcal{B}_j$ we ensure that the constraint is imposed only for the bins that can accommodate both items i and j

(we followed the same logic in (10.28)). (10.38) is a symmetry-breaking constraint that does not affect the optimal solution, but limits symmetries by enforcing that bins of each type $t \in \mathcal{T}$ are used in increasing sequence. With $\forall t \in \mathcal{T}, b \in \mathcal{B}_t \setminus \{1\}$ we mean that such a constraint should be imposed for every bin of a specific type apart from the first one of that type (whose index might be different than 1 though). Let us consider an example where $\mathcal{T} = \{1, 2\}$, $\overline{\mathcal{B}}_1 = \{1, 2, 3\}$, and $\overline{\mathcal{B}}_2 = \{4, 5, 6\}$. Constraint (10.38) enforces that $b_2 \leq b_1$ and $b_3 \leq b_2$ for $t = 1$ and $b_5 \leq b_4$ and $b_6 \leq b_5$ for $t = 2$. This allows the model to immediately use bin 1 (the first one of type 1) and bin 4 (the first one of type 2) and only later (and in sequence) the others. Finally, (10.39)-(10.44) define the nature of the decision variables.

We show an application example of this two-dimensional horizontal BPP in Example 10.2.

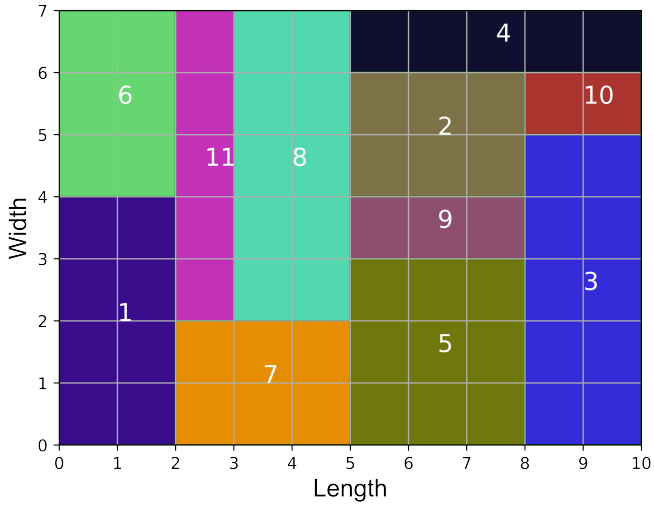
Example 10.2 *A farming company is considering acquiring land to cultivate various fruit and vegetable crops. In the designated area, two types of lots are available: two larger fields measuring* $1,000$ *meters long and* 700 *meters wide, and two smaller ones measuring* 800 *meters long and* 500 *meters wide. The cost of purchasing a large or small field is* $1,000,000$ *and* $800,000$€*, respectively. We define* $\mathcal{T} = \{1, 2\}$ *the set of field types, with 1 representing the large one and 2 the small one. The company aims to cultivate 16 different products* $i \in \mathcal{I}$*, each requiring a specific land area. Certain products can be grown in either field type due to soil specifications (*$\mathcal{T}_i$ *represents the allowed field types for crop* i*). However, some crops cannot be planted together due to the risk of attracting harmful insects or cannot be rotated with respect to the original orientation because of proper sun exposure. We report all the characteristics of the sixteen crops in Table 10.8 (dimensions have been divided by* 100 *for the sake of simplicity). The company is striving to formulate a mathematical model that can determine the most cost-effective solution, detailing which fields to acquire and how to arrange the various crops to fulfill all constraints effectively.*

Table 10.8: Data pertaining to the 16 crops of Example 10.2.

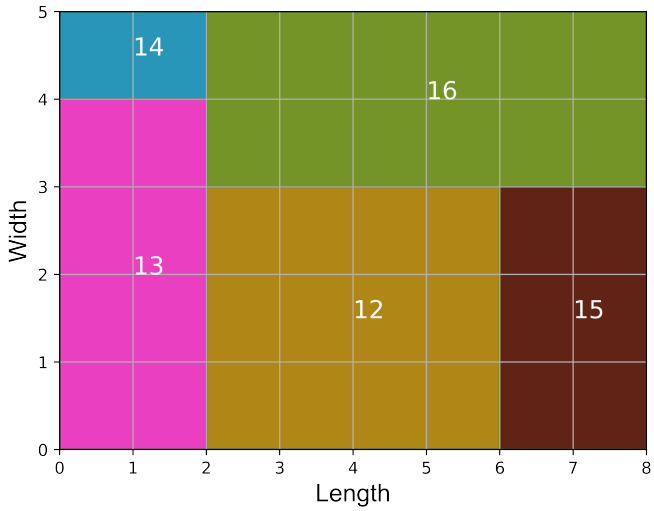
Crop	i	L_i	W_i	$\mathcal{T}_i$	Rotation	Incomp.
Strawberries	1	4	2	1,2	Y	12,15
Comice pears	2	2	3	1,2	Y	12,16
Fuji apples	3	2	5	1	N	
Eggplant	4	5	1	1	Y	14
Kabocha squash	5	3	3	1,2	Y	-
Orri mandarines	6	3	2	1,2	Y	-
Navel oranges	7	2	3	1,2	Y	13
Iceberg lettuce	8	5	2	1,2	Y	14
Raspberries	9	3	1	1	N	-
Kale	10	2	1	1,2	Y	-
Chioggia radicchio	11	5	1	1,2	Y	15
Shiitake mushrooms	12	4	3	2	N	2
White asparagus	13	4	2	1,2	Y	3
Blueberries	14	2	1	1,2	Y	4
Jintao kiwi	15	2	3	1,2	Y	5
Cantaloupe melon	16	2	6	1,2	Y	2,3

We realize that this problem can be interpreted as a two-dimensional horizontal BPP, where the products are the items $i \in \mathcal{I}$ and the fields are the bins $b \in \mathcal{B}$. We can define $\mathcal{I} = \{1, \cdots, 16\}$ and $\mathcal{B} = \{1, \cdots, 4\}$. Recalling the definition of the two field types (i.e., of the two bin types), we get $\overline{\mathcal{B}}_1 = \{1, 2\}$ and $\overline{\mathcal{B}}_2 = \{3, 4\}$. In this case, making the model explicit would be tedious and require quite a few pages, hence we refrain from doing that. We directly display the solution in Figure 10.4 and

critically discuss it.



(a) Large field (bin $b = 1$ of type $t = 1$).



(b) Small field (bin $b = 3$ of type $t = 2$).

Figure 10.4: The final solution in terms of fields purchased and location of crops to Example 10.2.

The optimal solution indicates that just two fields, one large and one small, are adequate to accommodate all the crops for an overall investment of 1,800,000€. It is worth noting that this scenario perfectly matches the required land for the 16 products. While some empty spaces are permissible in BPPs, we deliberately crafted this example to fully utilize the two fields for enhanced visualization[5]. In Figure 10.4a and Figure 10.4b the location and index of the cultivated crops are shown. Furthermore, crops that cannot be rotated maintain their original length and width alignment with the x- and y-coordinates, respectively, while incompatible crops are assigned to different fields. **Finally, the symmetry-breaking constraints, while not contributing to improving the optimal solution, ensured that the first bin available per type ($b = 1$ and $b = 3$, respectively) was selected**. In larger-scale models, such symmetry-breaking constraints might contribute quite significantly in reducing the computational time.

Another important consideration still addresses symmetry. **While we ensured to avoid symmetrical solutions in terms of which bins are used, the issue is more complex when it comes to relative positioning of items inside a bin**. Let us consider, for example, Figure 10.4a. An

5: We use this example to display that this particular version of the two-dimensional horizontal BPP can be interpreted as a *cutting stock problem*. In a cutting stock problem, rectangular sheets of paper or metal (or any other material) must be cut into smaller rectangles according to specific orders by customers. The goal of the problem is to find a minimum-waste solution where all necessary orders are cut from some sheets while the unused (i.e., wasted) material per sheet is minimized. Our example translates into an "ideal" cutting stock problem where no waste at all is produced. We refer interested readers to this Wikipedia page for more information.

equally optimal solution would be achieved if crops 11 and 8, or 6 and 1, or 2, 9, and 5 were shuffled. Many other reshufflings are possible. Furthermore, a rotation of π of each bin (i.e., crop 1 would be placed in the top-right and crop 4 in the lower-left corner of bin $b = 1$) would result in another equally optimal solution.

Coded example

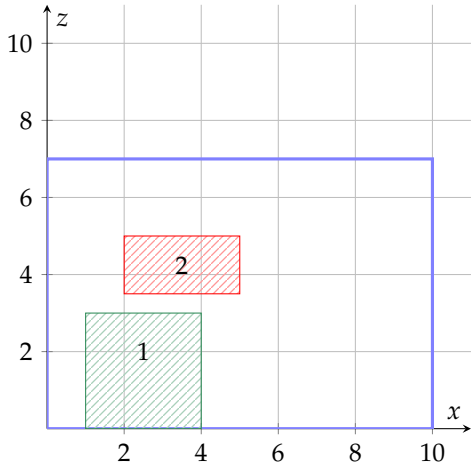
The code used to model and solve Example 10.2 is available here.

10.2.3 Other variants of the BPP

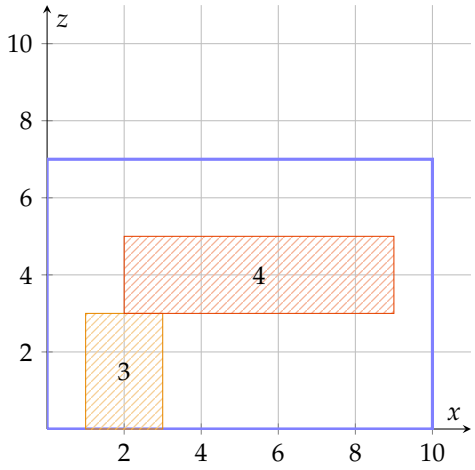
Regarding variants and extensions of the BPP, an initial consideration involves an increase in dimensionality. Having examined one- and two-dimensional versions, the logical progression leads to exploring a three-dimensional BPP. In Paquay et al., 2016, a formulation of such an extension is presented, primarily focusing on air cargo operations and the palletization of cargo items into containers for air transport.

A significant complication of transitioning to a fully three-dimensional BPP is the consideration of gravity. In our two-dimensional horizontal example (depicted in Figure 10.4), the absence of empty spaces between items posed no issues; however, in a three-dimensional scenario, the presence of empty spaces horizontally can still be accommodated. Yet, empty spaces vertically are not feasible as items cannot levitate. Hence, items must either rest on the ground or have their lower side vertices properly supported by other items, typically at least three out of four vertices according to Paquay et al., 2016 (although having just two vertices supported might be enough depending on considerations on the position of the center of gravity). We showcase some illustrative examples regarding vertical stability in a three-dimensional BPP in Figure 10.5

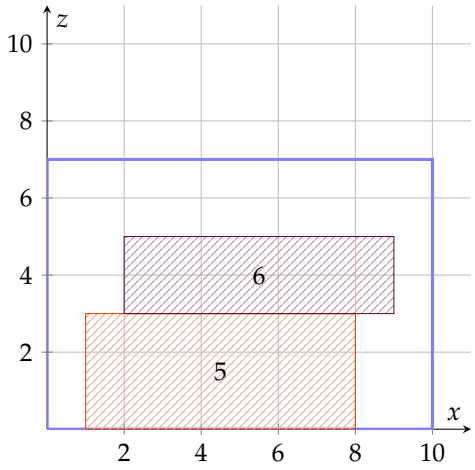
In addition to the three-dimensional extension, **bins with irregular shapes** represent another variant in BPPs. Because of the linear requirements of the objective function and constraints, the level of irregularity of the bin shape cannot be extreme. In Paquay et al., 2016, bins with some of their corner being cut are presented. This approach is adopted to simulate Unit Load Device (ULD)s utilized in air cargo operations, especially containers (see Figure 10.6). This cutting technique is essential to enable the container to conform more closely to the fuselage shape of the aircraft, thereby enhancing the transportable volume. Since the cut is represented as a line equation, only minor geometric adjustments are made to the three-dimensional BPP.



(a) Item 2 is floating. Hence, none of its four lower-side vertices are supported.



(b) Item 4 partially lies on item 3, but not enough to be stable as its center of gravity's projection falls outside item 3.



(c) Item 6 partially lies on item 5 and its center of gravity's projection falls inside item 5, hence suggesting a stable configuration.

Figure 10.5: Different examples of infeasible, unstable, and stable configurations in a three dimensional BPP. In all three figures, a frontal view of the (x, z) plane is depicted.



Figure 10.6: Several containers in front of an aircraft.

For a comprehensive exploration of two-dimensional BPPs handling irregular shapes of items and bins, along with tailored solution methodologies, we recommend referring to Guo et al., 2022. Additionally, it is worth mentioning another variant of the BPP known as the **online** BPP. Many real-world applications of BPPs revolve around logistical processes where items are not delivered all at once but rather scattered over time. In such scenarios, it may be more prudent to pack the items as they become available rather than waiting for the entire batch, even though waiting could potentially lead to a better solution. Therefore, making decisions regarding what to pack and when to wait is crucial for improving logistical processes. This stands in contrast to offline packing, where all information is known in advance (refer to Ali et al., 2022 for a literature review comparing the two approaches).

Part V

Networks

An introduction to graph theory | 11

We are all now connected by the Internet, like neurons in a giant brain.

Stephen Hawking

11.1 Definition of a graph . . 193
11.2 Properties of a graph . . 195
11.3 From “abstract" graphs to “concrete" networks . . . 203

Graph theory is a mathematical branch devoted to studying graphs, i.e., mathematical structures that define connections between elements. This branch of mathematics encompasses a wide array of domains such as sociology, linguistics, and computer science, as well as numerous applications like social networks and transportation networks. A comprehensive grasp of graph theory serves as a robust foundation for understanding many of the OR topics addressed in this book, particularly in Part V. This motivated us to include this chapter in the book. It is worth noting that definitions of what constitutes a graph and the distinction between a graph and a network can vary significantly across literature. While we have added our own perspective and customization to this chapter, our aim is not to completely supplant seminal books or other references on the topic. For readers interested in expanding their knowledge, we recommend for example Trudeau, 2013.

11.1 Definition of a graph

A graph $G = (\mathcal{V}, \mathcal{E})$ is an object consisting of two sets, namely the set of vertices $\mathcal{V}$ and the set of edges $\mathcal{E}$[1]. **While $\mathcal{E}$ can potentially be empty (although we will see this option is not interesting for our purposes), $\mathcal{V}$ must contain at least one element. Each edge $e \in \mathcal{E}$ is a 2-tuple**[2] **(v_1, v_2) where v_1 and $v_2 \in \mathcal{V}$ and $v_1 \neq v_2$: $\mathcal{E} \subseteq \{(v_1, v_2) | v_1, v_2 \in \mathcal{V} \wedge v_1 \neq v_2\}$.** If (v_1, v_2) is an edge of a graph, then such an edge *connects* the vertices v_1 and v_2 (in Section 11.2 we will elaborate on this) and is *incident* to both v_1 and v_2.

Note that, so far, we have not committed to any specific application domain for a graph. A first graph $G_1 = (\{A, B, C\}, \{(A, B), (A, C)\})$ and a second graph $G_2 = (\{1, 2, 3\}, \{(1, 2), (1, 3)\})$, respectively dealing with alphabet letters and numbers, are both graphs according to our definition. In Figure 11.1 and Figure 11.2 we show a possible representation of G_1 and G_2. We want to stress the importance of the term *possible*, as no more bounding indication has been provided regarding the nature of the two graphs. Hence, the fact that in G_1 the three vertices are positioned in a triangular fashion while they are linear in G_2 is arbitrary. The same applies to edges being straight lines in G_1 and sloped in G_2.

Conversely, if an object with vertices and edges fails to meet all of the conditions necessary for it to be labeled as a graph, it cannot be classified as a graph. We highlight three examples in Figure 11.3. In Figure 11.3a, an edge connects vertex A with itself. This is not allowed because $v_1 \neq v_2$ must hold, and hence an edge should connect two distinct vertices. In Figure 11.3b, an edge connects vertex A with nothing.

1: In the literature, this book included, the terms *vertex* and *node* might be used interchangeably. Along a similar note, the terms *edge*, *arc*, and *link* might be used interchangeably.

2: A 2-tuple is an ordered set of 2 elements. In general, a n-tuple is an ordered set of n elements. See this Wikipedia page for more information.

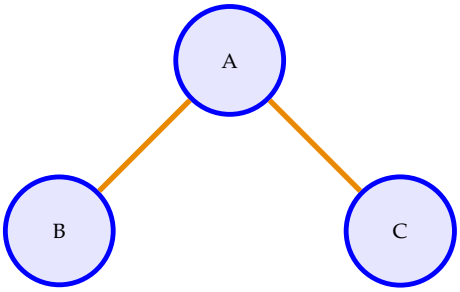


Figure 11.1: A graphical representation of G_1.

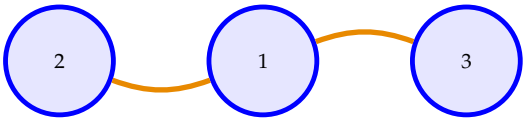
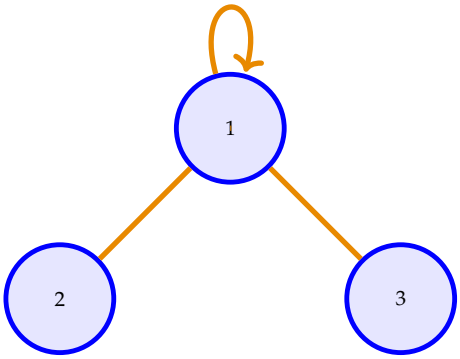
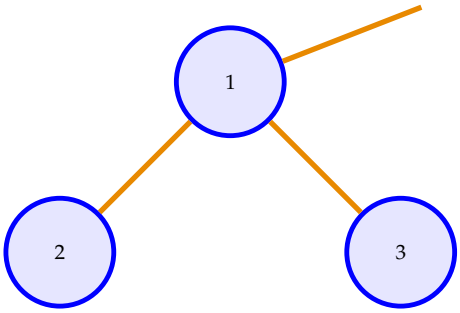


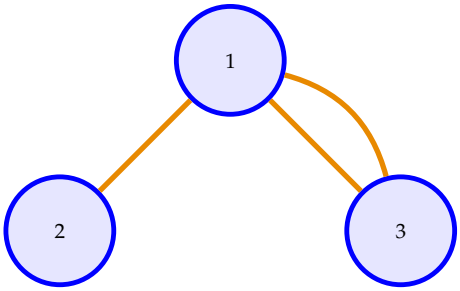
Figure 11.2: A graphical representation of G_2.



(a) Example of a “non-graph" due to an edge connecting a vertex with itself.



(b) Example of a “non-graph" due to an edge connecting a vertex with nothing.



(c) Example of a “non-graph" due to two edges connecting the same pair of vertices.

Figure 11.3: Examples of objects that closely resemble a graph, but that are not a graph due to failing to satisfy one necessary condition.

This is not allowed because $(v_1, v_2) \mid v_1, v_2 \in \mathcal{V}$ must hold, and hence an edge represents a connection between two vertices part of $\mathcal{V}$. Finally, in Figure 11.3c two edges connect the same vertex pair. Hence, set $\mathcal{E}$ would contain twice element (v_1, v_2): $\mathcal{E} = \{\cdots, (v_1, v_2), (v_1, v_2), \cdots\}$. Because a set is a collection of distinct items, this is not allowed either[3].

3: Multiple edges connecting the same (v_1, v_2) pair are not allowed in a graph. Notwithstanding, they are allowed if an extension of a graph, called *multigraph*, is considered.

11.2 Properties of a graph

Fully enumerating all the possible properties that might characterize a graph is a daunting task extending well and beyond the scope of this chapter. We renew our suggestion to consult Trudeau, 2013 for a thorough treatment of the topic. Here, we limit ourselves to the properties that are more relevant within the OR topics of interest.

One of the most important features of every graph is whether edges are characterized by directionality or not. The latter case identifies an **undirected** graph, the former a **directed**[4] graph.

4: In some references, directed graphs are generally abbreviated into *digraphs*.

Definition 11.1 An undirected graph is a graph where the set of edges $\mathcal{E} \subseteq \{(v_1, v_2) \mid v_1, v_2 \in \mathcal{V} \wedge v_1 \neq v_2\}$ contains **unordered pairs**. Hence, edge (v_1, v_2) is equivalent to edge (v_2, v_1) and only one of the two should be stored in $\mathcal{E}$. This also implies that **an edge in an undirected graph can be transversed in both directions**.

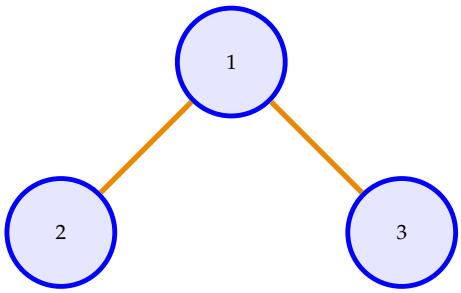
Definition 11.2 A directed graph is a graph where the set of edges $\mathcal{E} \subseteq \{(v_1, v_2) \mid v_1, v_2 \in \mathcal{V} \wedge v_1 \neq v_2\}$ contains **ordered pairs**. Hence, edge (v_1, v_2) is not equivalent to edge (v_2, v_1). This also implies that **an edge in a directed graph can only be transversed following the sequence of vertices that define it**.

In general, edges of undirected graphs are visualized without any arrow tips (rather than with an arrow per side), while edge (v_1, v_2) of a directed graph is visualized with an arrow tip pointing towards vertex v_2. Figure 11.4 emphasizes the contrasting visualizations of an undirected graph G_u (Figure 11.4a) and a directed one G_d (Figure 11.4b). In the figure, we assumed that every edge of G_u was duplicated into the two corresponding directed edges of G_d to maintain consistency. However, G_d would still be a valid directed graph even if any of the four edges in Figure 11.4b were removed.

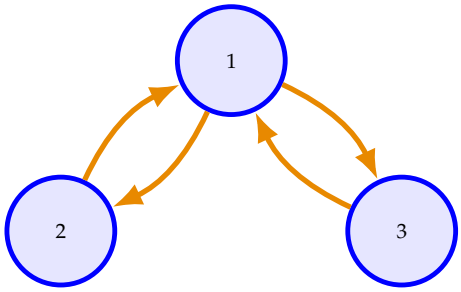
This final statement lays the groundwork for introducing the concept of a **subgraph**.

Definition 11.3 Given two graphs $G_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $G_2 = (\mathcal{V}_2, \mathcal{E}_2)$, G_2 is a subgraph of G_1 if $\mathcal{V}_2 \subseteq \mathcal{V}_1$ and $\mathcal{E}_2 \subseteq \mathcal{E}_1$, i.e., **if the set of vertices of G_2 is a subset of the set of vertices of G_1 and if the set of edges of G_2 is a subset of the set of edges of G_1**.

We showcase a directed graph G_1 and one of its potential subgraphs G_2 in Figure 11.5. In Figure 11.5b, we left in light gray the original vertices and edges of G_1 that are instead absent in G_2. It is crucial to note that any subgraph must still adhere to the fundamental properties of a graph mentioned earlier. In line with this, when a vertex is removed from a

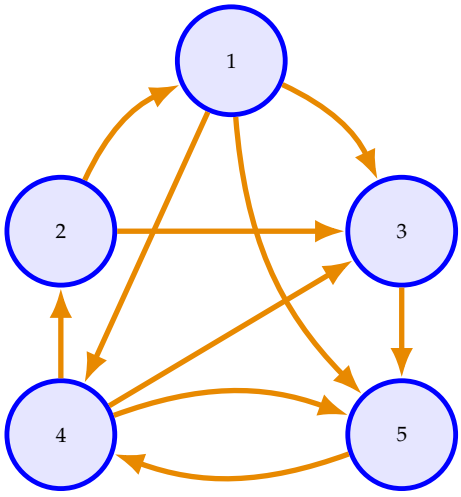


(a) Undirected graph.

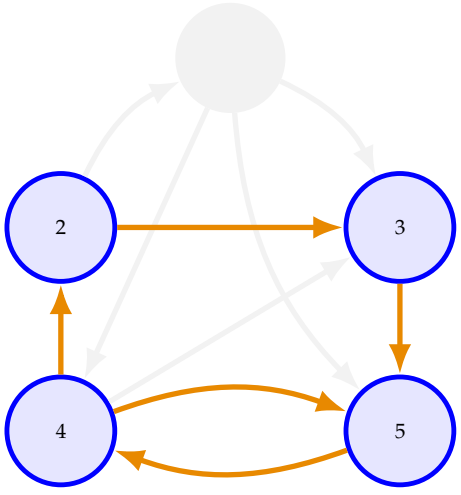


(b) Directed graph.

Figure 11.4: Example of undirected graph and its transformation into a directed graph. Note: we assumed that every (v_1, v_2) edge of the undirected version was replaced by both edge (v_1, v_2) and edge (v_2, v_1) in the directed version.



(a) G_1.



(b) G_2.

Figure 11.5: A directed graph G_1 and a subgraph G_2 of G_1. In Figure 11.5b we highlight with light gray the vertices and edges that were removed from G_1 to obtain subgraph G_2.

graph, every edge incident to that vertex must also be removed to prevent situations like the one depicted in Figure 11.3b.

While G_1 is characterized by $|\mathcal{V}| = 5$ vertices and $|\mathcal{E}| = 10$ edges, its subgraph G_2 has $|\mathcal{V}| = 4$ vertices and $|\mathcal{E}| = 5$ edges.

For both undirected and directed graphs, the number and "location" of the edges plays a role of paramount importance in determining the *connectivity* properties of the graph. Connectivity is an umbrella term related to questions such as *"Can any vertex be reached from any other vertex in the graph?"* or *"What is the minimum number of edges to be transversed moving from vertex v_1 to vertex v_2?"*. A first key definition pertaining to graph connectivity is **graph completeness**.

Definition 11.4 A complete graph is defined by having the maximum possible number of edges given the number of vertices $|\mathcal{V}|$. A complete undirected graph is characterized by $\frac{|\mathcal{V}|(|\mathcal{V}|-1)}{2}$ edges, a complete directed graph by $|\mathcal{V}|(|\mathcal{V}|-1)$ edges.

Definition 11.4 stems from the following consideration. In a complete undirected graph with $\mathcal{V} = \{v_1, v_2, \cdots, v_{|\mathcal{V}|}\}$, vertex v_1 must be connected to $v_2, v_3, \cdots, v_{|\mathcal{V}|}$, for a total of $|\mathcal{V}| - 1$ edges. Then, vertex v_2 must be connected to $v_3, v_4, \cdots, v_{|\mathcal{V}|}$ (not to v_1 as edge (v_1, v_2) already exists). Following the same logic, vertex $v_{|\mathcal{V}|-1}$ only connects to $v_{|\mathcal{V}|}$ while vertex $v_{|\mathcal{V}|}$ does not need any additional edge because all of them have already been defined. The overall number of defined edges is $(|\mathcal{V}|-1) + (|\mathcal{V}|-2) + (|\mathcal{V}|-3) + \cdots + 1 = \sum_{i=1}^{|\mathcal{V}|-1} i = \frac{|\mathcal{V}|(|\mathcal{V}|-1)}{2}$. For a directed graph, we can use the insight from Figure 11.4 and recognize that each edge in an undirected graph translates into two edges in the directed counterpart. Hence, in a complete directed graph, the number of edges in $|\mathcal{V}|(|\mathcal{V}|-1)$.

Because not every graph is complete, a related definition is graph **density**.

Definition 11.5 The density D of a graph $G = (\mathcal{V}, \mathcal{E})$ is the ratio between the number of edges characterizing G and the maximum number of edges that G could have (i.e., if G was complete). For an undirected graph, we have

$$D = \frac{|\mathcal{E}|2}{|\mathcal{V}|(|\mathcal{V}|-1)} \tag{11.1}$$

while for a directed graph we have

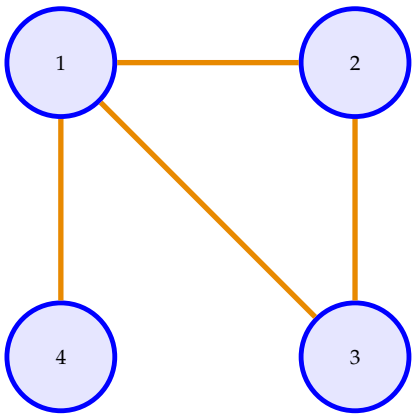
$$D = \frac{|\mathcal{E}|}{|\mathcal{V}|(|\mathcal{V}|-1)} \tag{11.2}$$

Note the subtle difference at the numerator of (11.1) and (11.2). Given an undirected and directed graph with the same number of vertices $|\mathcal{V}|$ and edges $|\mathcal{E}|$, the density of the undirected one is twice as large as the directed one. This is coherent with the fact that a complete undirected graph has half the number of edges as a complete directed graph with the same number of vertices.

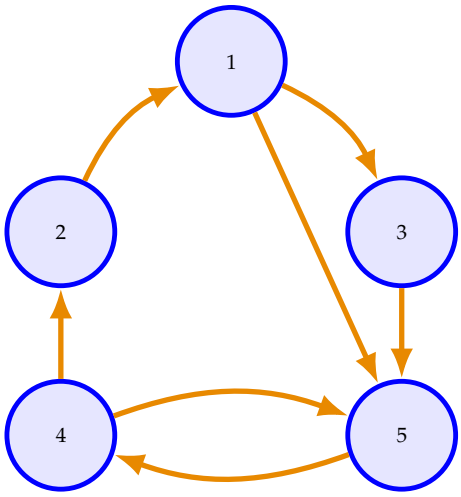
A compact and useful representation of the connectivity properties of a graph is the **adjacency matrix**.

Definition 11.6 For a graph G with $|\mathcal{V}|$ vertices, the adjancency matrix A is an $(|\mathcal{V}|, |\mathcal{V}|)$ binary matrix where A_{ij} is unitary if there exists an edge connecting vertices i and j and 0 otherwise. **A is symmetrical for undirected graphs because of the lack of directionality of edges, while it is not bound to be symmetrical for directed graphs**.

Example 11.1 *Given the two graphs of Figure 11.6, define the associated adjacency matrices.*



(a) G_1.



(b) G_2.

Figure 11.6: An undirected graph G_1 and directed graph G_2 used in Example 11.1.

Let us start with G_1. Because $|\mathcal{V}| = 4$, then A_{G_1} is a $(4, 4)$ matrix. Because there are 4 edges in Figure 11.6a, we expect to have eight 1s in A_{G_1} as elements (i, j) and (j, i) are equivalent due to the lack of directionality[5]. The set of edges is $\mathcal{E} = \{(1, 2), (1, 3), (1, 4), (2, 3)\}$, hence

$$A_{G_1} = \begin{matrix} \\ i=1 \\ i=2 \\ i=3 \\ i=4 \end{matrix} \overset{\begin{matrix} j=1 & j=2 & j=3 & j=4 \end{matrix}}{\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}} \tag{11.3}$$

5: As a matter of fact, for undirected graphs we do not need to fill a full adjacency matrix, but only the upper (or lower) triangular part due to symmetry.

where, to enhance clarity, the indices of rows and columns in Equation 11.3 are specified to reflect the vertices names in Figure 11.6a.

For G_2, we expect A_{G_2} to be a $(5,5)$ matrix as $|\mathcal{V}| = 5$. The set of edges is $\mathcal{E} = \{(1,3),(1,5),(2,1),(3,5),(4,2),(4,5),(5,4)\}$, with $|\mathcal{E}| = 7$. If we count the number of edges in Figure 11.6b, we obtain 7 edges. This highlights consistency. The resulting adjacency matrix is

$$A_{G_2} = \begin{array}{c} \begin{matrix} j{=}1 & j{=}2 & j{=}3 & j{=}4 & j{=}5 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} i=1 \\ i=2 \\ i=3 \\ i=4 \\ i=5 \end{matrix} \tag{11.4}$$

In both (11.3) and (11.4) the diagonal is filled with 0s. This is correct and a recommended check anytime an adjacency matrix is defined. If some unitary elements were present, they would define an edge of the (i,i) type, hence a loop: **"regular" graphs do not feature loops** (recall Figure 11.3a).

It is essential to note that both Figure 11.6a and matrix (11.3) convey identical information about G_1, albeit presented in different formats. While Figure 11.6a may offer visual appeal and flexibility for additional enhancements, both representations are equivalent in terms of the quality and quantity of information they provide about G_1.

We can leverage the concept of adjacency matrix to introduce the concept of **degree** of a vertex in a graph.

Definition 11.7 In an undirected graph G with $|\mathcal{V}|$ vertices, the degree of vertex i, generally defined as $k(i)$, is the **number of edges incident to it**[6]. Given the adjacency matrix A, $k(i)$ can be defined as

$$k(i) = \sum_{j=1}^{|\mathcal{V}|} A_{ij} = \sum_{j=1}^{|\mathcal{V}|} A_{ji} \tag{11.5}$$

6: In an undirected graph, we define a vertex *even* if its degree is even and *odd* if its degree is odd.

where in (11.5) we provided two equivalent expressions that entail, respectively, counting all the unitary values along the i-th row or column of adjacency matrix A. Because of the symmetrical properties of the adjacency matrix of an undirected graph, the two expressions are equivalent.

In a directed graph G with $|\mathcal{V}|$ vertices, the in-degree of vertex i, generally defined as $k_{in}(i)$, is the **number of edges pointing towards it**. Given the adjacency matrix A, $k_{in}(i)$ can be defined as

$$k_{in}(i) = \sum_{j=1}^{|\mathcal{V}|} A_{ji} \tag{11.6}$$

where (11.6) is equivalent to counting all the edges of G in the form (v_1, i). The in-degree has a counterpart, namely the out-degree. It is generally defined as $k_{out}(i)$ and is the **number of edges pointing away from vertex** i. Given the adjacency matrix A, $k_{out}(i)$ can be defined as

$$k_{out}(i) = \sum_{j=1}^{|\mathcal{V}|} A_{ij} \tag{11.7}$$

where (11.7) is equivalent to counting all the edges of G in the form (i, v_2). Finally, the degree of vertex i in a directed graph is the summation of its in- and out-degree values

$$k(i) = k_{in}(i) + k_{out}(i) = \sum_{j=1}^{|\mathcal{V}|} A_{ji} + \sum_{j=1}^{|\mathcal{V}|} A_{ij} \tag{11.8}$$

and represents the overall number of edges incident to it (pointing towards and away).

Example 11.2 *Given the two graphs of Figure 11.6, compute the degree values of all the vertices.*

Let us start with the undirected graph G_1. For vertex 1 we can write $k(1) = \sum_{j=1}^{4} A_{1,j} = 0 + 1 + 1 + 1 = 3$. This is confirmed by Figure 11.6a, where 3 edges are incident to vertex 1. With a similar strategy, we obtain $k(2) = 2$, $k(3) = 2$, and $k(4) = 1$.

We focus now on the directed graph G_2. For vertex 1 we can write $k_{in}(1) = \sum_{j=1}^{5} A_{j,1} = 0+1+0+0+0 = 1$ and $k_{out}(1) = \sum_{j=1}^{5} A_{1,j} = 0+0+1+0+1 = 2$, hence $k(1) = 3$. We can confirm these values by analyzing Figure 11.6b. For the other vertices we have $k_{in}(2) = 1$, $k_{out}(2) = 1$, $k(2) = 2$, $k_{in}(3) = 1$, $k_{out}(3) = 1$, $k(3) = 2$, $k_{in}(4) = 1$, $k_{out}(4) = 2$, $k(4) = 3$, and $k_{in}(5) = 3$, $k_{out}(5) = 1$, $k(5) = 4$.

We can leverage the definition of in- and out-degree to convey an interesting property of directed graphs. If a vertex i in a directed graph has an in-degree of zero, i.e., $k_{in}(i) = 0$, then such a vertex cannot be reached starting from another vertex in the graph and leaping from a vertex to another one using edges as bridges. We formally define that vertex as **not-reachable from any other vertex**. Conversely, if we place ourselves on a vertex i of a directed graph with an out-degree of zero ($z_{out}(i) = 0$), then we cannot leave such a vertex due to the lack of outbound edges.

Readers may have observed our discussion about navigating from one vertex to another in a graph using the available edges. As we will explore in Section 11.3, graphs serve as mathematical representations of real-world problems in OR and other fields. Many of these problems necessitate efficient[7] movement within the graph. To ease into these concepts, we will introduce the concept of a **walk**.

7: In general, the concept of *efficiency* can change quite broadly and relates to the specific objective function of the mathematical problem based on the graph under scrutiny.

Definition 11.8 A walk in a graph from vertex v_1 to vertex v_2 is a sequence of vertices (where the same vertex can appear multiple times in the walk) where each vertex v_i of the walk (apart from the initial v_1) can be reached from the previous one v_{i-1} because of the presence of edge (v_{i-1}, v_i). The walk is said to join v_1 and v_2. Additionally, the walk is defined **open** if $v_2 \neq v_1$ and **closed** if $v_2 = v_1$.

Example 11.3 *Considering Figure 11.6b, compute all the walks from vertex 1 to any other vertex.*

A walk from vertex 1 to vertex 2 is $\{1, 5, 4, 2\}$, a walk from vertex 1 to vertex 3 is $\{1, 3\}$, a walk from vertex 1 to vertex 4 is $\{1, 5, 4\}$, and a walk from vertex 1 to vertex 5 is $\{1, 5\}$. Note that there might be other potential walks connecting the same pair of vertices.

We now leverage the definition of walk to provide a formal definition of **connected graph**.

Definition 11.9 A connected graph is a graph where **every pair of vertices is joined by a walk**. **A complete graph (both undirected and directed) is connected by definition**.

We now introduce the beautiful concept of **Euler[8] walk**.

Definition 11.10 An Euler walk is a walk that **uses every edge in the graph exactly once**.[9]

Closely related to the concept of Euler walk is the concept of an **Hamilton[10] walk**.

Definition 11.11 An open Hamilton walk is a walk that **uses every vertex in the graph exactly once**. A closed Hamilton walk is a walk that **uses the first vertex twice (hence, it starts and ends there) and every other vertex once**.

Considering Figure 11.6, in Figure 11.6a the walk $\{(4, 1), (1, 3), (3, 2), (2, 1)\}$ is an open Euler walk as it used every edge exactly once starting in vertex 4 and ending in vertex 1. The walk $\{(1, 3), (3, 5), (5, 4), (4, 2), (2, 1)\}$ in Figure 11.6b is a closed Hamilton walk, as it uses every vertex once and vertex 1 twice (the walk starts and ends there). Hamilton walks, especially in their closed variant, will play a role of paramount importance in many mathematical models from Chapter 13.

We conclude this section with a special type of directed graph which will also play an important role in Chapter 13. Before introducing such a directed graph, we need to introduce the concept of **cycle**.

Definition 11.12 A cycle in a graph (undirected or directed) is a **sequence of connected vertices where only the first and last vertex are the same**.

For example, the closed Hamilton walk $\{(1, 3), (3, 5), (5, 4), (4, 2), (2, 1)\}$ from Figure 11.6b is also a cycle. Note that not every closed Hamilton walk is a cycle, as the former requires every vertex of the graph to be visited, while a cycle does not. To substantiate this claim, in Figure 11.6b, we have also cycle $\{(1, 5), (5, 4), (4, 2), (2, 1)\}$ which is not a Hamilton walk because it by-passes node 3. We provide an additional example in Figure 11.8, where both a closed Hamilton walk and a cycle are highlighted.

Having clarified the definition of a cycle, we are now ready to introduce readers to the concept of **Directed Acyclic Graph (DAG)**.

Definition 11.13 A DAG is a **directed graph that features no cycles**.

DAGs will also play a predominant role in some applications from Chapter 13. To better contextualize them, an example is provided in Figure 11.9.

8: Leonhard Euler (1707-1783) has been one of the greatest mathematicians of all time. To know more about him and his incredible discoveries, we refer readers to this Wikipedia page.

9: A beautiful application of such a concept is the *Seven Bridges of Königsberg* problem, as shown in Figure 11.7. We refer readers to this Wikipedia page.

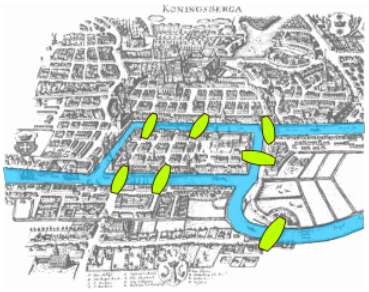


Figure 11.7: A representation of the famous Königsberg problem.

10: William Rowan Hamilton (1805-1865) has also been one of the greatest mathematicians of all time. To know more about him and his incredible discoveries, we refer readers to this Wikipedia page.

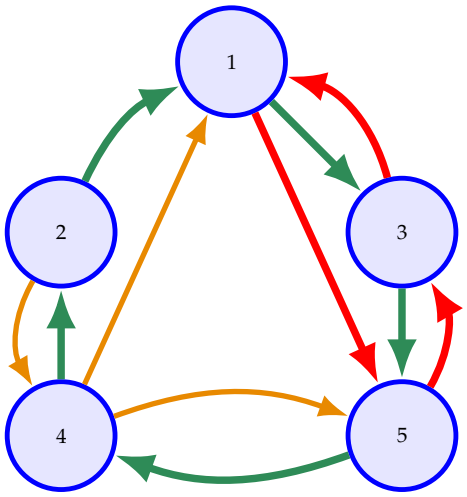


Figure 11.8: Example of a directed graph with a closed Hamiltonian walk (**green** arrows) and a cycle (**red** arrows). In **orange** are represented edges of the graph that are neither part of the closed Hamilton walk nor of the cycle.

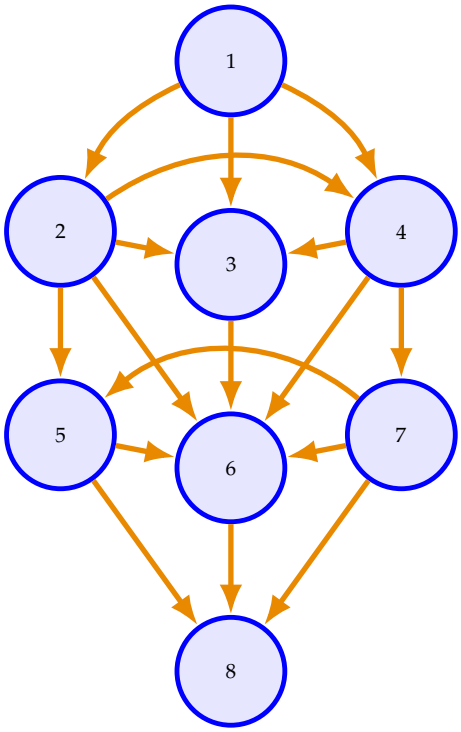


Figure 11.9: Example of a DAG.

11.3 From "abstract" graphs to "concrete" networks

In the relevant literature, the distinction between a graph and a network is frequently ambiguous and subject to various interpretations. Here, we take a specific approach. We use the term *graph*, as we have throughout this chapter, to denote an underlying and abstract representation of a mathematical problem. In contrast, once the graph is fully constructed and adapted to represent the practical problem at hand, it transforms into a *network*[11].

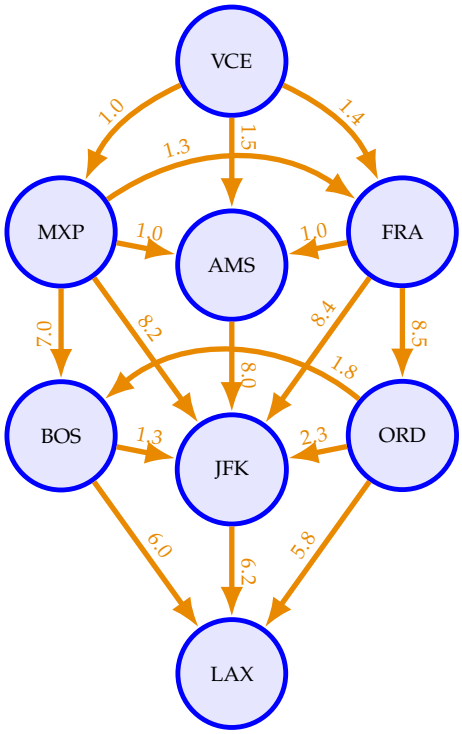
11: This does not mean that a network is a very faithful representation of the practical problem at hand. A network is still a mathematical abstraction of such a problem, yet tailored in a way that represents well enough the problem while being in a mathematical format that can be solved with ad-hoc algorithms.

Let us take, for example, the DAG from Figure 11.9. Its vertices could represent airports, with airport 1 being the origin airport for a planned trip and airport 8 the destination airport for the same trip. Vertices 2-6 then represent other airports where a traveler can perform a stopover on their journey from vertex (airport) 1 to vertex (airport) 8. Because of our association of a generic graph to a specific context, now our network can be customized. For example, the edges we highlighted are not random, but do represent direct flights between airports. Analyzing Figure 11.9, we can infer there is no direct flight between airports 1 and 8 as there is no $(1, 8)$ edge.

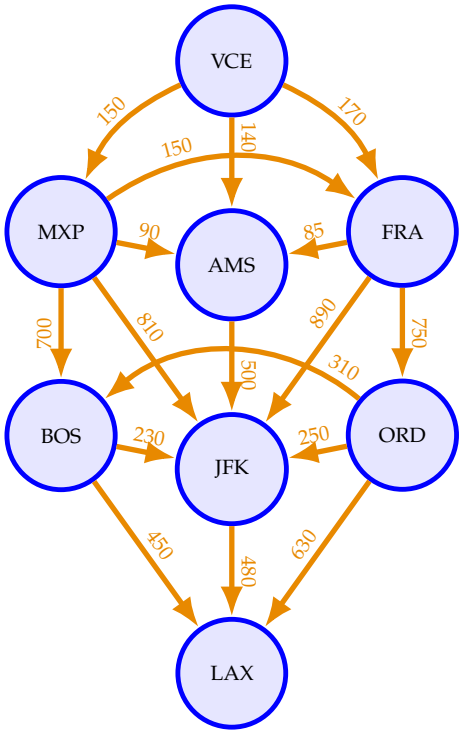
Now that we have transitioned from an abstract graph to a more concrete network, both vertices and edges can be contextualized with a set of features. For instance, vertices representing airports may be characterized by specific names, geographic locations, and other relevant information for modeling or visualization purposes. Similarly, edges can also be enriched with features. For example, in the context of air transport, an edge may represent a direct flight between two airports, with features such as average flying time and ticket fare. This notion sets the stage for the concept of a **weighted graph**. In the graphs we have discussed thus far, we assumed that the **cost** of an edge was unitary, symbolizing that traversing an edge equates to taking one step in the graph. However, when the graph is tailored to represent a practical problem and captures its features, thereby becoming a network, its underlying graph representation can incorporate weighted edges, where each weight represents the associated cost of using the edge. The adjacency matrix A can be used to store such weights by replacing, in each (i, j) element associated with an edge, the 1 with the appropriate weight. This matrix is generally called the **weighted adjacency matrix** W. We substantiate this statement in Example 11.4

Example 11.4 *We consider a couple of travelers who want to travel from Venice Marco Polo airport (VCE)*[12] *to Los Angeles International airport (LAX). The travelers have identified a set of airports and direct connections as represented in Figure 11.9, where vertex 1 represents VCE and vertex 8 represents LAX. The first traveler of the couple is mostly concerned about the overall flight time and, for every direct flight among the set of 8 airports, they collected data on the average flight time. This information is reported in Figure 11.10a, where the label of every edge reports the average time in hours. Conversely, the second traveler of the couple is mostly concerned about the overall money spent for the trip and, for every direct flight among the set of 8 airports, they collected data on the average ticket fare. This information is reported in Figure 11.10b, where the label of every edge reports the average ticket fare in €. Our goal is to help the two travelers assess what is the best traveling options available given their different needs.*

12: In Example 11.4 we will use use International Air Transport Association (IATA) codes to define airports to have a more concise representation.



(a) Edge weights represent the average flight time in hours.



(b) Edge weights represent the average ticket fare in €.

Figure 11.10: Graph representation of the airport network from Example 11.4.

To help the couple of travelers, we must formally compute a **shortest path** (see Section 12.5) in the airport network defined by the graphs of Figure 11.10. It is important to note that "shortest" is a broad term, and neither traveler is specifically concerned with distance as the primary KPI for their journey (which would typically be associated with a *shortest path*). The first traveler aims to minimize travel time, while the second traveler aims to minimize the financial cost of the trip. We can translate Figure 11.10a and Figure 11.10b into the associated weighted adjacency matrices to map the same amount of information in a different format (which might be less elegant visually, but handier for an algorithm[13]. For the first traveler we have

$$W = \begin{array}{c} \begin{matrix} VCE & MXP & AMS & FRA & BOS & JFK & ORD & LAX \end{matrix} \\ \begin{pmatrix} 0 & 1.0 & 1.5 & 1.4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0 & 1.3 & 7.0 & 8.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8.0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 & 0 & 8.4 & 8.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.3 & 0 & 6.0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6.2 \\ 0 & 0 & 0 & 0 & 0 & 2.3 & 0 & 5.8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} VCE \\ MXP \\ AMS \\ FRA \\ BOS \\ JFK \\ ORD \\ LAX \end{matrix} \end{array} \tag{11.9}$$

where we can practice with some of the definitions we covered in this chapter[14].

While we are not reporting the adjacency matrix A characterizing the airport network explicitly, it can be retrieved from either (11.9) or (11.10) by replacing any non-zero W_{ij} value with a unitary value in the corresponding A_{ij}. For instance, when computing $\sum_{j=1}^{8} A_{j,1}$, the result is 0, indicating that vertex 1 (VCE) has an in-degree of 0. This aligns with the airport network's structure, where VCE serves as the origin airport for the travelers and only features outbound connections. Similarly, when computing $\sum_{j=1}^{8} A_{8,j}$, the result is 0, signifying that vertex 8 (LAX) has an out-degree of 0. This corresponds to LAX being the destination airport for the travelers and only having inbound connections. **However, it is essential to clarify that this does not imply that VCE lacks inbound connections or LAX lacks outbound connections altogether. In the context of the relevant airport network for the travelers, such connections are simply not pertinent**.

Another insight is that JFK (vertex 6) is the airport with the most connections overall, as $k_6 = \sum_{j=1}^{8} A_{j,6} + \sum_{j=1}^{8} A_{6,j} = 5 + 1 = 6$ is the highest among all vertices. This degree value shows imbalance between inbound (5) and outbound (1) flights. Again, readers with some air travel experience might argue that JKF features connections towards all the other airports (set aside VCE, probably). The travelers might have overlooked flights from JFK to BOS or ORD indeed, but the choice not to include flights from JFK to MXP, AMS, or FRA is justified by the purpose of the problem. As the travelers are moving from Europe to the West Coast of the United States, it is logistically reasonable to mostly select flights headed towards west[15]. Similarly, we would agree with readers if they highlighted omissions by the travelers of flights from AMS to BOS,

13: In practice, especially for large problems, the full matrix format, as depicted in Equation 11.9, is highly inefficient. This inefficiency stems from the fact that many graphs, which serve as mathematical representations of real networks, are not densely populated. For instance, the graph illustrated in Fig.11.9 exhibits a density of 25%, and this percentage significantly decreases for larger networks. In our airport example, a low density implies that each airport typically connects to only a subset of all possible airports in the vertex set. Consequently, a majority of entries in the adjacency matrix are zero. To conserve memory space, alternative data structures such as **sparse matrices** are more adept at storing the same information with minimal memory consumption. We refer readers to this Wikipedia page for more information on sparse matrices.

14: Before doing so, we want to stress that while now we identify vertices with their IATA code both in Figure 11.10a and in (11.9), it is recommended to keep a numerical indexing as described in Section 4.1. Computers work more efficiently with numbers while we can switch between the numerical and any other convention of choice defining a relationship between the two. In this case, we kept the same numerical indexing as the original Figure 11.9 (also highlighted by the sequence of rows/columns in Equation 11.9), such that $1 \rightarrow VCE, 2 \rightarrow MXP, \cdots, 7 \rightarrow ORD, 8 \rightarrow LAX$.

15: While this reasoning sounds legitimate for travel time minimization, airline ticketing is a complex and highly nonlinear process. Sometimes, the cheapest traveling option entails weird itineraries.

ORD, and LAX. We wanted Example 11.4 not to become to cluttered at the cost of omitting well-known flight routes.

Focusing now on the second traveler, we build the weighted adjacency matrix as

$$W = \begin{array}{c} \begin{matrix} VCE & MXP & AMS & FRA & BOS & JFK & ORD & LAX \end{matrix} \\ \begin{pmatrix} 0 & 150 & 140 & 170 & 0 & 0 & 0 & 0 \\ 0 & 0 & 90 & 150 & 700 & 810 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 500 & 0 & 0 \\ 0 & 0 & 85 & 0 & 0 & 890 & 750 & 0 \\ 0 & 0 & 0 & 0 & 0 & 230 & 0 & 450 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 480 \\ 0 & 0 & 0 & 0 & 0 & 250 & 0 & 630 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} VCE \\ MXP \\ AMS \\ FRA \\ BOS \\ JFK \\ ORD \\ LAX \end{matrix} \end{array} \tag{11.10}$$

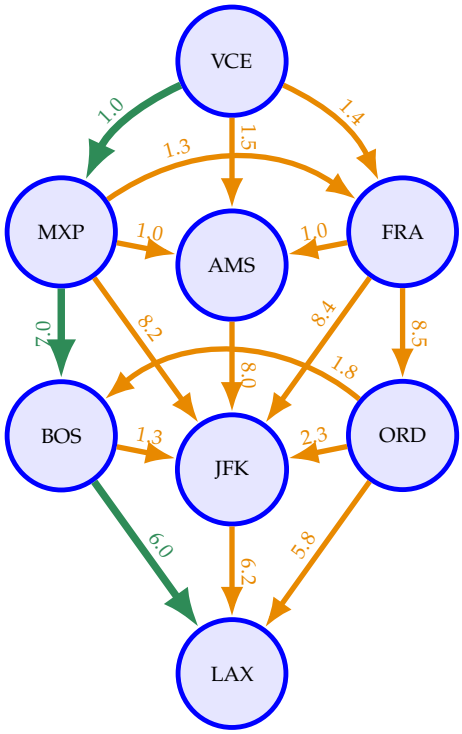
which features non-zero elements in the same (i, j) locations as (11.9) (assuming flying for free is not yet an option), as both weighted adjacency matrices relate to the very same adjacency matrix.

We abstain from presenting the actual mathematical formulation at this juncture. Nonetheless, we underscore that in this scenario, the best travel solution varies depending on whether we aim to minimize flight time to accommodate the first traveler or select the cheapest itinerary as requested by the second traveler. We highlight the two resulting itineraries in Figure 11.11.

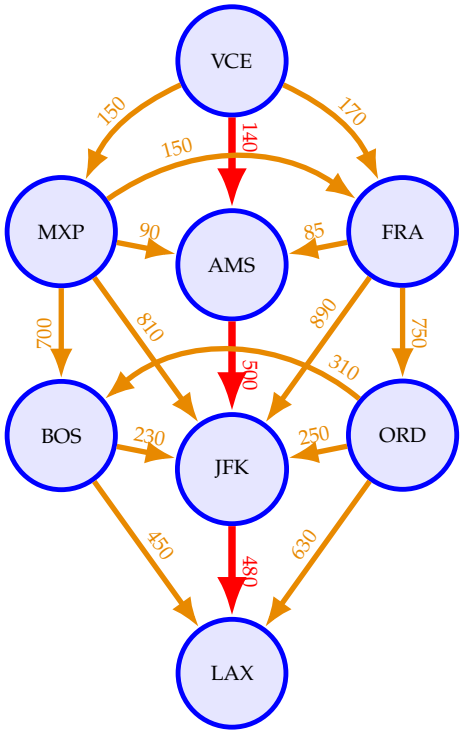
In Example 11.4 we provided readers with some insights on how the same graph (in this case, a graph with $|\mathcal{V}| = 8$ vertices and$|\mathcal{E}| = 16$ edges) can lead to different behaviors even when applied to the same practical network (an airport network), but with a different objective in mind. The same graph structure can actually define the same mathematical foundation for problems that have nothing in common (apart from the same graph representation). To better contextualize this claim, Figure 11.9 can represent the underlying graph of a network of water pipelines, where vertex 1 is the inlet and vertex 8 the outlet of water. In such a setting, our goal might be to process as much water throughput as possible without exceeding the structural capacity of each segment of the network. In another example, the graph could represent the network of cities reachable one another with a single charge of an electric car, with vertex 1 representing the origin city of a trip and vertex 8 representing the destination city of such a trip. The goal here could be to minimize the overall distance of the trip or to maximize the overall satisfaction (assuming every edge is associated with some sightseeing along the way which has been translated into a “positive" cost). Hence, three very distinct networks and related problems, but based on the very same graph. We will discuss more about network problems in Chapter 12.

Coded example

A tutorial to practice with the Python package **networkx**, designed for studying graphs and networks, is available here.



(a) Best itinerary according to the first traveler.



(b) Best itinerary according to the second traveler.

Figure 11.11: Best itinerary for Example 11.4 according to the first (**green** arrows) and second (**red** arrows) traveler.

Network problems 12

> Do not follow where the path may lead. Go instead where there is no path and leave a trail.
>
> Ralph Waldo Emerson

12.1 Transportation Problem (TP) 209
12.1.1 General Setting 209
12.1.2 TP: LP mathematical formulation 210
12.1.3 TP: solution with the transportation simplex 212
12.2 Maximum flow problem 225
12.2.1 An introduction to Column Generation (CG) 229
12.3 Minimum Cost Flow (MCF) problem 232
12.3.1 Single-source single-sink variant 232
12.3.2 Multiple-source multiple-sink variant . 233
12.4 Graph coloring problem 236
12.5 Shortest Path (SP) problem 241
12.6 Minimum Spanning Tree (MST) problem . 243

In this chapter, we deal with network problems, i.e., problems that can be represented using the graph representation we introduced in Chapter 11. Because many real-life problems can be modeled using a graph representation, network problems are part of the most studied and taught classes of problems in the OR domain. While some problems rely on a physical or geographical network representation, others are more abstract in nature and rely on such graph representation to ease the mathematical formulation.

Problems belonging to this category range are extremely diverse because a graph can be used to represent transportation systems (a set of roads, inland-water connections, airline connections, etc.), pipelines (e.g., water supply systems) and power grids, or precedence relationships between project activities, just to name a few examples.

Regardless of the specific case or nuance under scrutiny, a graph $G = (\mathcal{V}, \mathcal{E})$ can always be associated with the original network problem. The set of vertices $\mathcal{N}$ represents elements where flows (or commodities) can originate, end, or be exchanged, such as road intersections, airports, or activities that precede or follow other activities in a project. The edges represent the way flows or commodities can move within the graph, and hence define the connectivity properties of the graph. For example, as described in Chapter 11, some network problems might be defined on a complete graph, where flows can occur between any pair of nodes, while some other problems might offer a more restricted set of connectivity options. In addition, most network problems are defined on directed graphs, because the direction of flow is important in the problem at hand, but we shall see that there are some network problems based on undirected graphs[1].

1: In this chapter, we will use interchangeably *vertices* with *nodes* and edges with *links* and *arcs*. In fact, while vertices and edges are more appropriate when dealing with the more abstract graph representation, nodes and links/arcs are more common when dealing with the more concrete network representation, especially when the network entails routing of commodities of some sort.

12.1 Transportation Problem (TP)

12.1.1 General Setting

A Transportation Problem (TP), in its general form, can be defined as the problem of transporting goods from a set of sources ($s \in \mathcal{S}$) to a set of destinations ($d \in \mathcal{D}$) in a way that satisfies both the supply and demand requirements of the different stakeholders while minimizing the overall transportation costs. The set of sources $\mathcal{S}$ can represent warehouses or distribution centers where goods are produced or stored, while the set of destinations $\mathcal{D}$ can represent another set of warehouses downstream or the final recipients of the goods. Given such a setting, we can define

C_{sd} the transportation cost (per unit) from source s to destination d. In addition, we can define S_s as the supply (amount of goods produced) of source s and, in a similar fashion, D_d as the demand (amount of goods received) of destination d. Finally, we only need one set of decision variables x_{sd} (generally speaking, continuous) that define the flow of goods from source s to destination d.

Note that both supply and demand could be characterized by bounds and not by a fixed value. For example, a certain warehouse (source) could be able to deliver between 10 and 30 tonnes of flowers, and a certain flower retailer (destination) might be requesting between 5 and 10 tonnes of flowers. In order for a transportation problem to be feasible, it follows that **the maximum supply available should at least match the minimum demand requested**. We will see in Section 12.1.3 that a condition to ensure this is that

$$\sum_{s \in \mathcal{S}} S_s = \sum_{d \in \mathcal{D}} D_d \tag{12.1}$$

,i.e., that the overall supply produced exactly matches the requested demand. This is a necessary requirement if we want to solve the TP with ad-hoc algorithms, while it is a condition that might not be enforced if we decide to use a more general LP approach. Notwithstanding, the condition mentioned above of maximum supply and minimum demand should hold in order to have a feasible solution. **In addition, it might be the case that the transportation of goods from every source to every destination is not possible, because of distance constraints or other conditions**.

Figure 12.1 depicts an example of the TP with 3 sources ($|\mathcal{S}| = 3$) and 4 destinations ($|\mathcal{D}| = 4$). In the example, out of the 12 potential connections, only 9 are exploitable as source 1 can send goods to destinations 1,2, and 3, source 2 to destinations 1, 3, and 4, and source 3 to destinations 2, 3, and 4.

12.1.2 TP: LP mathematical formulation

We have already defined most of the parameters needed to model a generic TP as an LP in Section 12.1.1. Let us add parameters $(S_s^-, S_s^+)\ \forall s \in \mathcal{S}$ that define, respectively, the minimum and maximum supply a source can produce, such that $S_s^- \leq S_i \leq S_s^+\ \forall s \in \mathcal{S}$. We follow the same logic for the destinations with parameters $(D_d^-, D_d^+)\ \forall d \in \mathcal{D}$ that define, respectively, the minimum and maximum demand a destination can accept, such that $D_d^- \leq D_i \leq D_d^+\ \forall d \in \mathcal{D}$. In addition, let us define $\mathcal{S}_d$ the subset of sources that can serve destination d and $\mathcal{D}_s$ as the subset of demand destinations that source s can serve. We define the resulting LP as:

$$\min \sum_{s \in \mathcal{S}} \sum_{d \in \mathcal{D}_s} C_{sd} x_{sd} \tag{12.2}$$

s.t.:

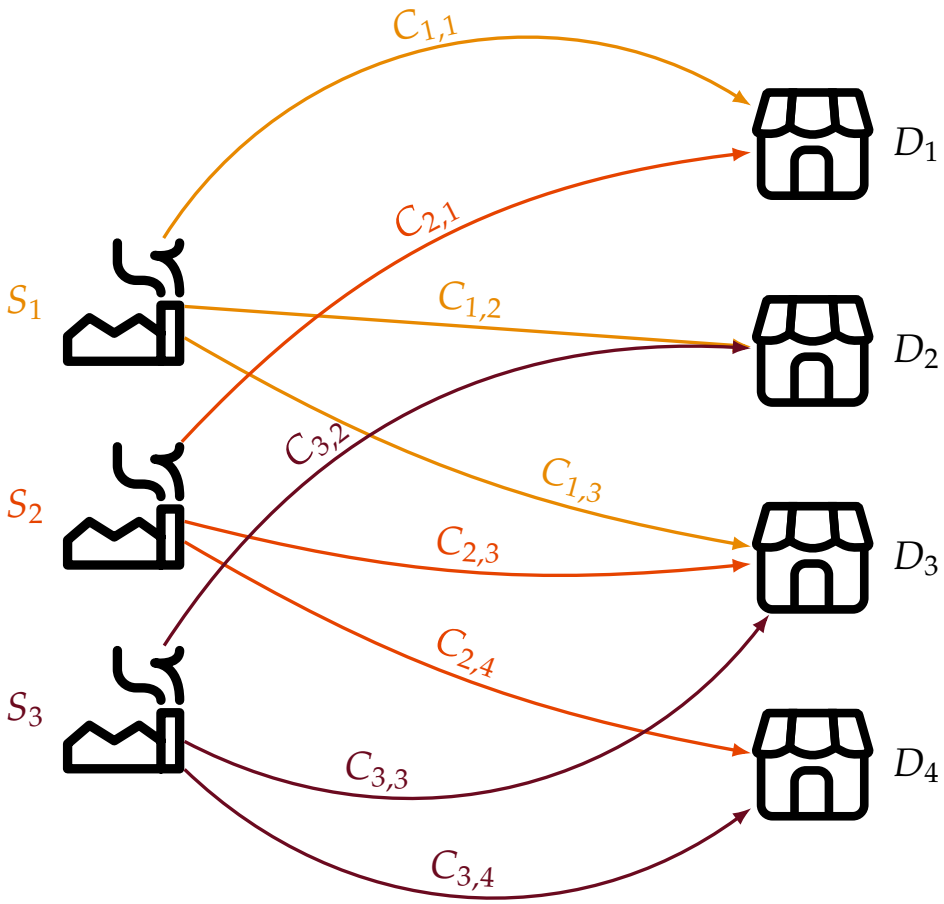


Figure 12.1: Generic framework of a TP.

$$S_s^- \leq \sum_{d \in \mathcal{D}_s} x_{sd} \leq S_s^+ \qquad \forall s \in \mathcal{S} \qquad (12.3)$$

$$D_d^- \leq \sum_{s \in \mathcal{S}_d} x_{sd} \leq D_d^+ \qquad \forall d \in \mathcal{D} \qquad (12.4)$$

$$x_{sd} \in \left[0, \min\{S_s^+, D_d^+\}\right] \qquad \forall s \in \mathcal{S}, d \in \mathcal{D}_s \qquad (12.5)$$

(12.2) defines the objective function, i.e., the minimization of transportation costs from all sources to all destinations that each source can serve. (12.3) ensures that each source delivers goods within its capabilities, while (12.4) ensures that each destination receives goods within its specified interval. Finally, (12.5) defines the continuous nature of each decision variable. Because of the capacity bounds on sources $s \in \mathcal{S}$ and destinations $d \in \mathcal{D}$, each x_{sd} cannot exceed the minimum between the maximum capacity that source s can produce or destination d can accommodate.

Let us now consider an example based on the same setting of Figure 12.1. The supply parameters are $S_1^- = 10$, $S_2^- = 10$, $S_3^- = 10$, $S_1^+ = 50$, $S_2^+ = 80$, and $S_3^+ = 60$. The demand parameters are $D_1^- = 50$, $D_2^- = 10$, $D_3^- = 70$, $D_4^- = 10$, $D_1^+ = 50$, $D_2^+ = 40$, $D_3^+ = 70$, and $D_4^+ = 50$. Transportation costs per unit are $C_{1,1} = 20$, $C_{1,2} = 11$, $C_{1,3} = 10$, $C_{2,1} = 14$, $C_{2,3} = 13$, $C_{2,4} = 13$, $C_{3,2} = 12$, $C_{3,3} = 11$, and $C_{3,4} = 20$.

By solving the TP model (12.2)-(12.5), the final solution is the following:

$x_{1,2} = 10$, $x_{1,3} = 40$, $x_{2,1} = 50$, $x_{2,4} = 10$, and $x_{3,3} = 30$. The overall transportation cost of such a solution is **1, 670 monetary units**. Hence, source 1 delivers as many goods as its production can allow ($S_1 = 50$), while sources 2 and 3 are utilized below maximum capacity ($S_2 = 60 \leq S_2^+ = 80$ and $S_3 = 30 \leq S_3^+ = 60$). As it concerns the destinations, destinations 1 and 3 receive exactly as requested (not surprisingly, as they require an exact amount), while destinations 2 and 4 receive an amount equal to their requested lower bound D_2^- and D_4^-. This is also not surprising because **it is a cost-minimization problem and supply providers have no incentive, in the current setting, to provide customers with more than the bare minimum requested**.

Coded example

The code used to model and solve the example TP as an LP is available here.

Coded example

In some circumstances, transporting goods from a source $s \in \mathcal{S}$ to a demand node $d \in \mathcal{D}_s$ might incur, on top of the transportation costs proportional to the amount of goods delivered, a fixed cost that "activates" the transportation arc (see Section 4.8.4). Because this variant is not a classic transportation problem, we do not treat it from a theoretical standpoint here, but directly provide an implementation with the associated LP formulation here.

12.1.3 TP: solution with the transportation simplex

In Section 12.1.2, we analyzed how to set up and solve a TP as, generally speaking, an LP if we allow transported values between origin and destinations to be continuous numbers. This is generally the case because we deal with aggregate values and not specific packages or items. In addition, if all the transportation costs C_{sd} are integer values, all the final values of our optimal solution will also be integers because of the way the simplex method operates.

In this section, we explain a solution method that is tailored to TP and that might be more efficient, especially for large-scale problems. Such a solution method is based on a specific matrix representation of the TP at hand, which we refer to as **TP table**. A necessary requirement to apply this ad-hoc method is that (12.1) must hold, because the underlying principle of such a method is to find the most efficient way (as in, cost-minimizing way) to move that aggregate supply from all the sources available to all the destinations available. We will see that using the TP table is only possible mathematically if $\sum_{s \in \mathcal{S}} S_s = \sum_{d \in \mathcal{D}} D_d$. In principle, this is necessary to comply with the never-aging concept of conservation of mass. This method requires two steps:

1. definition of an **initial solution** that allocates the supply from the different sources $s \in \mathcal{S}$ to the different destinations $d \in \mathcal{D}$;

2. **iterative revision of the supply and demand allocation** until a **specific optimality criterion is met**, i.e., until we can prove the achieved solution cannot be further improved.

We initially illustrate an example of the TP table set up in Table 12.1. In this example, we have 3 sources and 4 destinations, each delivering a precise amount of goods (sources) and requesting a precise amount of the same goods (destinations). Hence, we have $S_s^- = S_s^+ = S_s \ \forall s \in \mathcal{S}$ and $D_d^- = D_d^+ = D_d \ \forall d \in \mathcal{D}$. We assume that $S_1 = 40$, $S_1 = 50$, and $S_1 = 60$ for the supply side and $D_1 = 30$, $D_2 = 35$, $D_3 = 40$, and $D_4 = 45$ for the demand side, so that $\sum_{s \in \mathcal{S}} S_s = \sum_{d \in \mathcal{D}} D_d = 150$. Let us also assume that goods can be transported from any source to any destination with the following costs $C_{1,1} = 5$, $C_{1,2} = 3$, $C_{1,3} = 8$, $C_{1,4} = 9$, $C_{2,1} = 6$, $C_{2,2} = 4$, $C_{2,3} = 5$, $C_{2,4} = 3$, $C_{3,1} = 9$, $C_{3,2} = 8$, $C_{3,3} = 7$, and $C_{3,4} = 6$. Finally, let us assume that an analyst working on optimizing this TP came up with the following solution (where we inherit the same notation used in Section 12.1.2): $x_{1,1} = 30$, $x_{1,2} = 10$, $x_{2,2} = 25$, $x_{2,3} = 25$, $x_{3,3} = 15$, and $x_{3,4} = 45$.

We can map all the aforementioned information, which encompasses both parameters and a candidate solution, in Table 12.1 in a quite compact way. We have as many rows as sources (i.e., $|\mathcal{S}|$) and as many columns as destinations (i.e., $|\mathcal{D}|$). In each of the $|\mathcal{S}| \times |\mathcal{D}|$ cells, we report the transportation cost per unit in the small square in the top-right corner, and specify the amount of goods transported from s to d in the main cell. In addition, we pad the matrix at the bottom side with the values of requested demand per destination, and at the right side the values of produced supply per source.

	D_1	D_2	D_3	D_4	Supply
S_1	5 **30**	3 **10**	8	9	40
S_2	6	4 **25**	5 **25**	3	50
S_3	9	8	7 **15**	6 **45**	60
Demand	30	35	40	45	150

Table 12.1: Example of a TP table.

We can verify that the solution in Table 12.1 is feasible because every source delivers exactly the amount of goods produced (the summation of **colored values** in every row matches the supply value to the right) and every destination receives exactly the requested quantity (the summation of **colored values** in every column matches the demand value below).

Notwithstanding, some questions that might (and should) arise are:

- how to set up the TP table if supply and demand do not initially match?
- how to obtain a feasible, and possibly of good-quality, initial solution in a structured way?

- how to modify the values of the initial table to still satisfy $\sum_{s\in\mathcal{S}} S_s = \sum_{d\in\mathcal{D}} D_d$ while reducing costs, and how to perform this in an iterative fashion until we converge to the optimal solution?

For situations where no explicit balance between overall supply and demand is found, we need to evaluate the **maximum supply** that sources can provide and the **maximum demand** that destinations can receive, and assess where the deficit is. If there is a shortage of supply, a **dummy source** that produces that shortage must be defined. If there is a shortage of demand, a **dummy destination** that receives the excess of supply must be defined. Let us focus on a revised variant of the example showcased in Section 12.1.2, where we fix the three supply values to their upper bound and fix the four demand values to the lower bound (we proved that this is what they will receive anyway). If we sum all the supply and demand values we obtain

$$\begin{cases} S_1 = 50 \\ S_2 = 80 \\ S_3 = 60 \end{cases} \rightarrow \sum_{s\in\mathcal{S}} S_s = 190$$

$$\begin{cases} D_1 = 50 \\ D_2 = 10 \\ D_3 = 70 \\ D_4 = 10 \end{cases} \rightarrow \sum_{d\in\mathcal{D}} D_d = 140$$

The overall supply exceeds the overall demand by 50 units. If we want to solve a TP with the table structure shown in Table 12.1, we need balance between the two values. Hence, in this particular example, **we need to introduce a fifth demand such that $D_5 = 50$**. Readers must be wondering what is the role of a dummy supply or demand, since they are "fictitious" nodes that are added to ensure the overall balance between supply and demand. We will see later in this section how we can ensure that our solution avoids infeasible scenarios, such as when part of the minimum demand a destination requires comes from the dummy source, which is a non-existing source in reality. This scenario, albeit mathematically feasible, is not realistic in practice. We can map the revised problem as shown in Table 12.2.

Table 12.2: TP table with a dummy destination.

	D_1	D_2	D_3	D_4	D_5 (dummy)	Supply
S_1	**20**	**11**	**10**	**M**	**0**	50
S_2	**14**	**M**	**13**	**13**	**0**	80
S_C	**M**	**12**	**11**	**20**	**0**	60
Demand	50	10	70	10	50	190

In a TP table, every (source,destination) pair is mapped, even pairs that are formally not allowed to witness any flow of goods. We circumvent this apparent issue by assigning an extremely high cost to those pairs. This is the case, for example, of cell (S_1, D_4) whose cost is **M** (a big-M

as explained in Section 4.8.1: in our setting, source S_1 can only serve the first three destinations D_1, D_2, and D_3 (recall Figure 12.1).

A special mention goes to the D_5 column of Table 12.2, i.e., the column mapping the dummy destination. Because a dummy destination is a non-physical destination, it receives flows of goods that are meaningful only in our mathematical setting, but not in the real world. Hence, the cost of sending goods from a real source to a dummy destination is 0 because it ensures the mathematical balance between supply and demand but does not represent a "real" flow. Conversely, if a cell maps a flow of goods from a dummy source to a real destination, then the cost of that cell should be set to **M** because **we cannot provide a real destination node with goods emanating from a dummy source, as that is a non-physical, but just mathematically defined, flow**.

Having clarified how to ensure the balance of supply and demand in a TP table, we now describe in Section 12.1.3.1 two algorithms to obtain an initial solution.

12.1.3.1 Defining an initial solution of a TP

The first method is the simplest one, and is called the **North-West corner rule**. As the name implies, it requires starting from the North-West corner cell of a TP table (e.g., from cell (S_1, D_1) in Table 12.2). In that cell, which we define (s, d) to represent we are currently in row s and column d, we should place a value equal to the minimum between the remaining supply that row s can offer and the remaining demand that column d requires. If, for example, the remaining supply value is the smallest between the two, this means that supply node s is now saturated, as it is providing to some destinations everything it can produce. Hence, any additional value to the right along the same row s can be set to 0 (as they are using the whole supply s already), and we should move down to cell $(s + 1, d)$. Because the current demand d is not fully satisfied yet, we will need to rely on the next available supply node. Conversely, if the remaining demand value is the smallest between the two, we should set all the remaining values below cell (s, d) to 0, as the current demand has been satisfied. Because the current remaining supply s is not saturated yet, we will move to the right instead. We keep moving either to the right or below, introducing proper values in the cells, until we reach the South-East corner. **Note that this process works because of the assumption that the overall supply and demand levels are equal**.

We now show the application of the North-West corner rule to the TP defined in Table 12.2. The final result is shown in Table 12.3.

	D_1	D_2	D_3	D_4	D_5 (dummy)	Supply
S_1	**20** 50	**11** 0	**10**	**M**	**0**	50
S_2	**14**	**M** 10	**13** 70	**13** 0	**0**	80
S_3	**M**	**12**	**11**	**20** 10	**0** 50	60
Demand	50	10	70	10	50	190

Table 12.3: TP table with an initial (infeasible) solution generated with the North-West corner rule.

In Table 12.3 we see that two flow values are zero. Let us explain why. Starting from the North-West corner, we have an available supply of 50 (the full S_1) and a requested demand of 50 (the full D_1). Because the two values match, both the row and the column are saturated simultaneously and we then need to move diagonally. We split the diagonal movement into two steps: a horizontal and a vertical one. We arbitrarily decided to move horizontally first to (S_1, D_2) and then vertically to (S_2, D_2). It would have been equivalent to first move vertically and place a flow of zero in (S_2, D_1) and then move to the right to (S_2, D_2). The same "trick" is carried out between (S_2, D_3) and (S_3, D_4). With this initial solution, sources 1 and 2 use their supply in full (they do not send any supply to the dummy destination), while source 3 only provides 10 units of "real" supply, with the remaining 50 allocated to the dummy destination.

We can then retrieve the objective value by multiplying every non-zero coefficient in a cell by the associated cost. In our case $Z = 20 \times 50 + M \times 10 + 13 \times 70 + 20 \times 10 + 0 \times 50 = 2,110 + 10M$ (we omitted the two values with a flow of 0 as they are in the table just to ensure horizontal/vertical movements only). This is not a good solution for two reasons. First, we should recall from above that the optimal solution for this TP is $Z = 1,670$. **In addition, our solution obtained with the North-West corner rule features a $10M$ term, which suggests infeasibility: we provide 10 units from S_2 to D_2, which is not allowed**.

The low quality of the initial solution provided by the North-West corner rule can be explained as follows. While this algorithm is based on a very intuitive logic, it does not account anyhow the cost of each cell we place flow in. As a matter of fact, we just move right or down computing the smallest value between the remaining supply or demand, but such a move cannot prevent us from placing flow in a very costly cell. As we witnessed in Table 12.3, this process does not even prevent us from placing flow in a cell that maps an infeasible (source,destination) combination.

We now present an alternative algorithm to generate an initial solution for a TP that accounts for cost considerations when filling in the table, i.e., the **Vogel's method**. This method improves the myopic assignment performed by the North-West corner rule as follows. For each row and column of the table, it is computed the difference between the cheapest and second-cheapest cost. Then, the row or column with the highest difference is selected and a proper flow value is placed in the cell characterized by the cheapest cost so that either the row or column associated with the cell is saturated. The saturated row or column is "removed" from the table, supply and demand values are updated, and the process is repeated until no more rows or columns are left. In the case of ties, they can be broken arbitrarily. **The underlying idea of Vogel's method is that we should focus on rows or columns where the difference between the best (cheapest) and second-best (second-cheapest) option is the largest, because missing the opportunity of using that "cheap" cell will incur a considerable increase in cost**.

We showcase Vogel's method with the same TP table that we used for the North-West corner rule. In each presented table, we add an additional column displaying the row difference and an additional row displaying the column difference.

	D_1	D_2	D_3	D_4	D_5 (dummy)	Supply	Row diff.
S_1	20	11	10	M	0	50	10
S_2	14	M	13	13	0	80	13
S_3	M	12	11	20	0	60	11
Demand	50	10	70	10	50	190	
Column diff.	6	1	3	7	0		

Table 12.4: TP table filled in with the Vogel's method: initial setup.

Analyzing Table 12.4, the largest value is in the S_2 row. Focusing on that row, the cell with the smallest cost is (S_2, D_5) with a cost of 0 (being D_5 the dummy destination). We can place there a value of $\min\{80, 50\} = 50$ so that column D_5 is saturated. We update the table as shown in Table 12.5. Note that we removed column D_5 and updated the supply of S_2. We also updated the value depicting the remaining supply/demand available, which is now reduced to 140. Albeit not explicit in the table because of the removal of column D_5, we should remember that $(S_2, D_5) = 50$. Another relevant note regards row S_2. Because there are two columns with the same smallest cost of 13, the row difference for that row is 0.

	D_1	D_2	D_3	D_4	Supply	Row diff.
S_1	20	11	10	M	50	1
S_2	14	M	13	13	30	0
S_3	M	12	11	20	60	1
Demand	50	10	70	10	140	
Column diff.	6	1	1	7		

Table 12.5: TP table filled in with the Vogel's method: situation after setting $(S_2, D_5) = 50$ and removing column D_5.

In Table 12.5, the new highest value is yielded by column D_4 with a value of 7. A value of $\min\{30, 10\} = 10$ is placed in cell (S_2, D_4), saturating column D_4. The remaining supply of S_2 is then 20. We report the updated table in Table 12.6. The largest value is now in column S_2. We place a value of $\min\{20, 50\} = 10$ in cell (S_2, D_1) which saturates row S_2. We update the situation as displayed in Table 12.7. The largest difference is now in column D_1, where a value of $\min\{50, 30\} = 30$ is placed. Column D_1 is now saturated and removed. The revised situation is displayed in Table 12.8. In Table 12.8, every remaining row and column features a difference of 1, hence we have a tie. We decide arbitrarily to select cell (S_1, D_3) and insert a value of $\min\{20, 70\} = 20$ there. This saturates row S_1. We need one last step, as we are left with just row S_3, as shown in Table 12.9. Given Table 12.9, the only feasible option to satisfy the remaining demand is to set $(S_3, D_2) = 10$ and $(S_3, D_3) = 50$. We can now summarize the solution obtained with the Vogel's method restoring the original TP table and placing all the flow values that were selected at every iteration. The final result is displayed in Table 12.10.

The solution reported in Table 12.10 is feasible, as no cell with a cost of M has been assigned a flow value. **This is an inherent property of the Vogel's method, which identifies the row or column with the highest difference between the lowest and second-lowest cost. Hence, a cell**

Table 12.6: TP table filled in with the Vogel's method: situation after setting $(S_2, D_4) = 10$ and removing column D_4.

	D_1	D_2	D_3	Supply	Row diff.
S_1	20	11	10	50	1
S_2	14	M	13	20	1
S_3	M	12	11	60	1
Demand	50	10	70	130	
Column diff.	6	1	1		

Table 12.7: TP table filled in with the Vogel's method: situation after setting $(S_2, D_2) = 20$ and removing row S_2.

	D_1	D_2	D_3	Supply	Row diff.
S_1	20	11	10	50	1
S_3	M	12	11	60	1
Demand	30	10	70	110	
Column diff.	M	1	1		

Table 12.8: TP table filled in with the Vogel's method: situation after setting $(S_1, D_1) = 30$ and removing column D_1.

	D_2	D_3	Supply	Row diff.
S_1	11	10	20	1
S_3	12	11	60	1
Demand	10	70	80	
Column diff.	1	1		

Table 12.9: TP table filled in with the Vogel's method: situation after setting $(S_1, D_3) = 20$ and removing row S_1.

	D_2	D_3	Supply	Row diff.
S_3	12	11	60	1
Demand	10	50	60	
Column diff.	-	-		

Table 12.10: TP table with an initial (feasible) solution generated with the Vogel's method.

	D_1	D_2	D_3	D_4	D_5 (dummy)	Supply
S_1	20 **30**	11	10 **20**	M	0	50
S_2	14 **20**	M	13	13 **10**	0 **50**	80
S_3	M	12 **10**	11 **50**	20	0	60
Demand	50	10	70	10	50	190

with a cost equal to M is extremely unlikely to be ever chosen. We compute the objective value of the solution represented in Table 12.10 as $Z = 20\times30+10\times20+14\times20+13\times10+0\times50+12\times10+11\times50 = 1,880$. While we now have a feasible solution[2], the result still does not match the optimal one we previously identified. In other words, we should define a way to improve our initial solution and converge towards an optimal value. By achieving this, we address the third question from Section 12.1.3 that has been left unanswered so far. We answer such a question in Section 12.1.3.2.

2: The range of methods to determine the initial solution of a TP table is wider than the two options presented here. For example, Hillier and Lieberman (2015) discusses the Russel's method as well, while Carter et al. (2018) discusses the minimum cost and minimum "row" cost methods.

12.1.3.2 The transportation simplex

As the title of this section suggests, we shall see that the TP can be interpreted similarly to what is done for any LP with the simplex method, and the simplex tableau in particular. **As a matter of fact, a structure such as Table 12.10 can be interpreted as a tableau. Before diving into the specifications of the transportation simplex method, let us provide readers with an insight**.

Let us consider, in Table 12.10, row $s = 1$ associated with S_1. While we do not know yet how to assign the five flow values optimally, we are required to assign them such that $\sum_{d\in\mathcal{D}} C_{1d}x_{1d} = 50$, (as the 50 units from S_1 must be allocated somewhere). With C_{sd} we define the transportation cost per unit flow from supply node s to demand node d. The same reasoning applies to every other row (supply) or column (demand). If we model a TP with the table setting shown here, our goal is represented by the following mathematical model:

$$\min \sum_{s\in\mathcal{S}} \sum_{d\in\mathcal{D}} C_{sd}x_{sd} \tag{12.6}$$

s.t.:

$$\sum_{d\in\mathcal{D}} x_{sd} = S_s \qquad \forall s \in \mathcal{S} \tag{12.7}$$

$$\sum_{s\in\mathcal{S}} x_{sd} = D_d \qquad \forall d \in \mathcal{D} \tag{12.8}$$

$$x_{sd} \geq 0 \qquad \forall s \in \mathcal{S}, d \in \mathcal{D} \tag{12.9}$$

(12.6) aims at minimizing the overall transportation costs. Constraints (12.7) ensure that the supply of each source node $s \in \mathcal{S}$ is used entirely across the different demand destination nodes, and constraint (12.8) ensure that each demand $d \in \mathcal{D}$ is exactly met, regardless of which combination of supply nodes satisfies such a demand. Finally, constraints (12.9) ensure that the flow variables x_{sd} are non-negative. **Note that the formulation is slightly different than (12.2)-(12.5) because we are enforcing that each supply and demand node produces (resp. requests) a fixed amount of goods, not a value within a range**. We now reconnect to the assignment problem and stipulate why it can be interpreted as a network problem (see Hillier and Lieberman (2015) and Carter et al.

(2018) for even more details) in box **Interpretation of an assignment problem as a TP**.

Interpretation of an assignment problem as a TP

Given the formulation (12.6)-(12.9), we realize that the assignment problem is a special type of TP where the **set of supply nodes $\mathcal{S}$ becomes the set of tasks $\mathcal{T}$ and the set of destination nodes $\mathcal{D}$ is the set of recipients $\mathcal{R}$**. In addition, each task $i \in \mathcal{T}$ offers a supply of 1 unit (each task should be assigned to one recipient) and each recipient demands 1 task, hence a unitary demand as per TP jargon. Because each flow can be unitary at most, we can also replace continuous decision variables with binary ones (if all the coefficients are integer-valued, we can leave them continuous because all the corner points will be integer-valued anyway). Replacing S_s and D_d with 1 and the nature of the decision variables (and acknowledging the different notation set- and index-wise) we translate formulation (12.6)-(12.9) into formulation (9.1)-(9.4).

An important insight is now to analyze what happens if, for example, we reduce every coefficient of a specific row, i.e., row $s = 1$ mentioned above, by a fixed quantity u_1. **Because such a constant does not appear in any constraint, the current solution will remain feasible**. Conversely, our revised objective value Z' is:

$$Z' = \sum_{s\in\mathcal{S}}\sum_{d\in\mathcal{D}} C_{sd}x_{sd} - \sum_{d\in\mathcal{D}} u_1 x_{1d} = \underbrace{\sum_{s\in\mathcal{S}}\sum_{d\in\mathcal{D}} C_{sd}x_{sd}}_{Z} - u_1 \underbrace{\sum_{d\in\mathcal{D}} x_{1d}}_{S_1} = Z - u_1 S_1 \tag{12.10}$$

Hence, we are reducing our initial objective by a quantity equal to $u_1 S_1$, i.e., the constant we used to reduce every coefficient of row $s = 1$ times the supply associated with the row. We could apply the same process to every other row s by reducing all its coefficients by constant u_s and every column d by reducing all its coefficients by constant v_d. We hence obtain the general expression for the revised objective:

$$Z' = \underbrace{\sum_{s\in\mathcal{S}}\sum_{d\in\mathcal{D}} C_{sd}x_{sd}}_{Z} - \sum_{s\in\mathcal{S}} u_s S_s - \sum_{d\in\mathcal{D}} v_d D_d = Z - \sum_{s\in\mathcal{S}} u_s S_s - \sum_{d\in\mathcal{D}} v_d D_d \tag{12.11}$$

In practice, given a choice of constants u_s for the rows and v_d for the columns, we scale our objective value without hindering the feasibility of the solution. Furthermore, this implies that every transportation cost coefficient of the original problem C_{sd} is scaled as

$$C'_{sd} = C_{sd} - u_s - v_d \tag{12.12}$$

where the minus sign in front of u_s and v_d implies that we are **reducing** the original coefficient if the constants are positive and **increasing** the

original coefficient if the constants are negative.

We now take a step forward. Given a feasible solution of a TP tabulated as in Table 12.10, finding a combination of u_s and v_d such that every revised coefficient C'_{sd} of a cell characterized by a flow is 0, implies that $Z' = 0$. We can achieve this in Table 12.10 by setting $u_1 = 6$, $u_2 = 0$, $u_3 = 7$, $v_1 = 14$, $v_2 = 5$, $v_3 = 4$, $v_4 = 13$, and $v_5 = 0$ (we will explain soon how to compute such constants). Recalling that from the original Table 12.10 $Z = 1,880$, we apply Equation 12.11 to obtain $Z' = 1,880 - 6 \times 50 - 7 \times 60 - 14 \times 50 - 5 \times 10 - 4 \times 70 - 13 \times 10 = 1,880 - 1,880 = 0$

Note that this does not mean we managed to reduce our transportation cost to 0, as we should always refer to the original Table 12.10. Notwithstanding, this "revised" problem sets the basis for the transportation simplex method. We display the revised problem in tabular form in Table 12.11, where we added a column and row to display the chosen u_s and v_d and revised all cost coefficients according to (12.12). In addition, in the top-right box of each cell we retain the original transportation cost coefficient C_{sd}, while in the cell itself we display either the flow value in **orange** or the reduced cost C'_{sd} in **fuchsia**. Note that there should be no confusion in this way. **Orange** values represent flows, i.e., the basic decision variables for which C'_{sd} reduced costs are 0. Conversely, **fuchsia** values represent the reduced cost C'_{sd} of non-basic variables.

	D_1	D_2	D_3	D_4	D_5 (dummy)	Supply	u_s
S_1	20 / 30	11 / 0	10 / 20	M / M − 19	0 / −6	50	6
S_2	14 / 20	M / M − 5	13 / 9	13 / 10	0 / 50	80	0
S_3	M / M − 21	12 / 10	11 / 50	20 / 0	0 / −7	60	7
Demand	50	10	70	10	50	190	
v_d	14	5	4	13	0		

Table 12.11: TP table with the revised problem where cells with flow variables feature a cost coefficient $C'_{sd} = C_{sd} - u_s - v_d = 0$.

In Table 12.11, cells with flow values are now characterized by a cost coefficient equal to 0. This resembles very suspiciously what explained in Chapter 6 for the simplex method and the coefficients of each basic variable in the simplex tableau. As a matter of fact, we are leveraging the same intuition here. We define, as already anticipated above, the x_{sd} values characterizing a flow from s to d in a TP table the **basic variables** of the problem, with all the remaining cells mapping **non-basic variables** where no flow is present. While we leave the mathematical details out (but refer interested readers to Hillier and Lieberman (2015) for the explanation), it can be proven that for a TP with $|\mathcal{S}|$ sources and $|\mathcal{D}|$ demand nodes, $|\mathcal{S}| + |\mathcal{D}| - 1$ basic variables are needed. In our example, $|\mathcal{S}| = 3$ and $|\mathcal{D}| = 5$, hence 7 basic variables are needed. **This is the case of the solution obtained with Vogel's method (Table 12.10) and with the North-West corner rule (Table 12.3). In the latter case, 2 basic variables are 0[3], but the overall number is 7 anyway**.

Similar to what was done in Chapter 6, we can assess the "quality" of our solution by checking the revised coefficients of the non-basic variable in a TP table. A negative coefficient of a non-basic variable, given the way we defined u_s, v_d, and the table, implies that if we make such a decision

3: A case where some basic variables are characterized by a value of 0, as hinted at in Chapter 6, is called a degenerate solution. We refer readers to Hillier and Lieberman (2015) for more details.

variable basic, the objective will be reduced by that coefficient for every unit of flow we place in that cell. Referring back to **Table 12.11**, we have $C'_{3,5} = -7$. Hence, for every unit of flow that we place in cell $(3, 5)$, our transportation cost will be reduced by 7.

We now proceed how to accomplish this. Note that we cannot simply add a random positive value in cell $(3, 5)$. Every row and column of a **TP** table sums up to a specific supply or demand value. Hence, increasing a variable from 0 to a positive value (hence, making a non-basic variable basic) will start a chain reaction so that in every row and column the summation of all values is preserved. In particular, if we make x_{sd} basic, we will have to reduce the value of a basic variable both in row s and column d to leave the summation unscathed. Note that, as x_{sd} is currently non-basic, there must be at least one basic variable in row s and in column d to ensure that supply and demand values, respectively, are met. This step will in turn generate a snowball effect unless we identify a closed circuit starting and ending in cell (s, d). Within this closed circuit, we will re-arrange the flows so that their summation does not change in every row and column involved.

In addition, we will identify the maximum amount that we can subtract from any of the basic variables part of the circuit without making any other negative and that we can reallocate to variable x_{sd}. The former basic variable will be deducted by its current value, hence featuring a revised value of 0 and becoming non-basic. That amount will be added to the non-basic variable x_{sd} so that it becomes basic. This step ensures that the number of $|\mathcal{S}| + |\mathcal{D}| - 1$ remains fixed. After updating the C'_{sd} coefficients to ensure they are 0 for all the basic variables, the process is repeated until no negative coefficient of a non-basic variable is found. This condition, equivalent to what we already discussed in Chapter **6**, ensures the current allocation of flows is optimal.

Let us describe how to form the closed circuit for the case represented in Table 12.11. Making cell $(3, 5)$ basic means reducing the flow value in cell $(2, 5)$ to ensure demand D_5 is not exceeded. Reducing $(2, 5)$ means increasing either $(2, 1)$ or $(2, 4)$ to ensure S_2 is not underutilized. If we were to increase $(2, 4)$ we would exceed D_4 as there are no basic variables in column $d = 4$ that can then be reduced. Hence, we need to increase $(2, 1)$. By doing so, we exceed D_1 unless we reduce $(1, 1)$. Reducing $(1, 1)$ must be compensated by an increase of $(1, 3)$ to preserve S_1. In turn, the increase of $(1, 3)$ must be matched by an equivalent decrease of $(3, 3)$ to ensure D_3 is not exceeded. The decrease of $(3, 3)$ connects with the original sought increase of $(3, 5)$, closing the circuit, so that S_3 is used fully.

So far, we discussed which basic variables should increase or decrease their value, but did not discuss to what extent. recall that we are not creating or destroying flow, but only reallocating it. Hence, along the closed circuit, we identify the smallest value of the basic variable that should decrease its value. We subtract this value from every basic variable that must be reduced and add it to every other basic variable and to the entering basic variable. In essence, the non-basic variable “steals" the flow value from the smallest basic variable part of the closed circuit (which becomes non-basic).

While this process might seem daunting, it is nothing more than an application of the conservation of mass principle. We reallocate flows in every row and column part of the closed circuit so that the cell with the current most negative reduced cost receives a flow (doing this reduces the objective value). This reallocation must ensure that no negative flows are generated as a by-product of the process. We now proceed to translate this theoretical description into practice.

In the left matrix of (12.13) we show the closed circuit. The matrix replicates Table 12.11 but only reports the flow values for simplicity. In **green** we display values that should increase, while in **red** we display values that should decrease. Recall that our goal is to increase the flow value in element (3, 5), which is currently non-basic. Because the smallest red value is 30, then we will set $x_{3,5} = 30$ and increase all the other green values by 30 as well while reducing the red values by the same amount. The right matrix of (12.13) depicts the revised flows. Note that each row and column still sums up to the associated S_s or D_d value: as mentioned, **we are just reallocating flows across the closed circuit**.

$$\begin{pmatrix} 30 & 0 & 20 & 0 & 0 \\ 20 & 0 & 0 & 10 & 50 \\ 0 & 10 & 50 & 0 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & 0 & 50 & 0 & 0 \\ 50 & 0 & 0 & 10 & 20 \\ 0 & 10 & 20 & 0 & 30 \end{pmatrix} \tag{12.13}$$

We can update the TP table as shown in Table 12.12.

	D_1	D_2	D_3	D_4	D_5 (dummy)	Supply	u_s
S_1	20 ?	11 ?	10 50	M ?	0 ?	50	?
S_2	14 50	M ?	13 ?	13 10	0 20	80	?
S_3	M ?	12 10	11 20	20 ?	0 30	60	?
Demand	50	10	70	10	50	190	
v_d	?	?	?	?	?		

Table 12.12: TP table with the revised problem after having made $x_{3,5}$ basic and $x_{1,1}$ non-basic, but before having updated the u_s and v_d coefficients.

The revised value of the basic variables reflect the changes applied using the closed circuit and the conservation of mass principle. We can compute the new solution using such flow values and the original coefficients (as those are the coefficients mapping the actual transportation costs): $Z = 10 \times 50 + 14 \times 50 + 13 \times 10 + 0 \times 20 + 12 \times 10 + 11 \times 20 + 0 \times 30 = 1,670$. **While we know this solution to be optimal because we solved the same problem as an LP, we have not proven it yet with the transportation simplex**. As a matter of fact, Table 12.12 does not comply with the requirements of a TP table as not all the $C'_{sd} = C_{sd} - u_s - v_d$ reduced costs of the basic variables are 0. Hence, we need to recompute all u_s and v_d values to ensure all basic variables are characterized by a revised coefficient $C'_{sd} = C_{sd} - u_s - v_d = 0$. This is also highlighted by the question marks in the associated column and row in Table 12.12. As we did not explain how to properly compute those coefficients when constructing Table 12.11, we proceed to do it in the **How to determine u_s and v_d values to update a TP table** box.

How to determine u_s and v_d values to update a TP table

Note that there is not a single combination of u_s and v_d values that sets all the revised cost coefficients of the basic variables to be 0. Here, we provide the same procedure as described in Hillier and Lieberman (2015). While explaining the procedure, we will use Table 12.12 to showcase the procedure. The procedure is as follows:

- **Input**: TP table with original coefficients C_{sd} and an initial solution comprising $|\mathcal{S}| + |\mathcal{D}| - 1$ basic variables;
- identify the row s or column d with the most basic variables. Let us assume it is row s (if it was a column, we would need to swap the sequence of columns and rows when determining new values). We then set $u_s = 0$ and label that row as **marked** because all u_s and v_d of basic variables associated to that row have been determined. Because we need to enforce $C'_{sd} = C_{sd} - u_s - v_d = 0$, then $v_d = C_{sd}$ for every basic variable in that row. In our example, row $s = 2$ has 3 basic variables. We set $u_2 = 0$, which implies $v_1 = 14$, $v_4 = 13$, and $v_5 = 0$.
- WHILE not all rows and columns are marked:
  - from the last marked row (column), identify the column d (row s) characterized by basic variables with an assigned v_d (u_s) but no u_s (v_d). Update those values using the relationship $C_{sd} - u_s - v_d = 0$ and set the column (row) as marked
- Once all u_s and v_d constants are computed, we set all the C'_{sd} coefficients of basic variables to 0 and update all the coefficients of non-basic variables as $C'_{sd} = C_{sd} - u_s - v_d$ (**recall that regardless of which iteration we are, the C_{sd} coefficients are the original ones**);
- **Output**: updated TP table with new basic variables and C'_{sd} coefficients.

In our case, after marking row $s = 2$ and setting $u_2 = 0$ and then $v_1 = 14$, $v_4 = 13$, $v_5 = 0$, we focus on column $d = 5$, where we set $u_3 = C_{3,5} - v_5 = 0 - 0 = 0$ and label it as marked. We then focus on row $s = 3$ where we set $v_2 = C_{3,2} - u_3 = 12 - 0 = 12$ and $v_3 = C_{3,3} - u_3 = 11 - 0 = 11$ and label both columns as marked. Finally, because in column $d = 3$ there is still cell $(1, 3)$ with an unassigned u_1, we focus on row $s = 1$ and set $u_1 = C_{1,3} - v_3 = 10 - 11 = -1$.

Table 12.13: TP table with the revised problem after having made $x_{3,5}$ basic and x_1, 1 non-basic and having updated the u_s and v_d coefficients.

	D_1	D_2	D_3	D_4	D_5 (dummy)	Supply	u_s
S_1	20 7	11 0	10 50	M $M - 12$	0 1	50	-1
S_2	14 50	M M – 12	13 2	13 10	0 20	80	0
S_3	M M – 14	12 12	11 20	20 7	0 30	60	0
Demand	50	10	70	10	50	190	
v_d	14	12	11	13	0		

By using the described procedure, we update all C'_{sd} coefficients in Table 12.12 as shown in Table 12.13. **We notice that all C'_{sd} reduced costs of the non-basic variables are either 0 or positive. Hence, turning any of them into a basic variable will not further reduce the transportation costs: our solution is proven to be optimal**. We summarize the full transportation simplex algorithm in the **The transportation simplex algorithm** box.

The transportation simplex algorithm

- **Input**: TP table with original coefficients C_{sd}, supply and demand values $S_s \ \forall s \in \mathcal{S}$ and $D_d \ \forall d \in \mathcal{D}$;
- compute an initial solution (with any available method) comprising of $|\mathcal{S}| + |\mathcal{D}| - 1$ basic variables;
- determine all the u_s and v_d constants and update all C'_{sd} values. In the main portion of each cell of the TP table, **store either the flow value (if that cell represents a basic variable) or the C'_{sd} value** (if that cell represents a non-basic variable);
- WHILE min $\left\{C'_{sd}\right\} < 0$:
  - **identify most-negative reduced cost** C'_{sd} of a non-basic variable;
  - **construct a closed circuit** and update the flow values so that the aforementioned non-basic variable becomes basic and a basic variable becomes non-basic;
  - **recompute all constants** u_s and v_d.
- **Output**: optimal TP table with final basic variables and associated x_{sd} values.

12.2 Maximum flow problem

A maximum flow model deals with **finding a feasible flow distribution across a capacitated network so that the maximum inflow in the network is obtained**. Some practical applications of such a model are:

- **hydraulic engineering**. Maximum flow network models can be applied to optimize the flow of water in pipelines or distribution networks under normal circumstances. Additionally, they are applied in the context of flood management to determine the maximum capacity of flood barriers, thereby reducing the risk of flooding in urban areas;
- **transportation systems**. Maximum flow network models are used, for example, in road networks to determine the maximum capacity of roads or routes, helping to alleviate congestion and reduce travel times;
- **communication networks**. Maximum flow network models can be used to optimize data routing, ensuring efficient transmission of information between network nodes while avoiding bottlenecks and congestion.

As demonstrated in some of the previous examples, directionality significantly influences maximum flow network problems. Whether dealing with traffic flow networks, logistic systems for goods transportation,

or data transmission networks, the directional nature of flows in these networks is crucial. To this avail, the maximum flow network problem is defined in the context of a directed graph $G = (\mathcal{N}, \mathcal{E})$[4], with $\mathcal{N}$ the set of nodes and $\mathcal{E}$ the set of edges. Among the set of nodes, two special nodes are the source $s \in \mathcal{N}$ and the sink $t \in \mathcal{N}$[5]. The underlying setting entails determining the maximum flow incoming to s that can be feasibly routed across the network on its way to t. The limitation on processing all inbound flow to s arises from the maximum capacity U_e associated with each edge $e \in \mathcal{E}$. Consequently, we introduce a set of decision variables $x_e \in \mathbb{R}_0$, where x_e represents the flow quantity along edge e. We report all the notation employed in the description of the maximum flow problem in Table 12.14.

4: We use the set of nodes $\mathcal{N}$ instead of the set of vertices $\mathcal{V}$ that we introduced in Chapter 11 because in flow problems the concept of *node* is more widespread than the concept of *vertex*.

5: While s would be the best candidate option as the index for the *sink* node, s is already taken by the *source* node. Hence, we chose t to highlight the fact that the sink node acts as a *target* node.

Table 12.14: Notation for the maximum flow problem.

Sets and indices	
$\mathcal{N}$	Set of nodes $i \in \mathcal{N}$
$\mathcal{E}$	Set of edges $e \in \mathcal{E}$
Parameters	
U_e	maximum capacity of arc e
Variables	
$x_e \in \mathbb{R}_0$	flow along edge $e \in \mathcal{E}$

Before diving into the formulation, let us define some important subsets related to edges. We define $\delta_i^- \subseteq \mathcal{E}$ the subset of edges outbound from node i, i.e., all the edges in G in the form (i, v_2) (where i is the first node). Conversely, we define $\delta_i^+ \subseteq \mathcal{E}$ the subset of edges inbound to node i, i.e., all the edges in G in the form (v_1, i) (where i is the second node). We use this notation as sending flow along an edge $e \in \delta_i^-$ "reduces" the net flow across node i (the flow is leaving the node), while sending flow along an edge $e \in \delta_i^+$ "increases" the net flow across node i (the flow is approaching the node).

We define the maximum flow problem as:

$$\max \sum_{e \in \delta_s^-} x_e - \sum_{e \in \delta_s^+} x_e \tag{12.14}$$

s.t.:

$$\sum_{e \in \delta_i^-} x_e - \sum_{e \in \delta_i^+} x_e = 0 \qquad \forall i \in \mathcal{N} \setminus \{s, t\} \tag{12.15}$$

$$x_e \leq U_e \qquad \forall e \in \mathcal{E} \tag{12.16}$$

(12.14) defines our objective, namely to maximize the difference between the overall flow exiting and entering the source node s. In some cases, the network assumes no inbound arcs to s ($\delta_s^+ = \emptyset$) and no outbound arcs from t ($\delta_t^- = \emptyset$). In such a case, the objective reduces to max $\sum_{e \in \delta_s^-} x_e$. This scenario is prevalent in academic and educational literature on

OR. In some other cases, the network features a single inbound arc to s with a fixed flow F, leading to an objective of max $\sum_{e\in\delta_s^-} x_e - \sum_{e\in\delta_s^+} x_e = \sum_{e\in\delta_s^-} x_e - F$. Although F is a constant and could be omitted, it is retained in the objective to assess whether the final objective value is positive, zero, or negative. A negative difference signifies excess flow that the network cannot process, which must be redirected in practical operations[6]. In our examples, we will employ the more traditional first approach.

6: In essence, the difference between the two cases is simply whether to check if a network can feasibly process a certain expected inflow after the optimization (first case) or already as a direct result of the optimization (second case).

Example 12.1 *A water pipeline comprises 5 primary junctions and 7 connecting pipes. It foresees an influx of 600 liters of water per hour in the next few hours, following heavy rains, with the influx originating from a single entry point into the pipeline. The pipeline network, depicted in Figure 12.2, specifies the maximum hourly capacity for each pipe connection. The objective is to determine if the network can accommodate the anticipated water influx.*

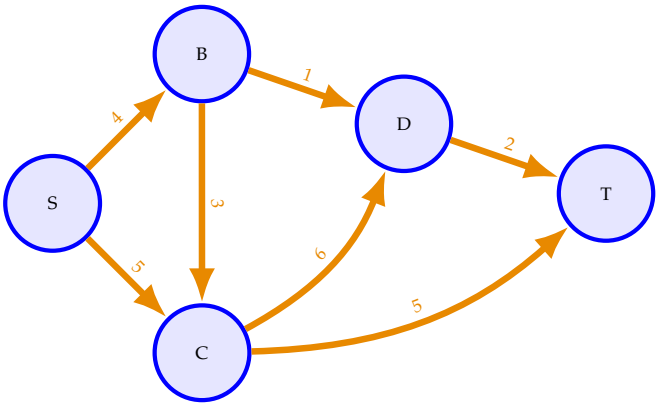


Figure 12.2: Network representation of the water pipeline of Example 12.1.

Our graph $G = (\mathcal{N}, \mathcal{E})$ is defined by the set of nodes $\mathcal{N} = \{S, B, C, D, T\}$ and edges $\mathcal{E} = \{(S,B),(S,C),(B,C),(B,D),(C,D),(C,T),(D,T)\}$. We assume the inflow enters the network in node S (source) and must be routed towards node T (sink). We divided every flow value by 100 to have a more compact notation. Given the small scale of the problem, we express the full LP formulation as

$$\max \quad x_{SB} + x_{SC} \tag{12.17}$$

s.t.:

$$x_{BD} + x_{BC} - x_{SB} = 0 \tag{12.18}$$
$$x_{CD} + x_{CT} - x_{SC} - x_{BC} = 0 \tag{12.19}$$
$$x_{DT} - x_{BD} - x_{CD} = 0 \tag{12.20}$$
$$x_{SB} \leq 4 \tag{12.21}$$
$$x_{SC} \leq 5 \tag{12.22}$$
$$x_{BC} \leq 3 \tag{12.23}$$
$$x_{BD} \leq 1 \tag{12.24}$$
$$x_{CD} \leq 6 \tag{12.25}$$
$$x_{CT} \leq 5 \tag{12.26}$$
$$x_{DT} \leq 2 \tag{12.27}$$

which, once solved with BB as described in Chapter 7, yields the following objective $x_{SB} + x_{SC} = 7$ with $x_{SB} = 2$, $x_{SC} = 5$, $x_{BC} = 2$, $x_{CD} = 2$, $x_{CT} = 5$, and $x_{DT} = 2$[7]. This example provides valuable insights into the solution and offers practical considerations. Among the 7 edges, only one, (B, D), is not intended to handle the flow. Significantly, this edge has the lowest hourly capacity of 100 liters, suggesting it may function as a bottleneck if utilized. **We conclude that the pipeline, pending no unexpected circumstances, will handle the water influx safely as it can accommodate 700 liters per hour compared with an expected 600**.

7: As described in Chapter 6 and Chapter 7, because all coefficients are integer, the final solution is also integral. Note that in this particular case, this was not needed as water flows could, and most likely will be, fractional values.

To illustrate this assertion, we present the solution in Figure 12.3. The green arcs represent edges utilized in the solution. Each arc is annotated with two values: the first indicates the current flow along the edge, while the second, enclosed in parentheses, denotes the remaining capacity available on the edge. For instance, the notation 5, (2) signifies a current flow of 5 with a total capacity of 7, indicating that 2 units could still be sent along the edge.

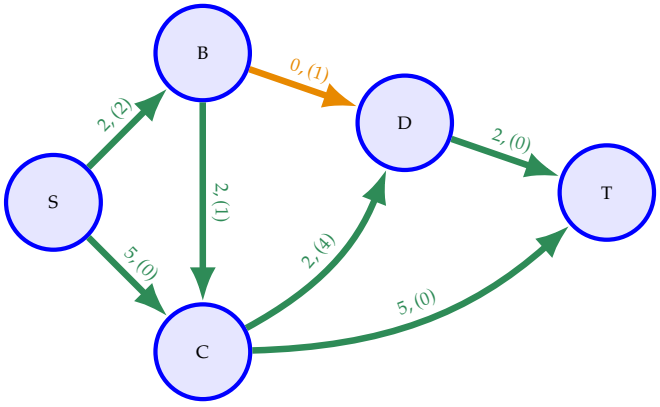


Figure 12.3: Final solution to the maximum flow problem applied to the water pipeline of Example 12.1.

We report a couple of additional considerations that can be drawn from the solution of Example 12.1. In a network like the one depicted in Figure 12.3, one can establish an upper bound on the maximum flow the network can accommodate by computing $\min\left\{\sum_{e\in\delta_s^-} U_e, \sum_{e\in\delta_t^+} U_e\right\}$. **The first term signifies the total capacity of all edges departing from the source node s, while the second term represents the total capacity of all edges leading into the sink node t. Since we presume that all flow originates from s and terminates at t, the minimum value between these two cumulative capacities defines a theoretical maximum that limits the flow the network can sustain**. Furthermore, the solution illustrated in Figure 12.3 demonstrates the dispatch of 700 liters of water within the network, not the expected 600. Being a *maximum flow* problem, it just adheres to its own set of semantics. Adjusting the solution from Figure 12.3 to reflect the expected 600 liters can be carried out as a post-processing step. For example, by reducing x_{SC} from 5 to 4 and x_{CT} from 5 to 4 as well. An alternative is to reduce x_{SB}, x_{BC}, x_{CD}, and x_{DT} from 2 to 1[8]. **This small example justifies the following statement. Generally, the ways to dispatch the maximum flow possible along a network are quite limited. If we reduce the injected flow, then the number of allowed options increases quite substantially**. Notwithstanding, we might be asked to find one of the many options already as part of the output of the mathematical model and not as a post-processing by-product. We display this in Example 12.2

8: There are other options available (we encourage readers to find them).

Coded example

The code used to model and solve Example 12.1 is available here.

Example 12.2 *After confirming that the network from Example 12.1 can handle a flow rate of $F = 600$ liters of water per hour, our next objective is to formulate an optimization model that determines a feasible routing of this water within the network.*

The graph $G = (\mathcal{N}, \mathcal{E})$ is unchanged with respect to Example 12.1. The only difference is that now we use the expected water flow F as an integral part of the model rather than in the post-processing phase to assess if the network can handle it (we verified it can). We define this new LP as:

$$\max \ 1 \tag{12.28}$$

s.t.:

$$\sum_{e \in \delta_i^-} x_e - \sum_{e \in \delta_i^+} x_e = \begin{cases} F, & i = s \\ -F, & i = t \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in \mathcal{N} \tag{12.29}$$

$$x_e \leq U_e \qquad \forall e \in \mathcal{E} \tag{12.30}$$

(12.28) implies that this is a feasibility and not an optimization problem. We already assessed in Example 12.1 that the network can handle 600 liters of water per hour. **Hence, our goal here is to find a feasible solution by ensuring the constraints are satisfied**. Formally, there is no objective to maximize or minimize, and we highlight this by specifying max 1 (choosing 1 is arbitrary) as a dummy objective. Because we require F units to be injected into the system via s and to exit the system via t, flow conservation constraints must be enforced there as well. This is what (12.29) achieves: it ensures a net flow equal to F leaves the source and converges to the sink, while maintaining a net flow of zero elsewhere in the network. (12.30) is inherited directly from Example 12.1. A potential solution to this problem is displayed in Figure 12.4, where for the sake of clarity we report with red arrows the flow of water $F = 600$ (recall that we divided all values by 100 in Figure 12.4) entering and exiting the network.

We conclude this section by briefly touching upon an alternative solution approach widely applied to problems dealing with the routing of flows in networks (among other applications).

12.2.1 An introduction to Column Generation (CG)

Column Generation (CG) is a mathematical modeling approach to efficiently deal with problems with a vast number of decision variables. **Instead of considering all possible variables upfront, the main insight of CG is to start with a small subset and iteratively add new variables**.

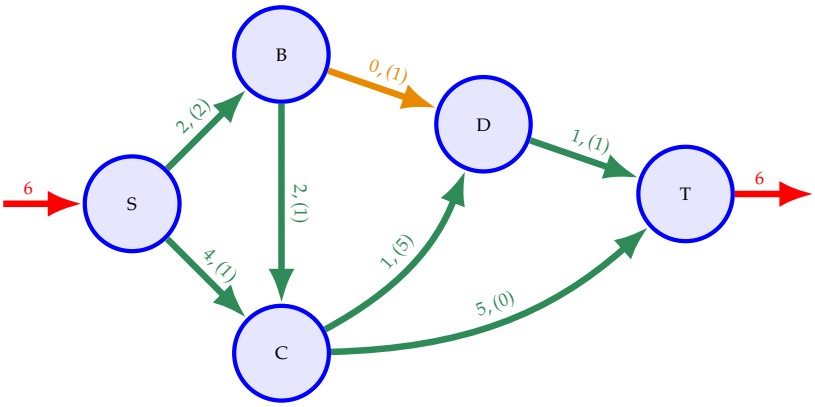


Figure 12.4: A feasible solution to the variation of the maximum flow problem applied to the water pipeline of Example 12.2.

In mathematical modeling, the constraints of a problem are typically represented as rows, while decision variables are represented as columns in the $Ax \leq b$ matrix structure. **Hence, because this methodology adds new decision variables iteratively, the approach is aptly named CG.**

It is not our goal to fully explain CG, and we refer readers to Desaulniers et al., 2006 for a seminal reference on the technique. In this section, our goal is to provide some insights and show how CG can be applied to solve the same problem as in Example 12.1. In problems based on a graph representation $G = (\mathcal{N}, \mathcal{E})$, usually decision variables are based on the edges (such as x_e from Example 12.1). This modeling choice, paired with ensuring that solutions make sense physically (for example enforcing conservation of flow), allows exploring the whole spectrum of possible solutions and, hopefully, converge to an optimal one. The issue of such an approach, as hinted at above, is that the number of variables explodes with the size of the problem. This approach is sometimes labeled an *arc-based* approach. An alternative approach, which is the underlying principle of CG, is to use a *path-based* approach. **With this approach, we decide how to route flows (or other commodities such as aircraft, cars, information packages, etc.) along paths directly going from the intended origin to the intended destination rather than along connected sequences of arcs.**

Each newly generated path serves as a distinct decision variable, effectively adding a new column to the model. However, unlike traditional methods where flow conservation is explicitly imposed as a constraint, in CG, the generated paths must adhere to fundamental conditions such as continuity without the need for explicit flow conservation constraints. The key for such a method is how and to what extent to compute "promising" paths, i.e., paths that can improve our objective. This step is the main challenge of the whole CG and is based on the concept of **duality** that we refrain from treating here (we refer interested readers to Hillier and Lieberman (2015)). Notwithstanding, we provide an intuitive interpretation in Example 12.3

Example 12.3 *We are asked to tackle the same problem as in Example 12.1, but by considering the water flow to move along paths connecting s and t rather than along sequences of edges.*

We start the exercise by providing an example of a path. Considering Figure 12.2, water could move from S to T using the sequence of arcs (S, C), (C, D), and (D, T), where the associated path can be expressed as the

sequence of nodes (S, C, D, T). We might wonder how much water can flow along the path, as each edge is capacitated. Because the path should be feasible, we need to satisfy the capacity of edges (S, C), (C, D), and (D, T) simultaneously. Hence, we compute $\min\{5, 6, 2\} = 2$. We could then add this flow of water to our network and update the values of the used and available capacity of all edges part of the path, and look for another path until we can: this is exactly the strategy we will follow.

While a proper CG will give us indications on how to select the most promising path, here we test out chances by randomly selecting them. We start with path (S, C, T) that can accommodate 5 units. We display the updated situation in Figure 12.5.

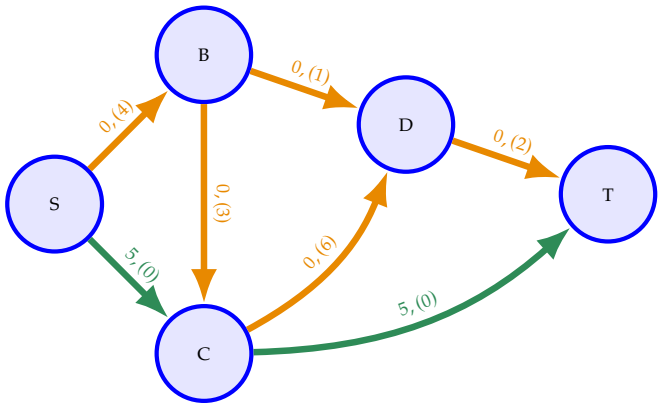


Figure 12.5: Solution to the maximum flow problem of Example 12.3 after adding path (S, C, T).

We notice that now both edges (S, C) and (C, T) are "saturated", i.e., we use them at their full capacity, as highlighted by the label $5, (0)$. The next step is to look for other paths to add. An option is (S, B, C, D, T), whose capacity is $\min\{4, 3, 6, 2\} = 2$. Hence, we add the path and our flow from S to T increases to 7. We display the updated solution in Figure 12.6.

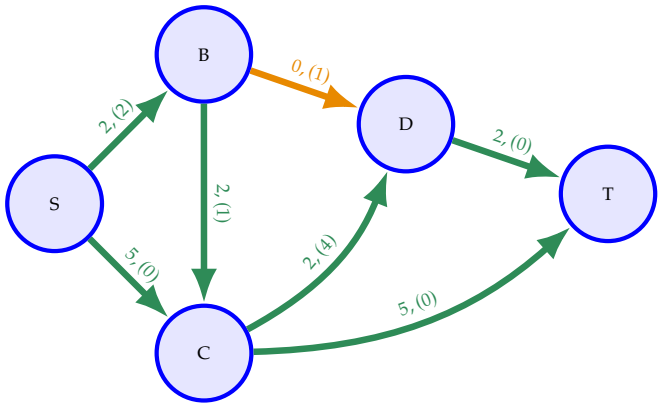


Figure 12.6: Solution to the maximum flow problem of Example 12.3 after adding path (S, B, C, D, T). The solution matches the one obtained in Example 12.1.

We could search for more paths, but to no avail. Because every path must end either with edge (D, T) or edge (C, T) and since both of them are saturated (respectively with 2 and 5 units of water), no additional path with a positive flow can be added. We could have guessed this already, as out current solution of 7 units (5 sent via (S, C, T) and 2 via (S, B, C, D, T)) matches the optimal one from Example 12.1.

Without realizing this, we employed the policy of **adding each time the path with the highest capacity left**. Given the objective at hand, this "greedy" approach sounds reasonable to achieve the intended goal. We encourage readers to try a different sequence, for example starting with

path (S, B, C, T) as the first path to add, and to verify if the same optimal solution is attained.

12.3 Minimum Cost Flow (MCF) problem

A Minimum Cost Flow (MCF) problem displays resemblance with the maximum flow network problem described in Section 12.2, yet the overarching goal is different. Here, the goal is to find the cheapest way to process a given amount of flow across a network. We will analyze two variants of such a problem, namely the single-source single-sink version in Section 12.3.1 and the multiple-source multiple-sink variant in Section 12.3.2. Both variants are based on a directed graph representation $G = (\mathcal{N}, \mathcal{E})$[9] of the network defining the problem. A selection of practical applications of such a model is:

9: Similar to Section 12.2, we use the set of nodes $\mathcal{N}$ instead of the set of vertices $\mathcal{V}$ that we introduced in Chapter 11.

- **transportation and logistics**. MCF models are used for optimizing the flow of goods and resources through networks. This includes optimizing shipping routes, managing vehicle fleets, and minimizing transportation costs while satisfying demand constraints;
- **network design**. MCF models are applied to design communication networks, such as telecommunication networks and computer networks. They help in routing data packets efficiently while considering factors like bandwidth constraints and minimizing communication costs;
- **water distribution networks**. MCF models are used to optimize the flow of water through distribution networks, such as water supply systems and irrigation networks. They help in managing water resources efficiently, minimizing water loss, and ensuring equitable distribution of water to consumers.

12.3.1 Single-source single-sink variant

In this version of the MCF problem, the goal is to process within the directed graph G a flow of a given commodity that spawns from a single vertex, namely the source $s \in \mathcal{N}$ and must be channeled towards a single vertex, namely the sink $t \in \mathcal{N}$ at minimum cost.

This single-source single-sink variant is characterized by the following parameters. Each arc $e \in \mathcal{E}$ features a maximum flow it can withstand U_e and a cost C_e we incur if the arc is used in the solution. In addition, the source node s and the sink node t are characterized by the same net flow value F that enters the network in s and exits in t. Using the same notation as in Section 12.2, we need the same set of decision variables $x_e \quad \forall e \in \mathcal{E}$, where x_e defines the amount of flow moving across arc e. We summarize all the introduced notation in Table 12.15

We define the single-source single-sink variant of the MCF as:

$$\min \sum_{e \in \mathcal{E}} C_e x_e \tag{12.31}$$

s.t.:

Sets and indices	
$\mathcal{N}$	set of nodes $i \in \mathcal{N}$
$\mathcal{E}$	set of edges $e \in \mathcal{E}$
Parameters	
C_e	cost of using edge $e \in \mathcal{E}$
U_e	maximum capacity of edge $e \in \mathcal{E}$
F	flow exiting source node $s \in \mathcal{N}$ and entering sink node $t \in \mathcal{N}$
Variables	
$x_e \in \mathbb{R}_0$	flow along edge $e \in \mathcal{E}$

Table 12.15: Notation for the single-source single-sink MCF problem.

$$\sum_{e \in \delta_s^-} x_e - \sum_{e \in \delta_s^+} x_e = F \tag{12.32}$$

$$\sum_{e \in \delta_t^-} x_e - \sum_{e \in \delta_t^+} x_e = -F \tag{12.33}$$

$$\sum_{e \in \delta_i^-} x_e - \sum_{e \in \delta_i^+} x_e = 0 \qquad \forall i \in \mathcal{N} \setminus \{s, t\} \tag{12.34}$$

$$x_e \leq U_e \qquad \forall e \in \mathcal{E} \tag{12.35}$$

where (12.31) defines the objective function, i.e., the minimization of the overall cost due to the utilization of the arcs in the network[10]. We activate some of the arcs via (12.32)-(12.33) which enforce, respectively, that the summation of the flows leaving the source node s and the summation of the flows entering the sink node t is equal to F. As a reminder, with $\delta_i^- \subseteq \mathcal{E}$ and $\delta_i^+ \subseteq \mathcal{E}$ we identify, respectively, the subset of arcs outbound from and inbound to node i. (12.34) enforce the more classic version of conservation of flow for all nodes that are neither the source nor the sink of the network. Finally, (12.35) enforces the proper upper bound to every flow decision variable x_e.

10: Note that here, differently from the maximum flow problem of Section 12.2, we assume the full flow F can be processed across the network. Otherwise, we would obtain the optimal solution $\sum_{e \in \mathcal{E}} C_e x_e = 0$ where no flow at all is processed at zero cost. In case there is no feasible solution for a given F, this entails such a value should be reduced as the current network cannot dispatch such a flow across the set of edges.

Coded example

A coded version of a single-sink single-source MCF problem is available here.

12.3.2 Multiple-source multiple-sink variant

The multiple-source multiple-sink variant of the MCF problem inherits all the features of the single-source single-sink version presented in Section 12.3.1. The additional complexity resides in the fact that now multiple source nodes are possible. They are stored in set $\mathcal{S} \subseteq \mathcal{N}$ where F_s is the flow emanating from source node s. Similarly, multiple sink nodes are possible. They are stored in set $\mathcal{T} \subseteq \mathcal{N}$ where F_t is the flow required by sink node t. **We assume that a node cannot behave as both a source and sink of flow, hence $\mathcal{S} \cap \mathcal{T} = \emptyset$. Even if there was a node j behaving, formally, both as a source and as a sink, we could compute the difference between the specified outbound flow and inbound flow. If such a difference is zero, then the node behaves as a "regular" node.**

If the difference is positive, the node behaves as a source where F_s is the computed difference. If the difference is negative, the node behaves as a sink where F_t is the absolute value of the computed difference (because the minus sign is already captured in the constraints). We define all the notation needed for the multiple-source multiple-sink MCF in Table 12.16.

Table 12.16: Notation for the multiple-source multiple-sink MCF problem.

Sets and indices	
$\mathcal{N}$	Set of nodes $i \in \mathcal{N}$
$\mathcal{E}$	Set of edges $e \in \mathcal{E}$
$\mathcal{S} \subseteq \mathcal{N}$	Set of sources $s \in \mathcal{N}$
$\mathcal{T} \subseteq \mathcal{N}$	Set of sinks $t \in \mathcal{N}$
Parameters	
C_e	cost of using edge $e \in \mathcal{E}$
U_e	maximum capacity of edge $e \in \mathcal{E}$
F_s	flow exiting source node $s \in \mathcal{S}$
F_t	flow entering sink node $t \in \mathcal{T}$
Variables	
$x_e \in \mathbb{R}_0$	flow along edge $e \in \mathcal{E}$

We define the multiple-source multiple-sink variant of the MCF as:

$$\min \sum_{e \in \mathcal{E}} C_e x_e \tag{12.36}$$

s.t.:

$$\sum_{e \in \delta_s^-} x_e - \sum_{e \in \delta_s^+} x_e = F_s \qquad \forall s \in \mathcal{S} \tag{12.37}$$

$$\sum_{e \in \delta_t^-} x_e - \sum_{e \in \delta_t^+} x_e = -F_t \qquad \forall t \in \mathcal{T} \tag{12.38}$$

$$\sum_{e \in \delta_i^-} x_e - \sum_{e \in \delta_i^+} x_e = 0 \qquad \forall i \in \mathcal{N} \setminus \{\mathcal{S} \cup \mathcal{T}\} \tag{12.39}$$

$$x_e \leq U_e \qquad \forall e \in \mathcal{E} \tag{12.40}$$

The structure of the problem is left unchanged. The only differences are in the definition of (12.37)-(12.38), which now potentially define multiple constraints according to the $|\mathcal{S}|$ and $|\mathcal{T}|$ values.

An important principle to emphasize is that, **while each source $s \in \mathcal{S}$ and sink $t \in \mathcal{T}$ may have different flow values F_s and F_t, it is essential that the following equality holds: $\sum_{s \in \mathcal{S}} F_s = \sum_{t \in \mathcal{T}} F_t$**. This equality, although left without a mathematical proof here, stems from the conservation of mass principle within the network. Simply put, mass cannot be created or destroyed within the system. Therefore, the total flow injected into the system ($\sum_{s \in \mathcal{S}} F_s$) must equal the total flow leaving the system ($\sum_{t \in \mathcal{T}} F_t$).

It is worth noting that we implicitly acknowledged this principle earlier in Section 12.3.1 by defining F as both the flow exiting the single source and the flow entering the single sink. **In essence, the single-source single-sink variant can be seen as a special case of the multiple-source multiple-sink variant when $|\mathcal{S}| = |\mathcal{T}| = 1$. However, we chose to introduce and discuss the single-source single-sink variant first before expanding to the more general case.**

Example 12.4 *A water supplier manages water from two origin stations, producing* $10,000$ *and* $5,000$ m^3 *per day, respectively. The total volume must be transported through a pipeline network to an urban area for distribution. The network, represented by graph* $G = (\mathcal{N}, \mathcal{E})$*, includes the two water sources and the final destination node, as shown in Figure 12.7. Each directed edge in the network indicates its maximum capacity in* $1,000$ m^3 *per day and its associated cost per* $1,000$ m^3 *processed. The objective is to determine the optimal flow of water across the network to minimize operational costs.*

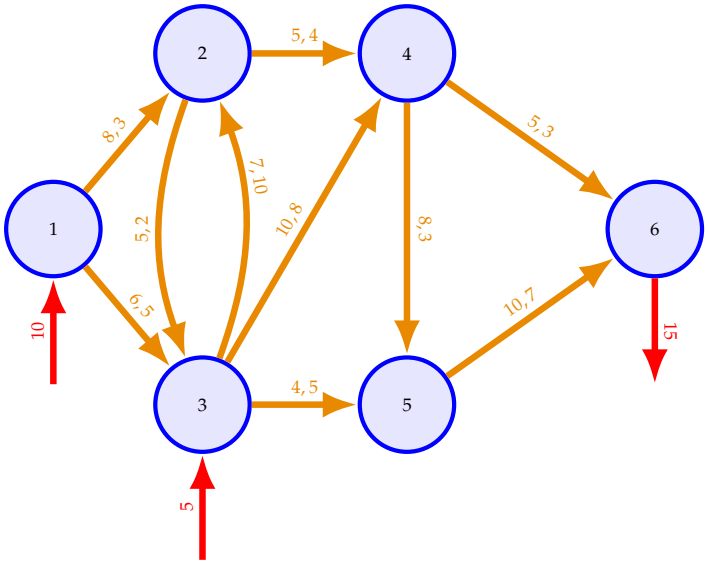


Figure 12.7: Graph representation $G = (\mathcal{V}, \mathcal{E})$ of the water pipeline network of Example 12.4.

We start tackling the problem by defining our sets: $\mathcal{N} = \{1, 2, 3, 4, 5, 6\}$ and $\mathcal{E} = \{(1,2), (1,3), (2,3), (2,4), (3,2), (3,4), (3,5), (4,5), (4,6), (5,6)\}$. In addition, $\mathcal{S} = \{1, 3\}$ and $\mathcal{T} = \{6\}$. We model this multiple-source single-sink MCF problem as:

$$\begin{aligned} \min \; & 3x_{1,2} + 5x_{1,3} + 2x_{2,3} + 4x_{2,4} + 10x_{3,2} \\ & + 8x_{3,4} + 5x_{3,5} + 3x_{4,5} + 3x_{4,6} + 7x_{5,6} \end{aligned} \tag{12.41}$$

s.t.:

$$x_{1,2} + x_{1,3} = 10 \tag{12.42}$$
$$x_{2,3} + x_{2,4} - x_{1,2} - x_{3,2} = 0 \tag{12.43}$$
$$x_{3,2} + x_{3,4} + x_{3,5} - x_{1,3} - x_{2,3} = 5 \tag{12.44}$$
$$x_{4,5} + x_{4,6} - x_{2,4} - x_{3,4} = 0 \tag{12.45}$$
$$x_{5,6} - x_{3,5} - x_{4,5} = 0 \tag{12.46}$$
$$-x_{4,6} - x_{5,6} = -15 \tag{12.47}$$
$$x_{1,2} \leq 8, x_{1,3} \leq 6, x_{2,3} \leq 5, x_{2,4} \leq 5, x_{3,2} \leq 7, \\ x_{3,4} \leq 10, x_{3,5} \leq 4, x_{4,5} \leq 5, x_{4,6} \leq 5, x_{5,6} \leq 10 \tag{12.48}$$

The optimal solution (all values are divided by 1,000) is $x_{1,2} = 5$, $x_{1,3} = 5$, $x_{2,4} = 5$, $x_{3,4} = 6$, $x_{3,5} = 4$, $x_{4,5} = 6$, $x_{4,6} = 5$, and $x_{5,6} = 10$, for an overall cost of 231 monetary units. We represent the final solution in Figure 12.8, where we highlight in green the used edges and report the used capacity and remaining capacity per edge using the same notation as in Figure 12.3.

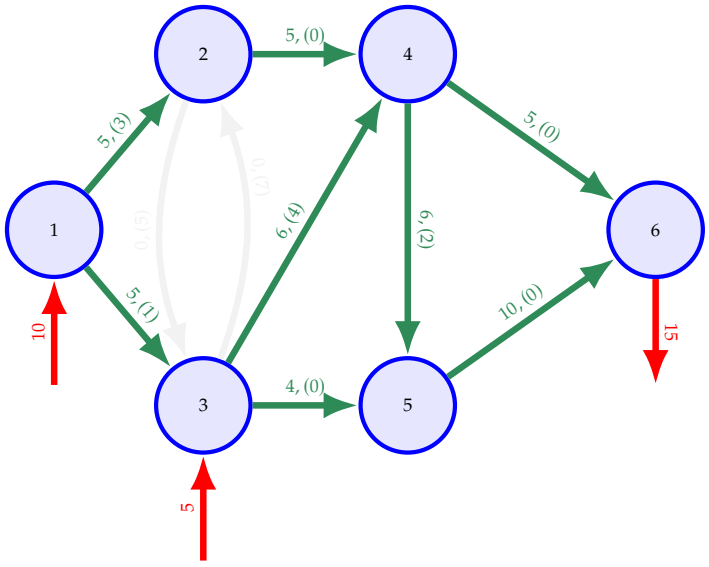


Figure 12.8: Final solution for Example 12.4.

Coded example

The code used to model and solve Example 12.4 is available here.

12.4 Graph coloring problem

The graph coloring problem is a fascinating network problem with various real-world applications. In its most general form, it involves an undirected graph $G = (\mathcal{V}, \mathcal{E})$[11], where the objective is **to determine the smallest number of colors needed to color the vertices such that no two adjacent vertices share the same color**. For the sake of notation, we will define each edge $e \in \mathcal{E}$ as $e = (v_1, v_2)$ where v_1 and v_2 are the vertices where e is incident to.

11: Because this model is described in a very generic fashion here, we stick with the graph-oriented notation of *vertices* and *edges*.

The mathematical model, on top of $G = (\mathcal{V}, \mathcal{E})$, needs a set of colors $\mathcal{C}$. **For graphs that are not too large, we can define as many distinct colors as $|\mathcal{V}|$. This upper bound covers the worst-case scenario of a complete graph, as described in Chapter 11, where every vertex is connected to every other and hence we need $|\mathcal{V}|$ distinct colors to satisfy our constraint**. We will see later that this approach is doomed to severely underperform if larger graphs are involved, and we will describe an approach to tackle this issue.

No specific parameter is needed for a generic graph coloring problem. The first decision variable that is needed is $x_{vc} \in \{0, 1\}$, which takes a unitary value if vertex $v \in \mathcal{V}$ is assigned color $c \in \mathcal{C}$. We also define $y_c \in \{0, 1\}$, which takes a unitary value if color $c \in \mathcal{C}$ is used in the final solution. We store all the notation needed for the graph coloring problem in Table 12.17.

Table 12.17: Notation for the graph coloring problem.

Sets and indices	
$\mathcal{V}$	set of vertices $v \in \mathcal{V}$
$\mathcal{E}$	set of edges $e \in \mathcal{E}$
$\mathcal{C}$	set of colors $c \in \mathcal{C}$
Variables	
$x_{vc} \in \{0, 1\}$	unitary if vertex v is colored with color c
$y_c \in \{0, 1\}$	unitary if color c is used

The graph coloring problem is a BP defined as:

$$\min \sum_{c \in \mathcal{C}} y_c \tag{12.49}$$

s.t.:

$$\sum_{c \in \mathcal{C}} x_{vc} = 1 \qquad \forall v \in \mathcal{V} \tag{12.50}$$

$$x_{vc} \leq y_c \qquad \forall v \in \mathcal{V}, c \in \mathcal{C} \tag{12.51}$$

$$x_{v_1 c} + x_{v_2 c} \leq 1 \qquad \forall e = (v_1, v_2) \in \mathcal{E}, c \in \mathcal{C} \tag{12.52}$$

$$x_{vc} \in \{0, 1\} \qquad \forall v \in \mathcal{V}, c \in \mathcal{C} \tag{12.53}$$

$$y_c \in \{0, 1\} \qquad \forall c \in \mathcal{C} \tag{12.54}$$

where (12.49) minimizes the number of selected colors, (12.50) ensures that every vertex is assigned to exactly one color, (12.51) labels a color as used as soon as it is assigned to a vertex, and (12.52) prevents two connected vertices from being assigned to the same color c. Finally, (12.53)-(12.54) define the binary nature of the two decision variable sets. The minimum number of colors needed to color graph G is also known as its **chromatic number**. Some practical applications of the graph coloring problem are:

- **lecture hall timetabling**. Graph coloring is employed in creating class schedules for schools and universities. Each class corresponds to a vertex with constraints such as room availability and teacher

availability represented as edges. The goal is to assign time slots to classes so that no two conflicting classes share the same time slot. A variant entails scheduling exams so that students can participate in all the exams they have planned without any scheduling conflicts;
- **frequency assignment in telecommunications**. Graph coloring is utilized in frequency assignment problems to assign frequencies to transmitters so that adjacent transmitters operate on different frequencies to avoid interference;
- **map coloring**. In cartography, maps featuring distinct regions should be colored so neighboring regions have distinct colors to avoid misinterpretation.

We illustrate the graph coloring problem with Example 12.5, where the application is map coloring and hence directly reflects the essence of the problem's name.

Example 12.5 *We consider the map of the United States with its 50 states and the federal district of Washington D.C., hence considering 51 states overall (with a slight abuse of notation, we consider the federal district of Washington D.C an additional state). We want to determine the minimum number of colors needed to color each of the 51 regions so that no two contiguous regions share the same color.*

To tackle the problem, the initial step is to represent the 51 states as a graph $G = (\mathcal{V}, \mathcal{E})$. The definition of the set of vertices is quite straightforward, as each state defines one $v \in \mathcal{V}$. Hence, $|\mathcal{V}| = 51$. For the set of edges $\mathcal{E}$, we analyze the boundaries of different states to determine which states border each other. This information is obtained from *Adjacency List of States of the United States (US)* 2024, where for each state the list of neighboring states is provided. We modeled each pair of neighboring states as an edge $e \in \mathcal{E}$ and ensured duplicates were removed, as G is undirected. We identified 109 neighboring relationships, hence $|\mathcal{E}| = 109$. The resulting graph $G = (\mathcal{V}, \mathcal{E})$ is depicted in Figure 12.10, where we used the centroid (in terms of longitude and latitude) of each state to locate each vertex. Additionally, each vertex is labeled with the postal code of its respective state (e.g., the state of California is CA, the state of Nevada NV, etc.). Some insights from Figure 12.10 are:

- Alaska (AK) and Hawaii (HI) are nodes with a degree of 0 (recall Chapter 11) and hence G is not connected. This correctly reflects the geography of the United States of America;
- vertices in the mid-West seem to connect with more vertices rather than vertices along the two coasts. In particular, states such as Missouri (MO) and Colorado (CO) share boundaries with many other states (respectively 8 and 7);
- related to the previous point, if we isolate the subgraph comprising Utah (UT), Colorado (CO), New Mexico (NM), and Arizona (AZ), we notice that it is a complete graph, with each of the four states sharing boundaries with the other three[12]. **Hence, at least 4 colors are needed for this problem**.

Alaska and Hawaii have no influence on the solution. In fact, we can solve the graph coloring for the remaining 49 states and then randomly assign a color to both from the set of colors selected. The number of vertices of the graph is hence reduced to $|\mathcal{V}| = 49$. Given the relatively small size

12: This is not a coincidence, but a natural consequence of the boundaries of those states converging into the *Four corners monument* (see Figure 12.9). We refer interested readers to this Wikipedia page for more information.



Figure 12.9: The Four corners monument at the intersection of Utah (UT), Colorado (CO), New Mexico (NM), and Arizona (AZ).

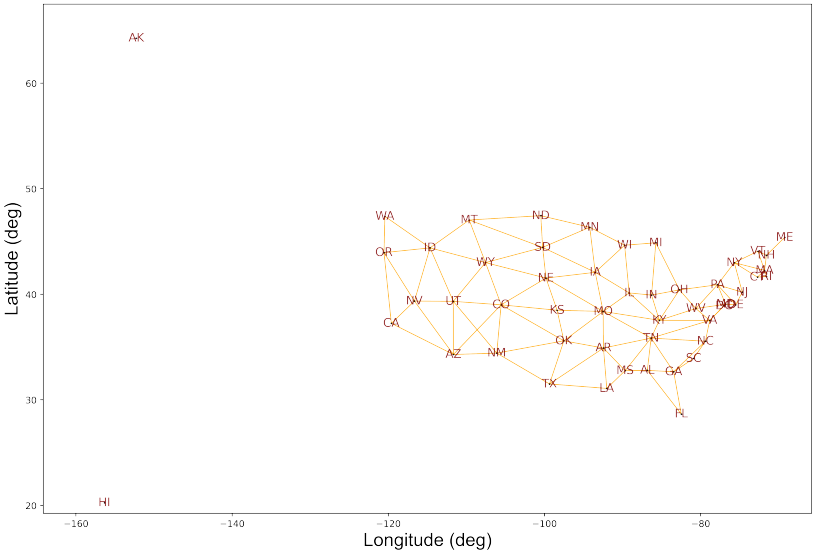


Figure 12.10: Graph representation of the 51 states forming the United States of America in Example 12.5.

of the graph ($|\mathcal{V}| = 49$, $|\mathcal{E}| = 109$), we assume the worst-case scenario and define a set $\mathcal{C}$ of colors with $|\mathcal{C}| = 49$. We will probably need far fewer colors than that, and it might not be a good idea algorithmically to employ such a risk-averse approach, but we will elaborate more about this later in the context of this example.

We then employ the BP formulation of (12.49)-(12.54) and the final solution suggests that we only need 4 colors, out of the 49 we initially designed, to color the 49 states (plus Alaska and Hawaii) in a way that no neighboring states share the same color. The final solution is reported in Figure 12.11, where we decided to color the name of each state rather than the full state for a neater visualization. Because of the small size of states along the East Coast, the coloring is less evident there.

We also report the states divided by color:

- **color 1**: Alabama (AL), Connecticut (CT), Washington D.C. (DC), Iowa (IA), Kansas (KS), Kentucky (KY), Louisiana (LA), Maine (ME), North Carolina (NC), North Dakota (ND), New Jersey (NJ), New Mexico (NM), Nevada (NV), Vermont (VT), Washington (WA), Wyoming (WY);
- **color 2**: Delaware (DE), Florida (FL), Illinois (IL), Michigan (MI), Minnesota (MN), Montana (MT), Nebraska (NE), New Hampshire (NH), New York (NY), Oklahoma (OK), Oregon (OR), Rhode Island (RI), South Carolina (SC), Tennessee (TN), Utan (UT), West Virginia (WV);
- **color 3**: California (CA), Colorado (CO), Georgia (GA), Idaho (ID), Massachussets (MA), Maryland (MD), Missouri (MO), Mississippi (MS), Ohio (OH), South Dakota (SD), Texas (TX), Wisconsin (WI);
- **color 4**: Arkansas (AR), Arizona (AZ), Indiana (IN), Pennsylvania (PA), Virginia (VA)

Eventually, only 4 colors are needed in the context of this graph coloring problem, where any of the 4 (or two distinct ones) can be used to "color" the disconnected Alaska and Hawaii. As expected, we were overly-conservative with our initial selection of $|\mathcal{C}| = 49$ potential colors. Because the optimal solution states that four colors are needed, but not which ones out of the pool of 49, any combination of 4 out of the 49 is

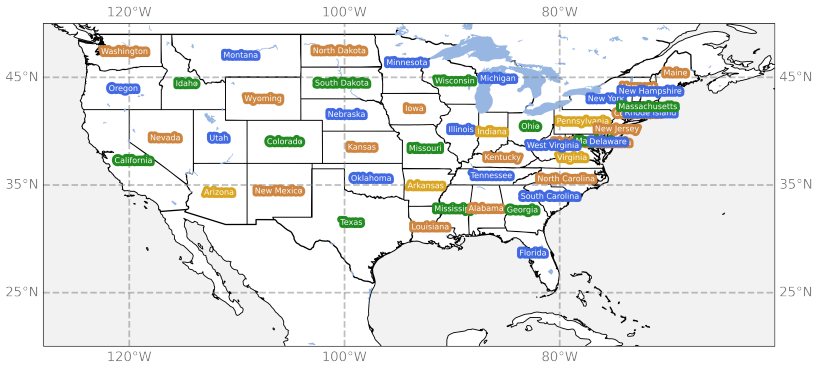


Figure 12.11: Final solution to the graph coloring model applied to the 51 states forming the United States of America in Example 12.5.. Four colors are sufficient to have any two neighboring states with different colors.

"equally optimal". This implies a staggering $\frac{49!}{45!4!} = 211,876$ equivalent (or symmetrical) solutions. For example, solution $\{1, 7, 12, 35\}$ is as good as solution $\{3, 27, 40, 49\}$ as they both entail four distinct colors from $\mathscr{C}$. **While this relatively high number of symmetrical solutions might not pose challenges for this small-scale problem, it might hinder convergence to an optimum for larger problems**.

We can devise a strategy to avoid symmetries by imposing a constraint that allows a color to be used only if colors with lower indices are also used. This constraint can be reformulated as follows: we sort our set of colors $\mathscr{C}$ in descending order of preference, with color $c = 1$ being the favorite and color $c = |\mathscr{C}|$ the least favorite. **Then, we enforce the model to select colors in order of preference**. For instance, if the model needs seven colors for a specific graph coloring application, it should choose the first seven colors from our list, representing our seven favorite colors[13]. Assuming that set $\mathscr{C}$ is defined with increasing indices as $\{1, 2, \cdots, |\mathscr{C}|\}$, we can implement symmetry-breaking constraints with the additional constraint set:

13: This strategy prevents the BB decision tree, as described in Chapter 7, from exploring regions of the solution associated with symmetrical solutions.

$$y_c \leq y_{c-1} \qquad \forall c \in \mathscr{C} \setminus \{1\} \tag{12.55}$$

(12.55) implies that we can use color 1 (our "favorite") with no restrictions, but we can only use color c if the previous color in the set is used ($y_c \leq 1$) while we cannot use color c if the previous one is not used ($y_c \leq 0 \rightarrow y_c = 0$). For example, if color 6 is not used ($y_6 = 0$), then the model is forcing $y_7 \leq 0 \rightarrow y_7 = 0$. This, in turn, forces every other color with an index greater to 7 not to be used as well. We elaborate an alternative solution strategy to the graph coloring problem, especially effective for large-scale problems, in the **An alternative solution method for the graph coloring problem** box.

Coded example

The code used to model and solve Example 12.5 is available here.

An alternative solution method for the graph coloring problem

We conclude this example by proposing an alternative strategy to manage the potentially high number of colors while adhering to our color preferences. Given that most decision variables and constraints in a graph coloring problem scale with the number of potential colors $|\mathscr{C}|$, limiting this value is crucial. However, pinpointing the exact value of $|\mathscr{C}|$ to solve the problem eliminates the need for graph coloring altogether.
To address this, we can employ an iterative approach inspired by the **divide-and-conquer** paradigm. We start with a conservative estimate for $|\mathscr{C}|$ and attempt to solve the model. If successful, we halt the process. If the model proves infeasible due to an underestimated $|\mathscr{C}|$, we increment $|\mathscr{C}|$ by adding at least one color and attempt to solve the model again. This iterative process continues until we find a feasible solution. **While this approach entails solving multiple models, each iteration should be relatively faster than setting $|\mathscr{C}|$ equal to $|\mathscr{V}|$ or an excessively large upper bound that could hinder convergence or lead to memory usage issues**.

12.5 Shortest Path (SP) problem

The Shortest Path (SP) problem is one of the most common and widespread OR models. **Its portfolio of applications is extremely wide, ranging from navigation systems to transportation networks, robotics path planning, and supply chain management**, just to cite a few examples. The main setting entails a directed graph $G = (\mathscr{V}, \mathscr{E})$, where each $e \in \mathscr{E}$ is characterized by a cost C_e. Note that the cost C_e depends on the nature of the problem at hand and is key in capturing and describing the essence of it.

The term “shortest" in the SP model might seem overly restrictive, as the model actually seeks to find the path with the minimum cost from an origin $s \in \mathscr{V}$ to a destination $t \in \mathscr{N}$[14]. Here, C_e represents the cost associated with traversing edge e, which could represent various metrics such as distance, time, or monetary value.

14: We use s to represent the origin and t to represent the destination because they serve the same purpose as the source and sink in the maximum flow problem and MCF problem.

While in some cases, the minimum cost path may indeed be the shortest in terms of distance, “minimum cost" can have broader implications. **For instance, consider routing through a congested road network on the way home. If the objective is to reach home as quickly as possible, the minimum cost path (the most efficient in terms of time) might not necessarily be the shortest one in terms of distance**.

In a general SP, we do not consider edges as capacitated. We are not interested in dispatching flows in a capacitated network, but to efficiently route inside the network from s to t. To this avail, one set of decision variables is needed: $x_e \in \{0, 1\}$ which takes a unitary value if edge e is used in the solution. We report all the notation needed for the SP problem in Table 12.18.

The SP problem is a BP[15] defined as:

15: In this version of the book, we only focus on the BP implementation of the SP problem, but there are other approaches. The most famous approach overall is the Dijkstra algorithm. We refer interested readers to this Wikipedia page.

Table 12.18: Notation for the SP problem.

Sets and indices	
$\mathcal{V}$	Set of vertices $v \in \mathcal{V}$
$\mathcal{E}$	Set of edges $e \in \mathcal{E}$
Parameters	
C_e	cost of edge $e \in \mathcal{E}$
Variables	
$x_e \in \{0,1\}$	unitary if edge e is used in the SP

$$\min \sum_{e \in \mathcal{E}} C_e x_e \tag{12.56}$$

s.t.:

$$\sum_{e \in \delta_i^-} x_e - \sum_{e \in \delta_i^+} x_e = \begin{cases} 1, & i = s \\ -1, & i = t \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in \mathcal{V} \tag{12.57}$$

$$x_e \in \{0,1\} \qquad \forall e \in \mathcal{E} \tag{12.58}$$

(12.56) defines the objective function, i.e., to minimize the overall cost of the used edges. (12.57) is a flow conservation constraint split into three parts according to the specific vertex considered. For the source s, we impose to leave such a node via one of the edges outbound from it $\sum_{e \in \delta_s^-} x_e - \sum_{e \in \delta_s^+} x_e = 1$ is equivalent to $\sum_{e \in \delta_s^-} x_e = 1$. For the sink t, we impose to reach such a node via one of the edges inbound to it $\sum_{e \in \delta_t^-} x_e - \sum_{e \in \delta_t^+} x_e = -1$ is equivalent to $\sum_{e \in \delta_t^+} x_e = 1$. For every other node, the net flow should be zero. (12.58) defines the binary nature of the decision variables.

Example 12.6 *A group of friends is organizing a road trip from Austin (TX) to Raleigh (NC). Along the trip towards East, the group plans potential visits to the following state capitals: Little Rock (AK), Baton Rouge (LA), Nashville (TN), and Montogomery (AL). Despite the keen interest on the beautiful South-East states of the United States of America, the group plans to arrive at the destination following the path with the shortest distance. The potential connections between the state capitals that the group considers are shown in Figure 12.12, where the numbers represent the distance (in miles) between cities. The goal is to formulate the problem as a SP problem and find the optimal solution.*

In Figure 12.12, on top of the the postal code of the state, we add a numerical index for the sake of simplicity: $TX \rightarrow 1, AK \rightarrow 2, LA \rightarrow 3, TN \rightarrow 4, AL \rightarrow 5, NC \rightarrow 6$. Hence, $s = 1$ and $t = 6$. We define the SP problem as:

$$\begin{aligned} \min \; & 446x_{1,2} + 432x_{1,3} + 343x_{2,3} + 347x_{2,4} + 463x_{2,5} \\ & + 587x_{3,4} + 365x_{3,5} + 539x_{4,6} + 281x_{5,4} + 569x_{5,6} \end{aligned} \tag{12.59}$$

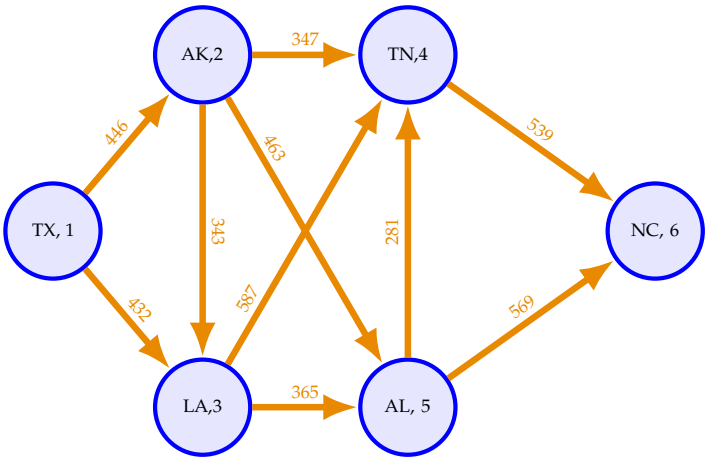


Figure 12.12: Graph representation $G = (\mathcal{V}, \mathcal{E})$ of the six state capitals ($\mathcal{V}$) and their connections ($\mathcal{E}$) of Example 12.6.

s.t.:

$$x_{1,2} + x_{1,3} = 1 \tag{12.60}$$
$$x_{1,2} - x_{2,3} - x_{2,4} - x_{2,5} = 0 \tag{12.61}$$
$$x_{1,3} + x_{2,3} - x_{3,4} - x_{3,5} = 0 \tag{12.62}$$
$$x_{2,4} + x_{3,4} + x_{5,4} - x_{4,6} = 0 \tag{12.63}$$
$$x_{2,5} + x_{3,5} - x_{5,4} - x_{5,6} = 0 \tag{12.64}$$
$$-x_{4,6} - x_{5,6} = -1 \tag{12.65}$$
$$x_{1,2}, \cdots, x_{5,6} \in \{0, 1\} \tag{12.66}$$

We wrote the flow conservation constraints following the numerical indexing, hence (12.60) relates to vertex 1 (TX) and (12.65) refers to vertex 6 (NC). Solving the problem yields the following routing: Austin (TX) → Little Rock (AK) → Nashville (TN) → Raleigh (NC), for an overall distance of 1,332 miles.

Coded example

The code used to model and solve Example 12.6 is available here.

12.6 Minimum Spanning Tree (MST) problem

The Minimum Spanning Tree (MST) problem is another widespread problem in OR. Its mathematical representation entails an undirected graph $G = (\mathcal{V}, \mathcal{E})$, with every $e \in \mathcal{E}$ characterized by a cost C_e. The goal of the MST problem is to connect all the vertices of G without any cycles[16] and with the minimum possible total edge cost. **Hence, among all the possible spanning trees of $G = (\mathcal{V}, \mathcal{E})$, we are interested in looking for the one with the minimum cumulative cost**.

In the SP problem, the focus lies on efficient traversal from a designated origin to a destination within a graph, emphasizing directionality as a key factor. Conversely, the adoption of an undirected graph in the MST problem stems from the goal of establishing comprehensive connections

16: We recollect some notation from Chapter 11 here. A spanning tree is a subgraph of G that features all the original vertices and a subset of edges so that every pair of vertices is connected by a single path only.

among vertices, rather than prioritizing specific paths between individual nodes. We use this last statement to list some applications of the MST problem: **transportation networks** (optimal design of road or railway lines that maximizes connectivity between locations while limiting infrastructure costs), **power grid design** (layout of power lines to connect different substations or power generation sources while minimizing the total length of wire needed), **cluster analysis** (constructing a tree that connects all data points with the minimum total edge weight can be leveraged to efficiently cluster such points), etc.

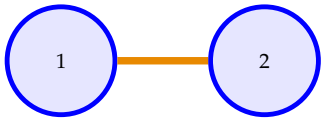
In the academic literature, several variants of the MST exist. We adopt the first version presented in *Columbia University: IEOR 6614 course notes 2024*[17], but after providing a couple of preliminary insights. We start by intuitively proving that in an undirected graph $G = (\mathcal{V}, \mathcal{E})$ **exactly $|\mathcal{E}-1|$ edges are needed to form a spanning tree**. In Figure 12.13 we provide three examples of spanning trees for graphs with $|\mathcal{V}| = 2$ (Figure 12.13a), $|\mathcal{V}| = 3$ (Figure 12.13b), and $|\mathcal{V}| = 4$ (Figure 12.13c) vertices. While in Figure 12.13a that is the only spanning tree existing, different options are available when $|\mathcal{V}| = 3$ or $|\mathcal{V}| = 4$ and our goal would be to find the "cheapest" one. In all three cases, if we start from any vertex we can reach any other vertex within a limited number of steps (the spanning tree indeed defines a connected graph) and no cycles are present. Therefore, we might naturally seek the optimal spanning tree by formulating the objective function as minimizing the sum of costs $\sum_{e\in\mathcal{E}} C_e x_e$, while enforcing the constraint $\sum_{e\in\mathcal{E}} x_e = |\mathcal{E}| - 1$. **In essence, among all the conceivable combinations of $|\mathcal{E}| - 1$ edges in G, our objective is to identify the combination with the lowest total cost**.

17: As will become apparent later, this formulation is called *subtour elimination formulation*, as opposed to the alternative *cutset formulation*. We refer readers to the suggested reference for the other formulation.

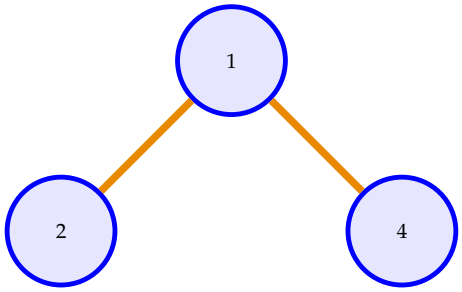
We now show why this approach might be doomed to fail under specific circumstances with the example of Figure 12.14. Figure 12.14a displays a graph with $|\mathcal{V}| = 4$ nodes and $|\mathcal{E}| = 5$ edges, where the number on each edge e represents C_e. Following our previous intuition, if we construct a mathematical model searching for the combination of $|\mathcal{V}| - 1 = 3$ edges with the smallest cumulative edge cost, the solution we obtained is the one reported in Figure 12.14b, where edges $(1, 2)$, $(1, 4)$, and $(2, 4)$ are selected for a cumulative cost of 6. **This solution is not a proper minimum spanning tree for two reasons: vertex 3 is disconnected and hence not reachable from any other vertex and vertices 1,2, and 4 form a cycle**.

This does not mean we were completely wrong, but that we need an additional set of constraints to avoid such a situation from occurring. More precisely, we need to prevent **subtours** (which is another term for cycles) from being selected by the model. To do so, we first need to define set $\mathcal{S}$. This set contains, given a graph $G = (\mathcal{V}, \mathcal{E})$, all the unique combinations of vertices ranging from 3 vertices up to $|\mathcal{V}| - 1$ vertices. Such a set plays an active role only for graphs where $|\mathcal{V}| \geq 4$ (as in Figure 12.14). At least 3 edges are needed to form a cycle, and we specified that a minimum spanning tree for a graph with $|\mathcal{V}|$ vertices is composed by $|\mathcal{V}| - 1$ edges. Hence, as shown by Figure 12.13a and Figure 12.13b, minimum spanning trees in graphs where $|\mathcal{V}| \leq 3$ cannot feature subtours anyway.

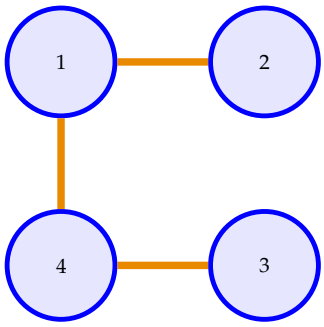
Considering again Figure 12.14, we showcase how to compute set $\mathcal{S}$ and its relevance. Because $|\mathcal{V}| = 4$ and we need to find combinations of vertices from 3 up to $|\mathcal{V}| - 1$, we only need to find combinations of 3 vertices out of



(a) $|\mathcal{V}| = 2$.

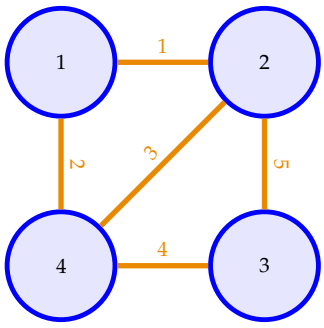


(b) $|\mathcal{V}| = 3$.

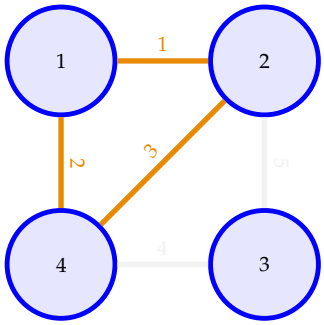


(c) $|\mathcal{V}| = 4$.

Figure 12.13: Examples of spanning trees for graphs with $|\mathcal{V}| = 2$, $|\mathcal{V}| = 3$, and $|\mathcal{V}| = 4$ vertices.



(a) Original graph $G = (\mathcal{V}, \mathcal{E})$.



(b) Incorrect MST where the only condition enforced is $\sum_{e \in \mathcal{E}} x_e = |\mathcal{V}| - 1$.

Figure 12.14: Example of a wrong mathematical modeling approach to compute a MST.

4. Hence, $\mathcal{S} = \{(1,2,3),(1,2,4),(1,3,4),(2,3,4)\}$. Additionally, for every combination $s \in \mathcal{S}$ we need to store $\mathcal{E}(s) \subseteq \mathcal{E}$, i.e., the subset of edges where both vertices belong to s. In our case, $\mathcal{E}((1,2,3)) = \{(1,2),(2,3)\}$, $\mathcal{E}((1,2,4)) = \{(1,2),(1,4),(2,4)\}$, $\mathcal{E}((1,3,4)) = \{(1,4),(3,4)\}$, and $\mathcal{E}((2,3,4)) = \{(2,3),(2,4),(3,4)\}$.

The last step of this process is to define the constraint set:

$$\sum_{e \in \mathcal{E}(s)} x_e \leq |s| - 1 \qquad \forall s \in \mathcal{S} \tag{12.67}$$

12.67 enforces that, given a subset $s \in \mathcal{S}$ (with $|s|$ vertices) in the graph, at most $|s| - 1$ edges that both originate and end in vertices part of s can be chosen. Imposing such a constraint set, we avoid the situation shown in Figure 12.14b because, for $s = \{(1,2,4)\}$, we impose that at most 2 edges among $(1,2)$, $(1,4)$, and $(2,4)$ can be chosen when determining the MST.

In practice, (12.67) plays a constraining role only for those $s \in \mathcal{S}$ where a subtour can occur in the first place, i.e., for those subsets s where $|\mathcal{E}(s)| = |s|$. In our example, this is the case for $s = \{(1,2,4)\}$, as shown in Figure 12.14b, and $s = \{(2,3,4)\}$, where without the constraint the simultaneous selection of edges $(2,3)$, $(2,4)$, and $(3,4)$ would be allowed. With (12.67), we impose instead $x_{2,3} + x_{2,4} + x_{3,4} \leq 2$, hence preventing that subtour from occurring. On the other hand, let us take the example of $s = \{1,2,3\}$. We see from Figure 12.14a that those 3 vertices alone cannot form a cycle on their own. This is confirmed if we apply (12.67) to $s = \{1,2,3\}$. Because $\mathcal{E}((1,2,3)) = \{(1,2),(2,3)\}$, the constraint becomes $x_{1,2} + x_{2,3} \leq |\{1,2,3\}| - 1 = 2 \rightarrow x_{1,2} + x_{2,3} \leq 2$ which is always verified anyway.

We showcase the notation needed for the subtour elimination formulation of the MST in Table 12.19

Table 12.19: Notation for the MST problem.

Sets and indices	
$\mathcal{V}$	Set of vertices $v \in \mathcal{V}$
$\mathcal{E}$	Set of edges $e \in \mathcal{E}$
$\mathcal{S}$	Set of vertices subsets $s \in \mathcal{S}$ with $3 \leq \lvert s \rvert \leq \lvert \mathcal{V} \rvert - 1$
Parameters	
C_e	cost of edge $e \in \mathcal{E}$
Variables	
$x_e \in \{0,1\}$	unitary if edge e is used in the MST

and define its mathematical BP formulation as:

$$\min \sum_{e \in \mathcal{E}} C_e x_e \tag{12.68}$$

s.t.:

$$\sum_{e \in \mathscr{E}} x_e = |\mathscr{V}| - 1 \tag{12.69}$$

$$\sum_{e \in \mathscr{E}(s)} x_e \leq |s| - 1 \qquad \forall s \in \mathcal{S} \tag{12.70}$$

$$x_e \in \{0, 1\} \qquad \forall e \in \mathscr{E} \tag{12.71}$$

(12.68) minimizes the overall cost of the edges forming the MST. (12.69) defines the number of edges needed for the MST, (12.70) prevents subtours from occurring, and (12.71) defines the binary nature of the decision variables.

We showcase an application of the MST model described in (12.68)-(12.71) using the same initial setting as in Example 12.6.

Example 12.7 *Using the same set of state capitals and edges as in Example 12.6, i.e., Austin (TX), Little Rock (AK), Baton Rouge (LA), Nashville (TN), Montogomery (AL), and Raleigh (NC), the goal is to determine the shortest length of the railway network that allows to reach any of the six capitals from any other. We assume that railway connections are only allowed according to the original set of edges and that the length of each allowed connection is the same as in Example 12.6.*

We realize that this problem can be solved as an MST problem because our task is to find a way to connect all 6 state capitals via a unique path per city pair. Because MSTs are defined on undirected graphs, we highlight the initial setting in Figure 12.15 by removing the arrow tips from all edges.

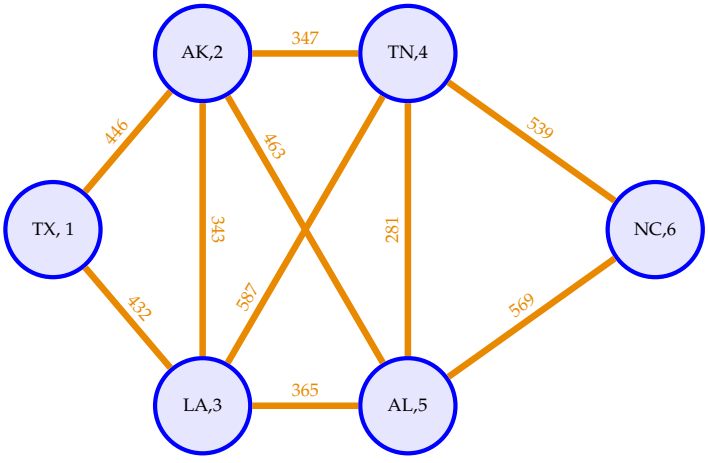


Figure 12.15: Graph representation $G = (\mathscr{V}, \mathscr{E})$ of the six state capitals ($\mathscr{V}$) and their connections ($\mathscr{E}$) for Example 12.7.

Before showcasing the mathematical formulation, we display how to compute set $\mathcal{S}$. Because $|\mathscr{V}| = 6$, we must compute all the combinations of 3, 4, and 5 cities out of 6. Their number is, respectively, $\frac{6!}{3!3!} = 20$, $\frac{6!}{4!2!} = 15$, and $\frac{6!}{5!1!} = 6$, so that $|\mathcal{S}| = 41$. For the sake of conciseness, we do not list all of them, but highlight a couple of relevant cases. If $s = (1, 2, 3)$, then $\mathscr{E}(s) = \{(1, 2), (1, 3), (2, 3)\}$, which results (applying (12.70)) in $x_{1,2} + x_{1,3} + x_{2,3} \leq 2$. If $s = (2, 3, 4, 5)$, then $\mathscr{E}(s) = \{(2, 3), (2, 4), (2, 5), (3, 4)\}$, which results in $x_{2,3} + x_{2,4} + x_{2,5} + x_{3,4} \leq 3$. In both cases, the constraint prevents the model from creating a subtour encompassing the selected vertices.

We formulate the MST problem as:

$$\begin{aligned}\min\ & 446x_{1,2} + 432x_{1,3} + 343x_{2,3} + 347x_{2,4} + 463x_{2,5} \\ & + 587x_{3,4} + 365x_{3,5} + 281x_{4,5} + 539x_{4,6} + +569x_{5,6} \end{aligned} \tag{12.72}$$

s.t.:

$$x_{1,2} + x_{1,3} + x_{2,3} + x_{2,4} + x_{2,5} + x_{3,4} + x_{3,5} + x_{4,5} + x_{4,6} + x_{5,6} = 5 \tag{12.73}$$

$$x_{1,2} + x_{1,3} + x_{2,3} \leq 2 \tag{12.74}$$

$$\dots$$

$$x_{2,3} + x_{2,4} + x_{2,5} + x_{3,4} + x_{3,5} + x_{4,5} + x_{4,6} + x_{5,6} \leq 4 \tag{12.75}$$

$$x_{1,2}, \cdots, x_{5,6} \in \{0,1\} \tag{12.76}$$

(12.72) defines the objective function, i.e., to minimize the overall length of the railway connections. (12.73) imposes that the MST should have 5 edges in this specific instance, and (12.74)-(12.75) define the first ($s = (1,2,3)$) and last ($s = (2,3,4,5,6)$) of the 41 subtour elimination constraints. (12.75) defines the binary nature of the decision variables.

The resulting solution suggests that the MST should be formed by the following edges: $(1,3)$, $(2,3)$, $(2,4)$, $(4,5)$, and $(4,6)$, for an overall length of the railway system of $1,942$ miles. Such a solution is depicted in Figure 12.16.

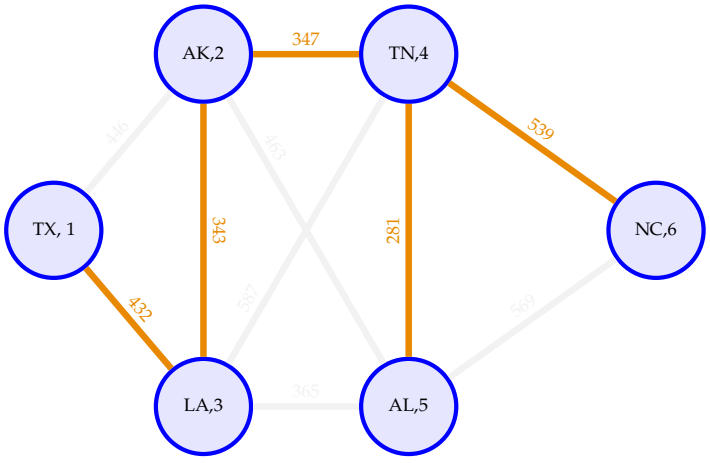


Figure 12.16: Resulting MST connecting the 6 state capitals.

Given the strong symmetry of the original graph $G = (\mathcal{V}, \mathcal{E})$, it is worth discussing the resulting MST and interpreting such a solution. The solution incentivizes connecting TX to the "main block" formed by AK, LA, TN, and AL by selecting the shortest edge (TX, LA) rather than (TX, AK). Following the same logic, NC connects to the main block via TN as it is a shorter connection than via AL. Then, to complete the MST the shortest 3 remaining edges, namely (AK, LA), (AK, TN), and (TN, AL) are chosen so that connectivity properties are ensured while minimizing the overall distance. North-south movements are shorter using the (AK, LA) and the (TN, AL) connections rather than

the longer (AK, AL) and (LA, TN) ones. Additionally, the solution would still be an MST is (LA, AL) was chosen instead of (AK, TN) to allow east-west movements, but as connection (AK, TN) is 18 miles shorter than (LA, AL), the model selects the first one.

Coded example

The code used to model and solve Example 12.7 is available here.

In the **An alternative way of dealing with subtour elimination constraints in the MST problem** box we describe an alternative approach to solve an MST which avoids full enumeration of all potential subtours upfront.

An alternative way of dealing with subtour elimination constraints in the MST problem

Readers with a penchant for combinatorics might have realized that defining all subtour elimination constraints upfront might be a daunting task. This is especially true for larger graphs $G = (\mathcal{V}, \mathcal{E})$ with respect to the one presented in Example 12.7. Let us take a slightly larger graph where $|\mathcal{V}| = 20$. Enumerating all the combinations of vertices from 3 up to 19 yields

$$\frac{20!}{17!3!} + \frac{20!}{16!4!} + \cdots + \frac{20!}{2!18!} + \frac{20!}{1!19!} = 1,140 + \cdots + 20 = 1,048,364$$

combinations already. For graphs where $|\mathcal{V}|$ is in the range of the hundreds or thousands (quite common in practical applications), the number of combinations spikes even more. **Adding so many constraints to the model might affect negatively the solution time, especially given that only a fraction of the "theoretically needed" subtour elimination constraints is really needed for a mathematically feasible solution**.

This is where the concept of **lazy constraints** comes into play. The idea is to discard initially any subtour elimination constraint and solve the MST problem with just constraint (12.69). Then the solution is analyzed and, if there are no subtours, that solution is accepted. **Otherwise, the constraints that prevent the identified subtours from occurring are added and the problem is solved again until a certain iteration is reached where the solution is subtour-free. This approach of adding constraints is labeled "lazy" because constraints are not added upfront, but only as needed to fix some infeasible behavior (the subtours)**. This process might entail solving an updated problem many times rather than once. Notwithstanding, each problem (albeit growing in size due to the added lazy constraints) is much smaller than the original problem where all subtour elimination constraints are present, and hence the process should be algorithmically more efficient. Considering again Figure 12.14, adopting this approach the first MST we would compute would be the one represented in Figure 12.14b, which features a subtour. Hence, we would add to the model the additional constraint $x_{1,2} + x_{1,4} + x_{2,4} \leq 2$ and solve the model again.

Routing problems 13

13.1 Traveling Salesman Problem (TSP) 251
13.2 Solution methods for the TSP 254
13.3 Vehicle Routing Problem (VRP) 255
13.4 Widely-used VRP variants 258
13.4.1 Vehicle Routing Problem with Time Windows (VRPTW) 258
13.4.2 Split Delivery Vehicle Routing Problem (SDVRP) 260
13.4.3 Multi-depot VRP . . . 262
13.4.4 Pickup and Delivery Problem (PDP)s 263
13.5 Further VRP variants . 264

> "Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop."
>
> Lewis Carroll

Routing problems are at the core of various collection and distribution systems which directly and indirectly are part of our lives. **Formally, they are network problems as well since they are defined over a graph** $G = (\mathcal{V}, \mathcal{E})$. Nevertheless, there have been substantial developments specific to the nature of routing problems in terms of the formulation and solution methods which is the reason why we cover them in a dedicated chapter.

For routing problems, the Traveling Salesman Problem (TSP) is the building block that carries the core complexity and is easy to explain. Consider a salesperson that needs to visit n cities, i.e., nodes, exactly once. This reminds us of a Hamilton cycle, as pointed out in the **A reminder on Hamiltonian cycles** box. The TSP looks for the least costly tour to achieve this. The visit may be related to delivering packages, containers, etc., or collecting them such as the case of waste collection. We refer to Lawler et al. (1985) for an overview of the historical development of the TSP. The TSP is generalized with different variants under the class of Vehicle Routing Problem (VRP)s. The main generalization is that more than one vehicle is available, which essentially means that we can have multiple tours. We refer to Toth and Vigo (2002) for an overview of VRPs and the different variants.

In this chapter, we will first cover the formulation and solution methods for a TSP in Section 13.1-13.2 and then discuss VRPs in Section 13.3-Section 13.5.

A reminder on Hamiltonian cycles

A TSP looks for a tour that visits each node, i.e., vertex, exactly once. As a reminder (see Section 11.2), such tours are referred to as Hamiltonian cycles. A graph that contains a Hamiltonian cycle is then a Hamiltonian graph.

13.1 Traveling Salesman Problem (TSP)

A TSP is defined over a graph where each city to be visited is denoted by a node and two nodes are connected by arcs (or edges). The set of nodes is given by $\mathcal{N}$. The objective is to have the least costly tour, where the cost of traveling from a city i to city j is given by C_{ij}. This cost can be representing the distance, time, or indeed a monetary cost or a combination of them in the form of a generalized cost. For formulating

the problem, we need to introduce a set of binary decision variables as follows:

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i,j) \text{ is in the tour} \\ 0 & \text{otherwise.} \end{cases} \tag{13.1}$$

We report the needed notation for the TSP in Table 13.1.

Table 13.1: Notation for the TSP.

Sets and indices	
$\mathcal{N}$	set of nodes $i, j \in \mathcal{N}$, $\lvert\mathcal{N}\rvert = n$
Parameters	
C_{ij}	cost from node $i \in \mathcal{N}$ to node $j \in \mathcal{N}$
Variables	
$x_{ij} \in \{0,1\}$	unitary if arc (i,j) is in the tour
u_i	order of node $i \in \mathcal{N}$ in the tour

Each of the cities needs to be visited exactly once and this means there will be one arc incoming to each node and one outgoing (see below in (13.3) and (13.4)). Up to now, the TSP resembles an assignment problem from Chapter 9. Yet it needs to be expanded to have subtour elimination constraints to avoid disjoint tours. This can be done in multiple ways and the two most common ones are known as DFJ (Dantzig et al., 1954) and MTZ (Miller et al., 1960) subtour elimination constraints. The problem formulation below utilizes MTZ subtour elimination constraints and is provided as follows:

$$\min \quad \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} C_{ij} x_{ij} \tag{13.2}$$

s.t.:

$$\sum_{i \in \mathcal{N}} x_{ij} = 1 \qquad \forall j \in \mathcal{N} \tag{13.3}$$

$$\sum_{j \in \mathcal{N}} x_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{13.4}$$

$$u_1 = 1 \tag{13.5}$$

$$2 \leq u_i \leq n \qquad \forall i \in \mathcal{N} : i > 1 \tag{13.6}$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad \forall i, j \in \mathcal{N} \tag{13.7}$$

$$x_{ij} \in \{0,1\} \qquad \forall i, j \in \mathcal{N} \tag{13.8}$$

$$u_i \geq 0 \qquad \forall i \in \mathcal{N} \tag{13.9}$$

(13.2) aims at minimizing the cost of the tour. (13.3)-(13.4) ensure that each node has one ingoing and one outgoing arc. The MTZ subtour elimination constraints order the visit of the nodes with the use of the u_i variables. The starting node is given the order of 1 (13.5) and the

remainder of the nodes will be ordered between 2 and n (13.6) based on the routing decision x_{ij} thereafter (13.7). In fact, if $x_{ij} = 1$ we obtain from (13.7) that $u_j \geq u_i + 1 \rightarrow u_j = u_i + 1$. Because j is visited right after i, their ordering is correctly captured by the constraint. (13.8)-(13.9) define the nature of the decision variables.

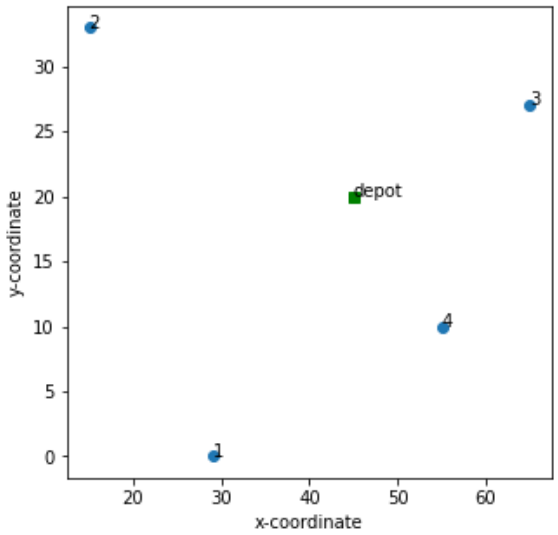


Figure 13.1: An example network for the TSP.

Example 13.1 *Let us now consider an example as given in Figure 13.1 where the coordinates of four cities (blue nodes) and a depot (green square) are provided as a network. Table 13.2 provides the Euclidean distances between the nodes.*

	depot	1	2	3	4
depot		26	33	21	14
1	26		36	45	28
2	33	36		50	46
3	21	45	50		20
4	14	28	46	20	

Table 13.2: Distance matrix for Example 13.1

Considering that the objective is to minimize the total distance, i.e., $\sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} C_{ij} x_{ij}$ with each C_{ij} given in Table 13.2, the optimal solution for this TSP is given in Figure 13.2 with a total distance of 138.

Coded example

The code used to model and solve Example 13.1 is available here.

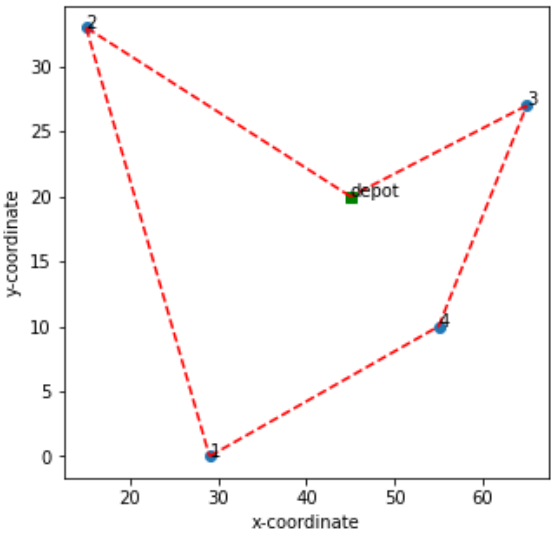


Figure 13.2: Solution to Example 13.1

13.2 Solution methods for the TSP

The formulation provided in (13.2)-(13.9) is a MILP and can be solved in different ways. An exhaustive enumeration for such problems will correspond to $(n-1)!$ permutations considering Hamiltonian cycles. Already a graph with 10 nodes would lead to the enumeration of more than 10^5 cycles, which shows the computational complexity of the problem.

As discussed in Part III, one option is to go for exact methods such as BB which are typically at the backbone of available optimization solvers. There are also various dedicated heuristic algorithms developed for the TSP, which we do not cover in this book.

In any solution method, an initial feasible solution is essential and serves as an upper bound for the problem (in the case of min problems). For obtaining such an initial feasible solution, there are different techniques available. *Nearest Neighbor*, *Farthest Neighbor*, *Nearest Insertion*, and *Farthest Insertion* are listed as the most commonly used ones. These are also referred to as *Tour Construction Heuristics*.

For the TSP provided in Example 13.1 let us apply the Nearest Neighbor method. From the depot, the nearest node is node 4. The nearest one to node 4 is node 3. The nearest one to node 3 is 4, but it will create a subtour so we select the next closest one which is node 1. From node 1 we need to cover the remaining node, which is node 2 and then we go back to the depot. So the nearest neighbor solution is: depot $\rightarrow$ 4 $\rightarrow$ 3 $\rightarrow$ 1 $\rightarrow$ 2 $\rightarrow$ depot which has a total distance of 148. **The Nearest Neighbor method is myopic when it comes to the objective of a minimum tour length. It looks for the nearest node, yet we need to come back to the depot and we may end up traveling longer** (this concept shares similarities with the myopic North-West corner rule we discussed in Chapter 12). In this case, the myopic choice of selecting node 4 at the start of the tour, comes at the cost of having a long-distance edge of $(3, 1)$ later.

13.3 Vehicle Routing Problem (VRP)

VRPs have been developed to generalize the TSP in terms of many assumptions. **First of all, having a fleet of vehicles to perform various transport and logistics activities brings the possibility of having multiple tours in the routing solution. Furthermore, capacity is a core limitation that cannot be avoided for practical applications**. Therefore, the Capacitated Vehicle Routing Problem (CVRP) is one of the most studied VRPs in the literature.

We provide the notation used for the CVRP formulation in Table 13.3. Note that, the terms already introduced for the TSP in Section 13.1 are kept to the same notation except for introducing a vehicle index to some of them. Note that, for the sake of an easier explanation, we are now representing the depot by two nodes at identical locations; 0 represents the start and $n+1$ represents the end. Considering a set of customers $\mathscr{C} = \{1, \cdots, |\mathscr{C}| = n\}$, we can then define the full set of nodes as $\mathcal{N} = \{0, 1, \cdots, n, n+1\}$ where the first and last terms are the nodes representing the same depot (the first as the origin of a tour, the second as the destination of a tour) and all the others are the customer nodes. It also follows that $|\mathcal{N}| = |\mathscr{C}| + 2 = n + 2$. In addition, we do define the ordering decision variables for the origin depot node and customer nodes, but not for the destination depot node.

Table 13.3: Notation for the CVRP.

Sets and indices	
$\mathcal{N}$	Set of nodes $i, j \in \mathcal{N}$, $\|\mathcal{N}\| = n+2$, 0 is the start and $n+1$ is the end node (depot)
$\mathscr{C}$	Set of customer nodes $i, j \in \mathscr{C}$, excluding the depot nodes 0 and $n+1$, $\mathscr{C} \subset \mathcal{N}$
$\mathscr{V}$	Set of vehicles $k \in \mathscr{V}$
Parameters	
C_{ij}	cost (or distance or time) from node $i \in \mathcal{N}$ to node $j \in \mathcal{N}$
D_i	demand of node i
Q_k	capacity of vehicle k
Variables	
$x_{ijk} \in \{0,1\}$	unitary if arc (i,j) is traversed by vehicle k, 0 otherwise
u_{ik}	order of node $i \in \mathcal{N} \setminus \{n+1\}$ in the tour of vehicle k

The mathematical formulation then follows as:

$$\min \quad \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} \sum_{k \in \mathscr{V}} C_{ij} x_{ijk} \tag{13.10}$$

s.t.:

$$\sum_{k\in\mathcal{V}}\sum_{j\in\mathcal{N}} x_{ijk} = 1 \qquad \forall i \in \mathcal{C} \quad (13.11)$$

$$\sum_{i\in\mathcal{C}} D_i \sum_{j\in\mathcal{N}} x_{ijk} \leq Q_k \qquad \forall k \in \mathcal{V} \quad (13.12)$$

$$\sum_{i\in\mathcal{N}} x_{0ik} = 1 \qquad \forall k \in \mathcal{V} \quad (13.13)$$

$$\sum_{i\in\mathcal{N}} x_{ihk} - \sum_{j\in\mathcal{N}} x_{hjk} = 0 \qquad \forall h \in \mathcal{C}, k \in \mathcal{V} \quad (13.14)$$

$$\sum_{i\in\mathcal{N}} x_{i,n+1,k} = 1 \qquad \forall k \in \mathcal{V} \quad (13.15)$$

$$u_{0k} = 1 \qquad \forall k \in \mathcal{V} \quad (13.16)$$

$$2 \leq u_{ik} \leq n+1 \qquad \forall i \in \mathcal{C}, k \in \mathcal{V} \quad (13.17)$$

$$u_{ik} - u_{jk} + (n+1)x_{ijk} \leq n \qquad \forall i,j \in \mathcal{N}\setminus\{n+1\}, i \neq j, k \in \mathcal{V} \quad (13.18)$$

$$x_{ijk} \in \{0,1\} \qquad \forall i,j \in \mathcal{N}, k \in \mathcal{V} \quad (13.19)$$

$$u_{ik} \geq 0 \qquad \forall i \in \mathcal{N}\setminus\{n+1\}, k \in \mathcal{V} \quad (13.20)$$

The objective function (13.10) is similar to the TSP case and minimizes the total cost across all nodes to be visited and the vehicles. Constraints (13.11) ensure that exactly one vehicle visits each of the nodes (except the depot). The capacity of the fleet is maintained by constraints (13.12). Constraints given in (13.14) state that if vehicle k arrives at customer node h, then the same vehicle must leave node h (hence, conservation of flow per vehicle $k \in \mathcal{V}$ is enforced in each customer node). Constraints (13.13) and (13.15) ensure that each vehicle starts in the depot and ends in the depot, respectively. Note that, the vehicles that are not needed to perform any tours can stay in the depot, i.e., can start at node 0 and end at node $n+1$ (which is a tour at zero cost as $C_{0,n+1} = 0$). Constraints (13.16)-(13.18) are the same subtour elimination constraints as before but expanded with the vehicle index k. In (13.17), the lower bound of every node except the origin depot is 2 while the upper bound is $n+1$: these bounds allow to map the case where a single vehicle visits all customers (the last one in the tour inheriting the index $n+1$ because index 1 is assigned to the origin depot via (13.13)) before reaching the destination depot[1]. The coefficients on the left- and right-hand sides are slightly different with respect to (13.7) as now $n+1$ (and not n) nodes are considered. Additionally, in (13.18) it is also made explicit that such constraints are only enforced for different nodes: $i \neq j$. Note that the same logic applies to the objective function and all the other constraints. For example, in (13.10), when writing $\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}$ we imply as well $i \neq j$ as C_{ii} is not defined. For the depot, we fix this problem by fictitiously defining two separate nodes $i = 0$ and $i = n+1$ which allows modeling departures/arrivals from/to the depot and unused trucks that travel between the two nodes at zero cost $C_{0,n+1} = 0$ as mentioned above. The type of the variables is given in (13.19)-(13.20).

1: Note that we need to ensure subtours are not created between customer nodes only. Because of this, we initialize $u_{0k} = 1\ \forall k \in \mathcal{V}$ as every vehicle starts its tour from the origin depot and we do not define $u_{n+1,k}$. Because of (13.15), every vehicle will move from its last visited customer node to the destination depot anyway.

Note that, there could also be an explicit upper bound on the total number of vehicles for example as follows:

$$\sum_{k\in\mathcal{V}}\sum_{j\in\mathcal{N}} x_{0jk} \leq |\mathcal{V}| \quad (13.21)$$

It is also another convention to define a binary variable for the assignment of the vehicles to the nodes, e.g., z_{ik} which is 1 if node i is served by vehicle k and zero otherwise. **It is not compulsory to do so as we did not have it above. We can reconstruct which vehicle visits which customer by analyzing decision variables x_{ijk}.** Yet for some formulations, it may help to explicitly have this assignment for easier readability.

The formulation provided in (13.10)-(13.20) assumes that each vehicle has a maximum of one tour and that we do not have any time-related limitations. Capacity is already defined for each vehicle separately so in principle the fleet can be heterogeneous. Yet if this brings additional cost considerations, it needs to be incorporated as part of the objective function. For adding such costs on the fleet utilization, we can make use of the total number of used vehicles $\sum_{k\in\mathcal{V}}\sum_{j\in\mathcal{C}} x_{0jk}$. **Note that, here we do not consider those staying at the depot and rather count those leaving the depot towards a customer node (the second summation $\sum_{j\in\mathcal{C}}$ only spans over customer nodes and does not consider the zero-cost connection between the depot nodes 0 and $n+1$).** This can be adapted for the case at hand, for example as follows. Let us define C_k the fixed cost incurred if vehicle $k \in \mathcal{V}$ is used, and $y_k \in \{0,1\}$ a binary variable that is unitary if truck k performs a "real" tour (with the only "non-real" tour being a direct trip from node 0 to node $n+1$). In our current setting, we consider variable costs that are proportional to the traveled distance via parameters C_{ij}, but not such a fixed cost yet. We leverage the fixed charge constraints from Section 4.8.4 to model fixed costs. In particular, we replace (13.13) with the following constraints:

$$x_{0,n+1,k} + \sum_{i\in\mathcal{C}} x_{0ik} = 1 \qquad \forall k \in \mathcal{V} \qquad (13.22)$$

$$\sum_{i\in\mathcal{C}} x_{0ik} \leq y_k \qquad \forall k \in \mathcal{V} \qquad (13.23)$$

(13.22) states that each truck must move from the origin depot node 0 to the destination depot node $n+1$, either via the "non-real" tour mapped with $x_{0,n+1,k} = 1$ (representing a truck not used) or via any "real" tour where at least one customer is visited (mapped with $\sum_{i\in\mathcal{C}} x_{0ik} = 1$). (13.23) is the fixed charge constraint set stating that a vehicle can visit its first customer only if the associated y_k decision variable is equal to 1. To ensure that the model decides to activate trucks only if advantageous for the objective, the objective function (13.10) must also be modified as follows:

$$\min \quad \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{V}} C_{ij}x_{ijk} + \sum_{k\in\mathcal{V}} C_k y_k \qquad (13.24)$$

The initial term in (13.24) mirrors (13.10), representing variable costs linked to the trucks' routing. The subsequent term, however, is novel, capturing fixed costs attributed to the trucks utilized in the solution.

Further variants of the CVRP will be discussed later in Section 13.4. We consider an application of the CVRP in Example 13.2.

Example 13.2 *Consider the network provided in Example 13.1. Given that we have a load of 20 units to be delivered to each of the 4 cities in the network and that a vehicle can carry a maximum of 40 units, we want to assess the minimum distance covered to solve the problem.*

We recognize this is an example of CVRP and, using the formulation presented in (13.10)-(13.20), we realize we need at least two vehicles to perform these deliveries. Figure 13.3 provides the two tours corresponding to these two vehicles. The cost, in this case, is 150 which is higher than the cost in Example 13.1 where the capacity of the vehicles was not an issue. As we already explained a few times throughout the book, Example 13.2 is a more constrained version of Example 13.1, hence its optimal solution can only be the same or worse (as it is in this instance).

Coded example

The code used to model and solve Example 13.2 is available here.

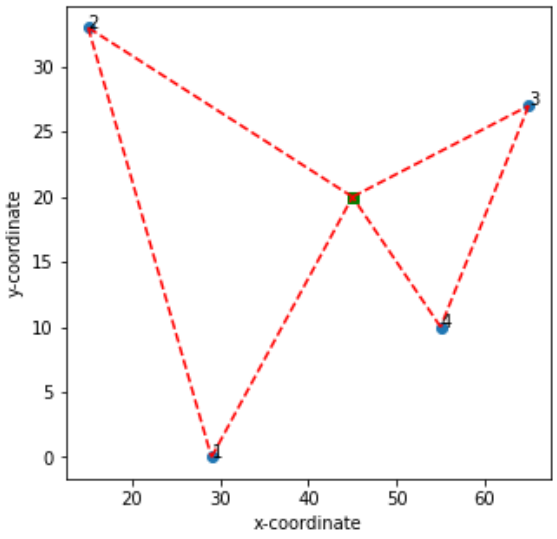


Figure 13.3: Solution to Example 13.2

13.4 Widely-used VRP variants

Up to now, we have only considered the fleet size and the capacity of the vehicles for VRPs. There are various interesting variants that enable representing real-life routing problems better. In this section, we will cover some of the most commonly used ones.

13.4.1 Vehicle Routing Problem with Time Windows (VRPTW)

Time windows are quite relevant in real-life cases as not all activities can be performed all around the clock. In the case of VRPs, this is typically considered as the time interval for each customer, i.e., node, to be served. Therefore, the earliest and latest start times of service at each node are introduced and together form the time window (A_i, B_i) for

node $i \in \mathcal{N}$. The additional notation for the Vehicle Routing Problem with Time Windows (VRPTW) is given in Table 13.4. We refer to Kallehauge et al. (2005) for an overview of the VRPTW together with its structure and solution methods developed.

Table 13.4: Additional notation for the VRPTW

Parameters	
A_i	earliest start time of service at node i
B_i	latest start time of service at node i
T_{ij}	travel time between nodes i and j
M_{ij}	large constant, i.e., big-M, for each node combination i and j
Variables	
s_{ik}	start time of service at node i by vehicle k

The set of constraints needs to be enhanced to respect the time windows, which means we need to now keep track of the service time at each node. Therefore, we need a new set of decision variables, s_{ik} to indicate the start time of service at node i by vehicle k. Constraints (13.25) ensure that a vehicle that visits two nodes consecutively starts the service considering the travel time in between, i.e., if node i is visited just before node j, the start time of service at node j should be at least T_{ij} time units later than the start time of service at node i. The time windows constraint is given in (13.26) such that the start time of service at each node respects the earliest and latest times indicated by the time windows.

$$x_{ijk}(s_{ik} + T_{ij} - s_{jk}) \leq 0 \qquad \forall i, j \in \mathcal{N}, k \in \mathcal{V} \tag{13.25}$$

$$A_i \leq s_{ik} \leq B_i \qquad \forall i \in \mathcal{N}, k \in \mathcal{V} \tag{13.26}$$

Note that constraints (13.25) are not linear as we have a product of x and s decision variables. Nevertheless, they can be linearized (as described in Section 5.4) as follows:

$$s_{ik} + T_{ij} - M_{ij}(1 - x_{ijk}) \leq s_{jk} \qquad \forall i, j \in \mathcal{N}, k \in \mathcal{V} \tag{13.27}$$

(13.27) ensures that if node i is visited just before node j, the start times at these nodes are related to each other based on the travel time between them. Otherwise, the constraint becomes non-binding thanks to the large constant M_{ij}. When we use such large constants, we need to have them large enough yet as small as possible for computational reasons. In this case, having the constant value specific to each node pair (i, j) enables us to tailor it to the time window of each node. We discussed in Section 5.4 how to tailor these constants.

The addition of the time tracking constraints (13.27) enables ordering the visit of the nodes, which is the idea behind the MTZ subtour elimination constraints introduced earlier. **If a node is visited after another node, the time is always later and this avoids going back to that earlier visited node again (as we cannot travel back in time, basically).** Therefore, when everything is put together, the MILP formulation for the VRPTW

can be provided as follows:

$$\min \quad \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{V}} C_{ij}x_{ijk} \tag{13.28}$$

s.t.:

$$\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{V}} x_{ijk} = 1 \qquad \forall i \in \mathcal{C} \tag{13.29}$$

$$\sum_{i\in\mathcal{C}} D_i \sum_{j\in\mathcal{N}} x_{ijk} \leq Q_k \qquad \forall k \in \mathcal{V} \tag{13.30}$$

$$\sum_{i\in\mathcal{N}} x_{0ik} = 1 \qquad \forall k \in \mathcal{V} \tag{13.31}$$

$$\sum_{i\in N} x_{ihk} - \sum_{j\in N} x_{hjk} = 0 \qquad \forall h \in \mathcal{C}, k \in \mathcal{V} \tag{13.32}$$

$$\sum_{i\in\mathcal{N}} x_{i,n+1,k} = 1 \qquad \forall k \in \mathcal{V} \tag{13.33}$$

$$s_{ik} + T_{ij} - M_{ij}(1 - x_{ijk}) \leq s_{jk} \qquad \forall i, j \in \mathcal{N}, k \in \mathcal{V} \tag{13.34}$$

$$A_i \leq s_{ik} \leq B_i \qquad \forall i \in \mathcal{N}, k \in \mathcal{V} \tag{13.35}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, j \in \mathcal{N}, k \in \mathcal{V} \tag{13.36}$$

We would like to mention that, in some problems, the time aspect also comes into play through time-dependent input parameters. For example, the travel times in the network change throughout the day which is the common feature of *time-dependent* VRPs. This in a sense is an attempt to handle the changing conditions in the network which is also the purpose of *dynamic* and/or *stochastic* VRPs which we will talk about in Section 13.5.

Coded example

The code used to model and solve a VRPTW here.

VRPTW as a serious game

A serious game based on the VRPTW can be found here. It entails three levels of increasing complexity.

13.4.2 Split Delivery Vehicle Routing Problem (SDVRP)

In real-life applications, it may not always be possible, and even not desirable sometimes, to serve a customer once in full. In other words, multiple vehicles may visit the same node each fulfilling part of the demand. This is also carried to VRP formulations under the name of Split Delivery Vehicle Routing Problem (SDVRP).

In order to facilitate the idea of split delivery, we need to make a few changes in the model. First of all, constraints (13.29) need to allow for multiple visits of the nodes as follows:

$$\sum_{j \in \mathcal{N}} \sum_{k \in \mathcal{V}} x_{ijk} \geq 1 \qquad \forall i \in \mathcal{C} \qquad (13.37)$$

Next, we need **a continuous variable to represent the quantity of demand served by each vehicle for each node**. Say y_{ik} is such a continuous variable that then needs to be constrained to ensure (i) that the total amount served by the set of vehicles indeed fulfills the entire demand of node i, (ii) that the capacity of each vehicle is respected, and (iii) that quantity served by vehicle k is positive only if x variables agree that there is an arc visiting node i with vehicle k. These constraints are provided as follows in the same order:

$$\sum_{k \in \mathcal{V}} y_{ik} = D_i \qquad \forall i \in \mathcal{C} \qquad (13.38)$$

$$\sum_{i \in \mathcal{N}} y_{ik} \leq Q_k \qquad \forall k \in \mathcal{V} \qquad (13.39)$$

$$y_{ik} \leq D_i \sum_{j \in \mathcal{N}} x_{ijk} \qquad \forall i \in \mathcal{C}, k \in \mathcal{V} \qquad (13.40)$$

In particular, (13.40) is similar to a fixed charge constraint (see Section 4.8.4) where y_{ik} (for a given $i \in \mathcal{C}$ and $k \in \mathcal{V}$) can only be greater than 0 if one x_{ijk} is unitary (i.e., if vehicle k reaches node i). The entire formulation for the SDVRP is as follows:

$$\min \quad \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} \sum_{k \in \mathcal{V}} C_{ij} x_{ijk} \qquad (13.41)$$

$$\sum_{j \in \mathcal{N}} \sum_{k \in \mathcal{V}} x_{ijk} \geq 1 \qquad \forall i \in \mathcal{C} \qquad (13.42)$$

$$\sum_{i \in \mathcal{N}} y_{ik} \leq Q_k \qquad \forall k \in \mathcal{V} \qquad (13.43)$$

$$\sum_{k \in \mathcal{V}} y_{ik} = D_i \qquad \forall i \in \mathcal{C} \qquad (13.44)$$

$$y_{ik} \leq D_i \sum_{j \in \mathcal{N}} x_{ijk} \qquad \forall i \in \mathcal{C}, k \in \mathcal{V} \qquad (13.45)$$

$$\sum_{i \in \mathcal{N}} x_{0ik} = 1 \qquad \forall k \in \mathcal{V} \qquad (13.46)$$

$$\sum_{i \in \mathcal{N}} x_{ihk} - \sum_{j \in \mathcal{N}} x_{hjk} = 0 \qquad \forall h \in \mathcal{C}, k \in \mathcal{V} \qquad (13.47)$$

$$\sum_{i \in \mathcal{N}} x_{i,n+1,k} = 1 \qquad \forall k \in \mathcal{V} \qquad (13.48)$$

$$s_{ik} + T_{ij} - M_{ij}(1 - x_{ijk}) \leq s_{jk} \qquad \forall i, j \in \mathcal{N}, k \in \mathcal{V}, \qquad (13.49)$$

$$A_i \leq s_{ik} \leq B_i \qquad \forall i \in \mathcal{N}, k \in \mathcal{V} \qquad (13.50)$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, j \in \mathcal{N}, k \in \mathcal{V} \qquad (13.51)$$

$$y_{ik} \in \mathbb{R}_0 \qquad \forall i \in \mathcal{N}, k \in \mathcal{V} \qquad (13.52)$$

where the full formulation (13.41)-(13.52) has already been described either in previous sections or above. One note we need to make is that, in this split delivery formulation, **we still have the assumption that each vehicle performs only one tour. The customer nodes can be visited by multiple vehicles**. There are also other relevant variants where the vehicles are allowed to perform multiple trips.

13.4.3 Multi-depot VRP

Considering distribution networks, a natural extension is that we have multiple depots in the network where the vehicles start and end their tours. This extension entails that we need to also have a set of depot nodes in our formulation. Namely, a depot set[2] $\mathcal{D}$ such that $\mathcal{D} \cup \mathcal{C} = \mathcal{N}$.

2: Note that in Section 13.3 we identify a single depot with two distinct nodes already. Notwithstanding, this was done just to facilitate some constraints, as the physical depot was only one. Here we extend the number of physical depots that can be used to perform delivery tours.

One decision that needs to be made is whether vehicles belong to a depot or not. In other words, in some applications, it might be possible for vehicles to end their tours at a different depot than the start depot. This version is related to the *open* VRP formulations where the vehicles are free to end at a customer node (Li et al., 2007). Nevertheless, in some cases, it is necessary to restrict the use of the depots such that a vehicle can use only one of the depots for its entire tour.

Let us now focus on a case where vehicles are assigned to a particular depot as part of the decision of the model, i.e., the vehicles are restricted to start and end at the same depot. The assignment of the vehicles to depots is given by z_{ik} such that it is 1 if vehicle k belongs to depot $i \in \mathcal{D}$ and 0 otherwise.

We need to make sure that the depot assignment is consistent with the routing decisions, i.e., z variables need to be linked to x variables:

$$\sum_{j \in \mathcal{N}} x_{ijk} = z_{ik} \qquad \forall i \in \mathcal{D}, k \in \mathcal{V} \tag{13.53}$$

(13.53) ensures that if vehicle k belongs to depot $i \in \mathcal{D}$, it starts its journey from that depot. In addition, here we decide

Therefore, the formulation for the multi-depot VRPTW can be provided as follows (without split deliveries, i.e., we only allow full deliveries):

$$\min \quad \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} \sum_{k \in \mathcal{V}} C_{ij} x_{ijk} \tag{13.54}$$

$$\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{V}} x_{ijk} = 1 \qquad \forall i \in \mathcal{C} \qquad (13.55)$$

$$\sum_{i\in\mathcal{C}} D_i \sum_{j\in\mathcal{N}} x_{ijk} \leq Q_k \qquad \forall k \in \mathcal{V} \qquad (13.56)$$

$$\sum_{j\in\mathcal{C}} x_{ijk} = z_{ik} \qquad \forall i \in \mathcal{D}, k \in \mathcal{V} \qquad (13.57)$$

$$\sum_{i\in\mathcal{N}} x_{ihk} - \sum_{j\in\mathcal{N}} x_{hjk} = 0 \qquad \forall h \in \mathcal{C}, k \in \mathcal{V} \qquad (13.58)$$

$$\sum_{j\in\mathcal{C}} x_{jik} = z_{ik} \qquad \forall i \in \mathcal{D}, k \in \mathcal{V} \qquad (13.59)$$

$$s_{ik} + T_{ij} - M_{ij}(1 - x_{ijk}) \leq s_{jk} \qquad \forall i, j \in \mathcal{N}, k \in \mathcal{V} \qquad (13.60)$$

$$A_i \leq s_{ik} \leq B_i \qquad \forall i \in \mathcal{N}, k \in \mathcal{V} \qquad (13.61)$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, j \in \mathcal{N}, k \in \mathcal{V} \qquad (13.62)$$

$$z_{ik} \in \{0, 1\} \qquad \forall i \in \mathcal{D}, k \in \mathcal{V} \qquad (13.63)$$

Constraints (13.55)-(13.56) and (13.60)-(13.62) are the same as in the case of the VRPTW. The core flow conservation constraints at the customer nodes (13.58) are also the same. We need to be careful about leaving the depot nodes and returning back to them. As mentioned earlier, this needs to be done by relating the x and z variables. Constraints (13.57) ensure that if a vehicle k uses depot i then this vehicle will start from that depot. Similarly, if vehicle k is assigned to depot i, it needs to return to that depot (13.59). Constraint set (13.63) defines the new set of decision variables. In the current setting, if vehicle k is assigned to a depot i ($z_{ik} = 1$), we force the vehicle to leave the depot $\rightarrow \sum_{j\in\mathcal{C}} x_{ijk} = 1$ and to go back to the depot $\rightarrow \sum_{j\in\mathcal{C}} x_{jik}$. If a truck k is not needed in the solution, then the model will set $z_{ik} = 0\ \forall i \in \mathcal{D}$. Hence, there is no strict need to duplicate each depot into an origin and destination node as seen previously in Section 13.3. Notwithstanding, we would bump into issues if we had to ensure that each vehicle does not exceed a pre-defined maximum time range. We explain this in detail in the **How to handle time range restrictions in model (13.54)-(13.63)** box.

Finally, sometimes each depot has already a certain number of trucks pre-assigned to it, similar to what showed in Figure 4.1. In such a case, assignment variable z_{ik} is not needed any longer, but we can define $\mathcal{V}_i\ \forall i \in \mathcal{D}$ the subset of trucks starting/ending their trip in depot i. We leave the modifications to model (13.54)-(13.63) that reflect such a change to interested readers.

13.4.4 Pickup and Delivery Problem (PDP)s

Another commonly used variant is the Pickup and Delivery Problem (PDP), which means that **we are not only delivering or picking up goods but rather doing both. For example, if we think of a ridesharing vehicle, a vehicle needs to pick up and drop off travelers and we need to ensure that the same vehicle performs both of the operations**. Therefore, typically a set of pickup and delivery nodes are defined and they are associated with each other to indicate the pairs that correspond to the

How to handle time range restrictions in model (13.54)-(13.63)

In model (13.54)-(13.63) $s_{ik}\ \forall i \in \mathcal{N}, k \in \mathcal{V}$ is defined for the origin depot and all customer nodes. With such a setting, time from the start of a vehicle's tour can be modeled via (13.60) and (13.61) to, respectively, apply time precedence constraints from the depot to every visited customer and between customers and to ensure customers are visited between the specified time window. If the tour of each vehicle $k \in \mathcal{V}$ is bounded by a maximum time range T_k, we need to duplicate each depot into an origin an destination depot so an s_{ik} decision variable for each destination depot can be defined. This can be done by extending the concept seen in Section 13.3 with a set of origin depots $\mathcal{D}_o$ and a set of destination depots $\mathcal{D}_d$ so that $\mathcal{N} = \mathcal{D}_o \cup \mathcal{C} \cup \mathcal{D}_d$ and origin depot i is associated with destination depot indexed by $j = i + |\mathcal{C}| + |\mathcal{D}_o|$. With such a change, we can define a maximum time range constraint per truck as:

$$s_{i+|\mathcal{C}|+|\mathcal{D}_o|,k} - s_{ik} \leq T_k \qquad \forall i \in \mathcal{D}_o, k \in \mathcal{V} \tag{13.64}$$

Accordingly, we will have to modify (13.57) to only focus on origin depots and (13.59) to only focus on destination depots as follows:

$$\sum_{j \in \mathcal{C}} x_{ijk} = z_{ik} \qquad \forall i \in \mathcal{D}_o, k \in \mathcal{V} \tag{13.65}$$

$$\sum_{j \in \mathcal{C}} x_{jik} = z_{ik} \qquad \forall i \in \mathcal{D}_d, k \in \mathcal{V} \tag{13.66}$$

same job, e.g., traveler or a package. We refer to Dumas et al. (1991) for an overview of PDPs with time windows.

The main differences in the formulation lie in tracking the entities that are being transported to make sure that the pickup and delivery of the same entity is performed by the same vehicle. Moreover, the capacity tracking needs to be adapted accordingly. until now, it was sufficient to consider the total demand of the served customers in order to comply with the capacity of the vehicles, yet in this case, we need to take into account the increased or decreased load at each node visit.

PDPs are also widely studied under the name of Dial-A-Ride Problem (DARP)s, which focus on the case of travelers who are picked up and dropped off for different activities. We refer to Cordeau and Laporte (2007) for an overview of DARPs with different mathematical models and solution algorithms.

13.5 Further VRP variants

We have covered important variants developed for VRPs and there are further variants that may be relevant for different settings. We refer to a recent review paper, Elshaer and Awad (2020), for an overview of different variants and solution methods provided for them. In this section, we will list some important variants with references and brief explanations:

- **Multi-trip VRP**. In the problems covered so far, we assumed that each vehicle performs a single tour. In real-life settings, the vehicles may perform multiple tours given that operational hours permit that. There are different conventions adopted to represent these multiple trips, e.g., a set of trips defined for each vehicle or handling the assignment of trips to vehicles. We refer to Brandão and Mercer (1998) for an early introduction of multi-trip VRPs and to Cattaruzza et al. (2016) for a variant including time windows and release times (e.g., when each truck is available to be operated);
- **Dynamic VRP**. The transportation systems have been shaped to have a more dynamic nature in the last few decades in order to keep up with the changing behavior in demand. For example, in delivery systems, the customers expect to receive their orders in much shorter times. This means that the delivery system needs to be prepared to accommodate newly arriving requests in a timely manner. In this regard, routing problems are also shaped to have a dynamic nature accordingly. We refer to a review by Pillac et al. (2013) for an overview of this class of problems. Typically, in these problems, the demand arrives dynamically and the routing problem needs to be re-optimized in different ways. In some other problems, the travel and/or service times are dynamic in order to accommodate changing conditions on the network. Note that, dynamic problems in a sense handle uncertainties in the system through their capability of adapting the decisions and are often considered together with stochastic approaches which we mention in the next class of problems;
- **Stochastic VRP**. Stochastic variants of VRPs are very relevant considering the various uncertainties embedded in transportation systems. Some of those uncertainties are related to the demand, e.g., the amount of demand that will be observed or whether the customer will show up at a given node or not. There are also various uncertainties related to the transportation network, e.g., travel and/or service times on the network. We refer to Gendreau et al. (1996) for a relatively early review paper on stochastic TSPs and VRPs. These problems are formulated in different ways in order to handle the embedded uncertainty such as *Chance Constrained Programming* or *Stochastic Programming*. We touch upon some stochastic optimization techniques in Chapter 14;
- **Rich VRP**. This terminology is used for VRPs where various constraints and specifications are combined to better represent the complexities of real-life routing problems. We refer to Lahyani et al. (2015) for the taxonomy and an overview of such problems. It is important to note that rich VRPs do not refer to a particular type of problem but rather highlight the real-life challenges and how to model and solve them.

Part VI

Stochasticity

Two-stage stochastic programming 14

14.1 Stochastic programming 269
14.2 Two-stage recourse problem 270
14.3 Final words and recommended literature 281

> Life is like a box of chocolates. You never know what you're gonna get.
>
> Forrest Gump

Uncertainty plays a significant role in several industries and applications. Investments in accurate forecasting tools are rising to predict the future better and make more sound decisions. When parameters of a certain problem are affected by uncertainty, the proposed mathematical model should take that into account. What a waste to buy some vehicles with high capacity when it turns out later that the demand volumes are lower than expected. How unfortunate to be late to our customers due to traffic. **These examples highlight that how we described parameters in Section 4.2, i.e., as deterministic (certain) values, might be an overly simplistic approach sometimes**.

On the other hand, models considering deterministic parameters are somewhat acceptable in several cases. When the total demand can be easily determined, for example, by pre-orders, or when the traffic can cause, on average, delays of a few minutes, exact quantities for our parameters may not severely affect the goodness of the model's solution. In such cases, avoiding the computational burden of more complex **stochastic** (i.e., uncertain) models should be avoided. It is the task of the modeler to opt for a deterministic or stochastic model based on the application and the pre-conditions.

In this chapter, we study how to turn a deterministic model into a stochastic one, and we show a set of indicators to appreciate the benefit of using a stochastic approach. The reader must be aware of two main issues. **First, a stochastic model will increase the number of variables and the complexity of solving the model. Second, a good definition of the parameters and the related probabilities must be done carefully**.

14.1 Stochastic programming

In a stochastic setting, information on parameters is uncertain. However, the value of these parameters will become known at some point. An important aspect is whether or not the decision can be adapted after knowing the information. **The possibility of adapting the information is called recourse**. We now proceed to elaborate on both cases.

The first example concerns the possibility of recourse. When driving from home to a destination in an urban area, our Global Positioning System (GPS) may provide two similar alternatives with a given Estimated Time of Arrival (ETA) based on real-time traffic or projections. During travel, more information is disclosed, and the GPS could suggest alternative routes because traffic has increased on the original path or decreased on connecting roads. The key point here is that while we can adjust our plans

to some extent, our initial decisions have already influenced the initial portion of the route. However, we can mitigate the impact by adapting our route as we progress. **If the GPS had foreseen the traffic conditions from the start, it could have favored a different route, minimizing the time required to reach our destination**.

A no-recourse situation arises when the initial decision cannot be adjusted. For instance, consider a scenario in the transportation domain where we must decide whether to travel by boat or by flight, balancing cost and time considerations, amidst the possibility of unexpected delays caused by flight cancellations or rough seas. Once the journey begins, we are unable to alter our initial decision or make any minor adjustments. We must accept the consequences of our initial choice.

In this chapter, our focus will be on recourse problems. We will leverage scenarios and probabilities to devise robust solutions against future uncertainties. The model can be deployed at the outset of the decision-making period, offering both the initial decision and its adaptation based on realized scenarios. On the other hand, no-recourse problems could be addressed using simulation tools.

14.2 Two-stage recourse problem

Although the GPS example can somehow easily show the dynamics of a recourse problem, its actual implementation is quite challenging. The main reason is that we can have multiple moments during the route where we can make adjustments to our decision. This can generate a huge quantity of possibilities to consider. Because of this, a more suitable approach could be to solve a SP multiple times while traveling. We will use a simpler but more common example in the literature to explain stochastic programming, namely manufacturing. But first, let us formalize a stochastic programming model.

We consider a problem where some parameters ξ are initially uncertain. Our problem requires us to make some decisions now, given what we know about the present situation while trying to acknowledge and capture effectively the uncertainty of those ξ parameters. This first set of decisions is generally defined **x, i.e., first stage decisions**. At some point in the future, the unknown parameters ξ will become known and we might take different actions according to the revealed outcome. This point in future is called **second stage** and is where **some recourse action can be taken**. Let us assume our problem is defined by an objective function to maximize. We can define such a problem in mathematical form as:

$$\max \underbrace{c^T x}_{Z_1} + \underbrace{\mathbb{E}\left[Q(x,\xi)\right]}_{Z_2} \tag{14.1}$$

s.t.:

$$Ax \leq b \tag{14.2}$$
$$x \geq 0 \tag{14.3}$$

where the objective function (14.1) is divided into two parts as follows:

- Z_1: contribution to the objective from the first stage. c is the known vector of coefficients of the first stage decision variables x;
- Z_2: expected contribution to the objective from the second stage. **Because we are evaluating this problem "now" and not in the future, this contribution is an expected value $\mathbb{E}$ that depends on the decision we take now (x) and on the unknown set of parameters ξ**. Referring back to our GPS example, our initial decision on which route to take (x) will take us to a different location in the future. This decision, coupled with the current congestion level (ξ) which was unknown when the initial decision was made, will shape and influence the available recourse actions.

Constraint set (14.2) encapsulates all the constraints that are known and we need to comply with from the first stage only while constraint set (14.3) defines the general non-negative nature of the first stage decision variables (while still allowing them to appear in a mix of continuous, integer, and binary).

With $[Q(x, \xi)]$ we mean a realization of the second stage part of our problem, which can be expanded as:

$$\max \; q_\xi^T y \tag{14.4}$$

s.t.:

$$G_\xi x + U_\xi y \leq h_\xi \tag{14.5}$$
$$y \geq 0 \tag{14.6}$$

where (14.4) is the counterpart of (14.1) with q_ξ being the coefficients of the second stage decision variables that depend on a specific realization of ξ. Similarly, G_ξ and U_ξ in constraint set (14.5) are the coefficient matrices for the first and second stage decision variables. They also depend on the actual realization of the unknown parameters ξ[1]. Additionally, constraint set (14.5) emphasizes how first stage decisions do affect the second stage. If we consider a specific h_ξ as some capacity we can use without exceeding, the more capacity we have used in the first stage, the less we have still available in the second stage. Finally, (14.6) defines the non-negativity of second stage decisions.

The final stage of this process is to embed (14.4)-(14.6) into the initial formulation **by finding a way to mathematically approximate the $\mathbb{E}_\xi[\cdot]$ (expected value) operator with something that a linear solver can handle**.

1: In general, U does not depend on ξ. This case is called *fixed recourse* and features mathematical properties that make it easier to solve. We refer readers to Birge and Louveaux, 2011 for more details.

We achieve such a goal with a technique called Sample Average Approximation (SAA). This technique resembles a Monte Carlo simulation and approximates the uncertainty regarding the future and the unknown parameters ξ with a set of scenarios $\mathcal{S}$ indexed by s. In practice, each scenario is a possible realization of the future and its unknown parameters ξ with a certain probability P_s. We assume that our future can only be represented by our set of scenarios $\mathcal{S}$, hence $\sum_{s\in\mathcal{S}} P_s = 1$. A crucial component of such an approach is then to define $\mathcal{S}$ in a way that can well capture at least the most likely realizations of the future: tools such as simulation can help in this regard. For each scenario $s \in \mathcal{S}$, the specifications are known. Hence, we can define ξ_s and y_s as the parameters and second stage decision variables specific to scenario s. **To summarize, we translate uncertainty into a set of "certain" (i.e., deterministic) scenarios, each characterized by a certain probability P_s of occurring. For example, if a weather forecast states that, in one week from now, there is an equal probability that it will be sunny, cloudy, or rainy with no other alternative, we could consider the three options as the three scenarios $\mathcal{S} = \{1, 2, 3\}$ each characterized by $P_s = \frac{1}{3}$.** As it concerns y_s, these variables represent the decision we would take when we would find ourselves in the realized scenario s.

Considering SAA, our final formulation is:

$$\max \; c^T x + \sum_{s\in\mathcal{S}} P_s q_s^T y_s \tag{14.7}$$

s.t.:

$$Ax \leq b \tag{14.8}$$
$$G_s x + U y \leq h_s \qquad \forall s \in \mathcal{S} \tag{14.9}$$
$$x \geq 0 \tag{14.10}$$
$$y_s \geq 0 \qquad \forall s \in \mathcal{S} \tag{14.11}$$

(14.7) and constraints (14.8)-(14.11) define the two-stage recourse problem we aim at solving. In (14.7), if we assume that all scenarios are equally likely we can replace the second term with $\frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}} q_s^T y_s$ as $P_1 = P_2 = \cdots = P_{|\mathcal{S}|} = \frac{1}{|\mathcal{S}|}$. Constraint set (14.8) defines the constraints stemming from the first stage only, while constraint set (14.9) defines, for every scenario, the constraints linking the first stage decisions and the recourse. Note that we assume a fixed recourse matrix U. Finally, constraint sets (14.10)-(14.11) define the non-negativity of the decision variables.

Before applying the model (14.7)-(14.11) to an example, we briefly elaborate on the key role of the set of scenarios $\mathcal{S}$. We introduced that $\mathcal{S}$ is key in approximating the uncertainty of the second stage. **Therefore, akin to a Monte Carlo simulation, one might assume that having a vast array of scenarios could lead to a superior final solution. This notion holds particularly true for large-scale problems, where the potential realizations of the future are marked by such significant uncertainty that the possible scenarios essentially become infinite**. Considering model (14.7)-(14.11), both second stage decision variables and constraints

grow with $|\mathcal{S}|$. Hence, providing our model with a large set of scenarios might make it more realistic, but also computationally way harder to solve. Finally, it is also worth noting that more scenarios are generally better pending that those scenarios are "good" ones. **In essence, a good two-stage stochastic problem is the one where the minimum number of good-quality scenarios is added**.

One of the most prominent examples in the literature for two-stage stochastic programming is the procurement of resources for producing a set of goods. Each unit of a good requires a defined amount of resources. The demand for the goods is uncertain, but we know the profit for each unit and the cost to buy a unit of each resource. The problem is about deciding the quantities of resources to buy in the first stage based on projected demand, with the aim of maximizing the profit. The timing of the problem is as follows. During the first stage (at time t_{FS}), we decide the quantity of each resource to buy. We assume that this decision cannot be adapted later because, for example, it takes a long time for the supply to arrive and, if ordered later, the resources would arrive too late for production. Once the resources are ordered, we can wait to start production until the demand is known at a later time (the second stage, at time t_{SS}). Here, we can decide the production quantities based on the realized scenario. Due to the assumption that only the available scenarios can occur in the future, we can define a two-stage stochastic model at time t_{FS} and decide both the first stage variables x and the second stage ones y_s which are scenario-dependent, for every scenario $s \in \mathcal{S}$ we have predicted. **The model will find a good compromise that considers the probability of each scenario and the trade-off between cost and profit, so that at time t_{SS} our second stage decisions should be applicable in a robust manner regardless of the realized scenario**. We showcase and critically assess such a framework in Example 14.1

Example 14.1 *A company produces three products, A, B, and C, from three resources (e.g., raw materials), R1, R2, and R3. To produce A, they need 3 units of R1, 2 of R2, and 4 of R3. To produce B, they need 1 unit of R1, 2 units of R2, and 2 of R3. Finally, to produce C, they need 4 units of R1, 2 units of R2, and no units of R3. With regard to profit, it is obtained 25€ per unit for product A, 20€ for product B, and 10€ for product C. Costs for the resources per unit are: 3€ for R1, 2€ for R2, and 1€ for R3. The company has forecasted 3 possible scenarios to occur. With probability 50%, the expected demand for the three products is, respectively, 40, 30, and 10. With a 40% probability, it is 30, 20, 0. Finally, with a 10% probability, the expected demand is 10, 30, and 50. All the input data is summarized in Table 14.1. The company would like to devise a strategy in terms of resource acquisition and production so that the expected profit is maximized.*

We recognize that this is a problem where we can apply the two-stage stochastic programming methodology that was described above. Before even diving into the model, it is good practice to analyze the input data. For example, it appears that C is as costly as A, roughly, production-wise but generates less than half the profit. It is also very likely that the demand for this product will not be high, except for a 10% chance. On average, we expect a demand from C of $\frac{1}{2} \times 10 + \frac{2}{5} \times 0 + \frac{1}{10} \times 50 = 10$.

Let us define the set of products $\mathcal{P} = \{1, 2, 3\}$, indexed by p (where $A = 1$, $B = 2$, and $C = 3$), and the set of resources $\mathcal{R} = \{1, 2, 3\}$, indexed by r (where $R1 = 1$, $R2 = 2$, and $R3 = 3$). In addition, we define the set

Table 14.1: Input data for Example 14.1.

	Cost (unit)	A	B	C
R1	3	3	1	4
R2	2	2	2	2
R3	1	4	2	0
Revenue (unit)		25	20	10
Demand	Probability			
	50%	40	30	10
	40%	30	20	0
	10%	10	30	50

of scenarios $\mathcal{S} = \{1, 2, 3\}$, where $P_1 = \frac{1}{2}$, $P_2 = \frac{2}{5}$, and $P_3 = \frac{1}{10}$. The only parameter that is scenario-dependent is the demand per product D_{ps}, i.e., the demand of product p in scenario s. We define C_r as the cost of one unit of resource r and Q_{rp} as the units of resource r needed to produce one unit of product p. Finally, R_p is the revenue per unit of product p.

The decision variables relate to the two-stage stochastic programming framework. In the first stage, we purchase the resources needed for the production. Hence, we define $x_r \in \mathbb{R}_0$ as the purchased units of resource $r \in \mathcal{R}$. Then, we define $y_{ps} \in \mathbb{R}_0$ as the units of product p produced and sold in scenario s.

We define the two-stage recourse model as:

$$\max - \sum_{r \in \mathcal{R}} C_r x_r + \mathbb{E}\left[Q(x, D)\right] \tag{14.12}$$

s.t.:

$$x_r \in \mathbb{R}_0 \qquad \forall r \in \mathcal{R} \tag{14.13}$$

where in (14.12) the contribution of the first stage is non-positive. **This is correct as, in the first stage, we are only buying resources, hence contributing negatively to the profit**. Additionally, with $\mathbb{E}\left[Q(x, D)\right]$ we imply that the uncertainty of the second state resides in the demand D. Note that, apart from the usual non-negativity of the decision variables (we also assume decision variables can be continuous, and hence we can purchase partial units), there are no specific constraints in the first stage decision. This is generally not the case, as in some other problems, for example, some budget-related constraints could be necessary.

For a specific scenario $s \in \mathcal{S}$ we can express $Q(x, D)$ as:

$$\max \sum_{p \in \mathcal{P}} R_p y_{ps} \tag{14.14}$$

s.t.:

$$\sum_{p \in \mathcal{P}} Q_{rp} y_{ps} \leq x_r \qquad \forall r \in \mathcal{R} \qquad (14.15)$$

$$y_{ps} \leq D_{ps} \qquad \forall p \in \mathcal{P} \qquad (14.16)$$

$$y_{ps} \in \mathbb{R}_0 \qquad \forall p \in \mathcal{P} \qquad (14.17)$$

where (14.14) aims at maximizing the revenue generated from the sales of the three products. The production is capped by the first stage decisions in constraint set (14.15) and by the demand occurring in the current scenario s in constraint (14.16). Finally, (14.17) defines decision variables y_{ps} as continuous and non-negative (some readers might argue it could have been omitted because of (14.16). We left it to make it explicit that the nature of the decision variables is continuous and not necessarily integer).

We can combine models (14.12)-(14.13) and (14.14)-(14.17) and leverage the SAA paradigm to obtain the full two-stage recourse problem as follows:

$$\max - \sum_{r \in \mathcal{R}} C_r x_r + \sum_{s \in \mathcal{S}} P_s \sum_{p \in \mathcal{P}} R_p y_{ps} \qquad (14.18)$$

s.t.:

$$\sum_{p \in \mathcal{P}} Q_{rp} y_{ps} \leq x_r \qquad \forall r \in \mathcal{R}, s \in \mathcal{S} \qquad (14.19)$$

$$y_{ps} \leq D_{ps} \qquad \forall p \in \mathcal{P}, s \in \mathcal{S} \qquad (14.20)$$

$$x_r \in \mathbb{R}_0 \qquad \forall r \in \mathcal{R} \qquad (14.21)$$

$$y_{ps} \in \mathbb{R}_0 \qquad \forall p \in \mathcal{P}, s \in \mathcal{S} \qquad (14.22)$$

where (14.18) defines the objective function comprising the first stage costs and the expected revenue from the second stage, (14.19) and (14.20) ensure that production is carried out considering the available resources acquired and the maximum demand, respectively, and (14.21)-(14.22) define the continuous and non-negative nature of decision variables x_r and y_{ps}. **In particular, constraint (14.19) is the explicit form of (14.9) and depicts how first stage decisions affect the recourse actions of the second stage**.

Given the relatively small size of the model, we write it in explicit form as:

$$\begin{aligned} \max \quad & -3x_1 - 2x_2 - x_3 + \frac{1}{2}(25y_{11} + 20y_{21} + 10y_{31}) \\ & + \frac{2}{5}(25y_{12} + 20y_{22} + 10y_{32}) + \frac{1}{10}(25y_{13} + 20y_{23} + 10y_{33}) \end{aligned} \qquad (14.23)$$

s.t.:

$$3y_{11} + y_{21} + 4y_{31} \leq x_1 \tag{14.24}$$
$$3y_{12} + y_{22} + 4y_{32} \leq x_1 \tag{14.25}$$
$$3y_{13} + y_{23} + 4y_{33} \leq x_1 \tag{14.26}$$
$$2y_{11} + 2y_{21} + 2y_{31} \leq x_2 \tag{14.27}$$
$$2y_{12} + 2y_{22} + 2y_{32} \leq x_2 \tag{14.28}$$
$$2y_{13} + 2y_{23} + 2y_{33} \leq x_2 \tag{14.29}$$
$$4y_{11} + 2y_{21} \leq x_3 \tag{14.30}$$
$$2y_{12} + 2y_{22} \leq x_3 \tag{14.31}$$
$$2y_{13} + 2y_{23} \leq x_3 \tag{14.32}$$
$$y_{11} \leq 40 \tag{14.33}$$
$$y_{21} \leq 30 \tag{14.34}$$
$$y_{31} \leq 10 \tag{14.35}$$
$$y_{12} \leq 30 \tag{14.36}$$
$$y_{22} \leq 20 \tag{14.37}$$
$$y_{32} \leq 0 \tag{14.38}$$
$$y_{13} \leq 10 \tag{14.39}$$
$$y_{23} \leq 30 \tag{14.40}$$
$$y_{33} \leq 50 \tag{14.41}$$
$$x_1, x_2, x_3 \geq 0 \tag{14.42}$$
$$y_{11}, y_{12}, y_{13}, y_{21}, y_{22}, y_{23}, y_{31}, y_{32}, y_{33} \geq 0 \tag{14.43}$$

The model depicted by (14.23)-(14.43) is an LP. If we had been presented with this model initially, we might not have recognized that it incorporates uncertainty through SAA. The level of complexity of the suggested example is highly manageable. In these types of efforts, the numerical analysis before and after the model is usually quite interesting and impactful for the final application.

Post-analysis typically involves evaluating the advantages of employing a stochastic approach over a deterministic one. Here, we will present an analysis along with some KPIs to facilitate this evaluation. In Table 14.2, we present the results of the stochastic model alongside its deterministic counterpart. **For the latter, we adopt a straightforward approach in which the weighted average demand is considered**. This yields a demand of 33 units for product A, 26 units for product B, and 10 units for product C.

Table 14.2: Different solutions for Example 14.1. *Stoc. Sol.* stands for the solution from the two-stage stochastic model. *Det. Sol.* stands for the solution from the deterministic counterpart using the weighted average demand. *Det. Sol.* $s = *$ stands for the outcome of the deterministic solution in case of realized demand from scenario $*$.

	x_1	x_2	x_3	y_{p1}	y_{p2}	y_{p3}	y_{det}	Profit
Stoc. sol.	110	113.33	166.7	26.67,30,0	30,20,0	10,30,12.5		467.5
Det. sol.	125	118	184	-	-	-	33,26,0	550
Det. sol. $s = 1$	125	118	184	-	-	-	33,26,0	550
Det. sol. $s = 2$	125	118	184	-	-	-	30,20,0	355
Det. sol. $s = 3$	125	118	184	-	-	-	10,30,16.25	217.5
Det. sol. AVG								438.75

Table 14.2 conveys some interesting insights. The expected profit of the stochastic solution is 467.5€, whereas the deterministic solution is 550€. **Although the deterministic solution initially appears more**

favorable, it is contingent upon the "expected" weighted average demand materializing. In essence, we achieve a profit of 550€ only if we are fortunate enough to experience the expected demand value. We should also check what happens when the three scenarios occur. Based on our initial assumption, these are the only possibilities in the future that we considered in the stochastic model. This is displayed in the second half of the table. If we opted for producing the quantities suggested by the deterministic model, we would still get a profit of 550€ if the first scenario occurs. In this case, the model is giving priority to the more profitable products A and B. The same occurs for scenario 2, but in this case, the production for A and B is lower, and C is not produced because its demand is 0. Resources are now wasted and the profit is lower. In scenario 3, C is now produced with the remaining resources from the production of A and B. If this scenario occurs, the company will suffer a somewhat hard blow with a profit of 217.5€ only. The average profit of the deterministic approach is 438.75€, providing evidence that, in the long run, the stochastic approach is safer and more profitable. **Note that product C is not profitable to produce, however the stochastic model will produce it to reduce damage in case the third scenario occurs**.

From the numerical analysis, we can also evaluate other KPIs to assess the benefit of using a stochastic approach. The first performance indicator is named Value of Stochastic Solution (VSS), which is given by the difference between the solution of the two-stage recourse problem and the weighted average of the deterministic solutions. In the example, VSS= 467.5 − 438.75 = 28.75€. **The positive value further corroborates the added value of the stochastic approach**.

Another important KPI is the Expected Value of Perfect Information (EVPI). EVPI expresses the amount we should be willing to pay to get perfect information about the future. It is equal to the difference between stochastic solution and the so-called Wait-and-See (WS) approach. WS means deciding everything only when the future is disclosed. In our example, it basically means deciding in the first stage while knowing what will happen in the future. Hence, in the context of Example 14.1, this is equivalent to solving three deterministic problems, one for each scenario: there is no distinction any longer between the first and second stage. In each problem, we assume the demand for the products is equivalent to the one of the scenario considered. The results are shown in Table 14.3.

Table 14.3: WS solution to Example 14.1.

	x_1	x_2	x_3	y	Profit
$s = 1$	150	140	220	40,30,0	650
$s = 2$	110	100	160	30,20,0	460
$s = 3$	60	80	100	10,30,0	410
Det. sol. AVG					550

Because product C is not profitable, it will never be produced in the WS solution. The solution is equal to the one solving the deterministic model with weighted averages for the demand. The EVPI is equal to 550 − 467.5 = 82.5€.

We now showcase a second example, i.e., Example 14.2, very different in nature. This example strongly correlates with Example 10.1 as it features the same setting but with a revamped twist.

Coded example

The code used to model and solve Example 14.1 is available here.

Example 14.2 *The same bounty hunter from Example 10.1 (hence, with the same $L = 16$ life points) has been told that the outlaws they captured have escaped and are hiding again in the forest. This time, armed with valuable intel from their informant, they confront a new dilemma. Initially encountering four outlaws - Zorgoiln the Zombie, Henry the Hermit Crab, Ghost of your past, and Marion of the Haron - the bounty hunter must decide their fate wisely. However, the twist lies in the subsequent encounter, where only two out of the remaining four outlaws - Gerald the Gunk, The Big Brown Bear, the Frog Prince, and the Mummy - will appear, each pair having equal probability. The bounty hunter's objective is clear: to strategically determine which outlaws to defeat and which to spare among the initial four in light of the future encounter with two unknown additional outlaws, ensuring the maximum expected return in terms of gold coins.*

We recognize this as another example of a two-stage recourse program, where the first stage decision variables define whether we decide to defeat or not the four opponents we know with certainty, and the second stage decision variables define, for each scenario, our recourse actions. Because in the second stage two outlaws out of four will appear, we have a fairly limited set of scenarios: $|\mathcal{S}| = \frac{4!}{2!2!} = 6$. They are (Gerald the Gunk, The Big Brown Bear), (Gerald the Gunk, The Frog Prince), (Gerald the Gunk, The Mummy), (The Big Brown Bear, The Frog Prince), (The Big Brown Bear, The Mummy), and (The Frog Prince, The Mummy). We report the outlaws from the first stage in Table 14.4, and the outlaws associated to each $s \in \mathcal{S}$ in Table 14.5.

Our first set of decision variables entails binary variables $x_1, \cdots, x_4$ which are unitary if the associated outlaw is defeated in the first stage. Then, in the second stage we define binary decision variables y_{s,o_s} that are unitary if in scenario s we decide to defeat outlaw o_s. Because we have 6 scenarios and 2 outlaws per scenario, there will be 12 y_{s,o_s} variables. For example, when $s = 1$ we have $y_{1,5}$ and $y_{1,6}$, related, respectively, to the possibility of defeating Gerald the Gunk or The Big Brown bear. In this regard, we define $\mathcal{O}_1$ as the subset of outlaws that the bounty hunter faces in the first stage. On a similar note, $\mathcal{O}_{2,s}$ is the subset of outlaws present in the second stage of scenario s, so that (using indices for compactness) $\mathcal{O}_{2,1} = \{5, 6\}$, $\mathcal{O}_{2,2} = \{5, 7\}, \cdots, \mathcal{O}_{2,6} = \{7, 8\}$.

In this specific case, given the small-scale of the problem, $\mathcal{S}$ covers the full set of realizations of the second stage. The bounty hunter does not know which of the six scenarios will they face, but they know it will be one of those six. This is generally not the case for larger-scale problems, where infinite realizations of the future are possible. In addition, we explicitly stated that each scenario is equally likely to happen, hence $P_1 = P_2 = \cdots = P_{||\mathcal{S}|} = \frac{1}{||\mathcal{S}|}$.

The objective of the bounty hunter is to maximize the expected gold coins obtained in the first and second stage. The only set of constraints pertains to the necessity to stay alive.

Outlaw	o	C_o	L_o
Zorgoiln the Zombie	1	5	2
Henry the Hermit Crab	2	17	5
Ghost of your past	3	15	4
Marion of the Haron	4	19	5

Table 14.4: Data pertaining to the four outlaws in the first stage of Example 14.2.

Outlaw	o	C_o	L_o
$s = 1$			
Gerald the Gunk	5	55	14
The Big Brown Bear	6	8	2
$s = 2$			
Gerald the Gunk	5	55	14
The Frog Prince	7	8	2
$s = 3$			
Gerald the Gunk	5	55	14
The Mummy	8	32	7
$s = 4$			
The Big Brown Bear	6	8	2
The Frog Prince	7	8	2
$s = 5$			
The Big Brown Bear	6	8	2
The Mummy	8	32	7
$s = 6$			
The Frog Prince	7	8	2
The Mummy	8	32	7

Table 14.5: Data pertaining to the two outlaws for each scenario $s \in \mathcal{S}$ in the second stage of Example 14.2.

We can write the two-stage recourse mathematical formulation of this problem as:

$$\max \sum_{o \in \mathcal{O}_1} C_o x_o + \sum_{s \in \mathcal{S}} P_s \sum_{o \in \mathcal{O}_{2,s}} C_o y_{s,o} \tag{14.44}$$

s.t.:

$$\sum_{o \in \mathcal{O}_1} L_o x_o + \sum_{o \in \mathcal{O}_{2,s}} L_o y_{s,o} \leq L - 1 \qquad \forall s \in \mathcal{S} \tag{14.45}$$

$$x_o \in \{0, 1\} \qquad \forall o \in \mathcal{O}_1 \tag{14.46}$$

$$y_{s,o} \in \{0, 1\} \qquad \forall s \in \mathcal{S}, o \in \mathcal{O}_s \tag{14.47}$$

(14.44) maximizes the gold coins from the first stage and the expected gold money return from the second stage. Constraint set (14.45) ensures that the bounty hunter stays alive in every scenario considering both the known outlaws from the first stage and the scenario-specific outlaws of

the second stage, while (14.46)-(14.47) define that nature of the decision variables.

In this case, we do not display the explicit version of the model but directly report the solution and critically assess it. The solution states that $x_1 = 0$ and $x_2 = x_3 = x_4 = 1$. Hence, in the first stage the bounty hunter should let Zorgoiln the Zombie go and defeat Henry the Hermit Crab, the Ghost of your past, and Marion of the Haron. This implies spending 14 life points out of 16 and accruing 51 gold coins. **As there is no outlaw in the second stage who only reduces our life points by a single unit, the model suggests doing nothing in the second stage, regardless of the scenario**.

Differently than Example 14.1, we cannot compute a deterministic solution based on an "average" value, as shown in Table 14.4. Conversely, we can compute the 6 WS solutions. The results are reported in Table 14.6.

Table 14.6: WS solution to Example 14.2.

	x_1	x_2	x_3	x_4	y	Gold coins
$s = 1$	0	0	0	0	1,0	55
$s = 2$	0	0	0	0	1,0	55
$s = 3$	1	0	0	1	0,1	56
$s = 4$	1	0	1	1	1,1	55
$s = 5$	1	0	1	0	1,1	60
$s = 6$	1	0	1	0	1,1	60
Det. sol. AVG						56.8

We obtain that EVPI is equal to $56.8 - 51 = 5.8$. In particular, there are two scenarios (1 and 2) where it is recommended to let go of all the outlaws in the first stage so that we can the bounty hunter can allocate sufficient life points for Gerald the Gunk. The most profitable scenarios are 5 and 6, where 15 life points can be spent to muster 60 gold coins. **Notably, these two scenarios are equivalent in practice as, in mathematical terms and as mentioned in Example 10.1, the Big Brown Bear and the Frog Prince are equivalent**.

Coded example

The code used to model and solve Example 14.2 is available here.

We elaborate a bit more on the obtained solution to Example 14.2 in the **A note on the solution to Example 14.2** box.

A note on the solution to Example 14.2

The obtained solution to Example 14.2 seems greedy at first glance. It suggests that the bounty hunter should allocate all the available life points to defeat outlaws in the first stage, basically by-passing every potential outlaw they might face in the second stage. This is not the result of a greedy approach, but the result of the two-stage recourse problem given the choice of outlaws for the first and second stage. If the outlaws of the first stage were Zorgoiln the Zombie, Henry the Hermit Crab, Ghost of your past, and the Mummy, and two out of the four remaining were to appear during the second stage, we would witness some second stage decision variables to be unitary in the final solution. We encourage interested readers to check that by modifying the provided code.

Two-stage stochastic programming as a serious game

A serious game based on the 0-1 KP stochastic variant shown in Example 14.2 can be found here in the *Through hills and brambles* final set of cards.

14.3 Final words and recommended literature

Compared to a deterministic model, the number of combinations of a stochastic counterpart increases notably, and for relatively large models, tailor-made algorithmic approaches can be needed. **The effort of using a stochastic model should be justified by the application, while simple preliminary analyses using EVPI and VSS may provide evidence to investigate further. Also, the way the scenarios are generated can substantially impact the validity of the outcomes**. We refer to Kaut and Stein, 2003 for a review of methods. Multi-stage models can also be formulated for situations where more than two decision-making stages are permissible, tailored to the specific practical requirements of the problem at hand. One of the most prominent examples of stochastic models using continuous random variables is the newsvendor problem, where an analytical approach is performed. For interested readers, we refer to the seminal book Birge and Louveaux, 2011.

Bibliography

Adjacency List of States of the United States (US) (2024). https://writeonly.wordpress.com/2009/03/20/adjacency-list-of-states-of-the-united-states-us/. Accessed: 2024-02-19.

Ali, S., A. G. Ramos, M. A. Carravilla, and J. F. Oliveira (2022). On-line three-dimensional packing problems: A review of off-line and on-line solution approaches. *Computers & Industrial Engineering* 168, p. 108122.

Beliën, J., J. Colpaert, L. De Boeck, J. Eyckmans, and W. Leirens (2013). Teaching integer programming starting from an energy supply game. *INFORMS Transactions on Education* 13.3, pp. 129–137.

Birge, J. R. and F. Louveaux (2011). *Introduction to stochastic programming*. Springer Science & Business Media.

Brandão, J. C. S. and A. Mercer (1998). The multi-trip vehicle routing problem. *Journal of the Operational Research Society* 49, pp. 799–805.

Burrito Optimization Game (2024). https://www.gurobi.com/burrito-optimization-game/. Accessed: 2024-03-02.

Cacchiani, V., M. Iori, A. Locatelli, and S. Martello (2022a). Knapsack problems-An Overview of Recent Advances. Part I: Single Knapsack Problems. *Computers & Operations Research* 143, p. 105692.

Cacchiani, V., M. Iori, A. Locatelli, and S. Martello (2022b). Knapsack problems-An overview of recent advances. Part II: Multiple, multidimensional, and quadratic Knapsack problems. *Computers & Operations Research* 143, p. 105693.

Carter, M., C. C. Price, and G. Rabadi (2018). *Operations Research: a practical introduction*. Crc Press.

Cattaruzza, D., N. Absi, and D. Feillet (2016). The Multi-Trip Vehicle Routing Problem with Time Windows and Release Dates. *Transportation Science* 50.2, pp. 676–693.

Cochran, J. J. (2015). Extending "Lego® my Simplex". *INFORMS Transactions on Education* 15.3, pp. 224–231.

Columbia University: IEOR 6614 course notes (2024). https://www.columbia.edu/~cs2035/courses/ieor6614.S16/mst-lp.pdf. Accessed: 2024-02-22.

Cordeau, J.-F. and G. Laporte (2007). The dial-a-ride problem: models and algorithms. *Annals of Operations Research* 153, pp. 29–46.

Dantzig, G., R. Fulkerson, and S. Johnson (1954). Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America* 2.4, pp. 393–410.

Daş, G. S., F. Gzara, and T. Stützle (2020). A review on airport gate assignment problems: Single versus multi objective approaches. *Omega* 92, p. 102146.

Desaulniers, G., J. Desrosiers, and M. M. Solomon (2006). *Column Generation*. Vol. 5. Springer Science & Business Media.

Dumas, Y., J. Desrosiers, and F. Soumis (1991). The pickup and delivery problem with time windows. *European Journal of Operational Research* 54.1, pp. 7–22.

ELB learning (2024). https://blog.elblearning.com/the-key-difference-between-serious-games-and-gamification-in-elearning#:~:text=The%20key%20difference%20between%20the,educational%20value%20and%20not%20simply. Accessed: 2024-03-02.

Elshaer, R. and H. Awad (2020). A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers & Industrial Engineering* 140, p. 106242.

Essame, C. (2020). Developmental play: a new approach to understanding how all children learn through play. *Childhood Education* 96.1, pp. 14–23.

Gendreau, M., G. Laporte, and R. Séguin (1996). Stochastic vehicle routing. *European Journal of Operational Research* 88.1, pp. 3–12.

Guo, B., Y. Zhang, J. Hu, J. Li, F. Wu, Q. Peng, and Q. Zhang (2022). Two-dimensional irregular packing problems: A review. *Frontiers in Mechanical Engineering* 8, p. 966691.

Gurobi Optimization (2023). https://www.gurobi.com.

Gurobi Optimizer Reference Manual: Cuts (2023). https://www.gurobi.com/documentation/current/refman/cuts.html.

Hillier, F. S. and G. J. Lieberman (2015). *Introduction to Operations Research*. McGraw-Hill.

IBM ILOG CPLEX Optimization Studio (2023). https://www.ibm.com/products/ilog-cplex-optimization-studio.

Johnson, E. L., G. L. Nemhauser, and M. W. Savelsbergh (2000). Progress in Linear Programming-Based Algorithms for Integer Programming: An Exposition. *INFORMS Journal on Computing* 12.1, pp. 2–23.

Kallehauge, B., J. Larsen, O. B. Madsen, and M. M. Solomon (2005). *Vehicle Routing Problem with Time Windows*. Springer.

Kaut, M. and W Stein (2003). *Evaluation of Scenario-Generation Methods for Stochastic Programming*. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, Institut für Mathematik.

Lahyani, R., M. Khemakhem, and F. Semet (2015). Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research* 241.1, pp. 1–14.

Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley.

Li, F., B. Golden, and E. Wasil (2007). The open vehicle routing problem: Algorithms, large-scale test problems, and computational results. *Computers & Operations Research* 34.10, pp. 2918–2930.

Matplotlib: Visualization with Python (2024). https://matplotlib.org/. Accessed: 2024-03-15.

Miller, C. E., A. W. Tucker, and R. A. Zemlin (1960). Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM (JACM)* 7.4, pp. 326–329.

OR in an OB World (2024). https://orinanobworld.blogspot.com/2018/09/choosing-big-m-values.html#:~:text=First%2C%20branch%2Dand%2Dbound,(producing%20very%20loose%20bounds).. Accessed: 2024-03-09.

Paquay, C., M. Schyns, and S. Limbourg (2016). A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. *International Transactions in Operational Research* 23.1-2, pp. 187–213.

Pillac, V., M. Gendreau, C. Guéret, and A. L. Medaglia (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research* 225.1, pp. 1–11.

Postek, K., A. Zocca, J. Gromicho, and J. Kantor (2024). *Hands-On Optimization with Python*. Online. URL: https://mobook.github.io/MO-book/.

Salkin, H. M., K. Mathur, and R. Haas (1989). *Foundations of Integer Programming*. Elsevier Science Ltd.

The Beergame (2024). https://beergame.org/the-game/. Accessed: 2024-03-02.

Toth, P. and D. Vigo (2002). *The vehicle routing problem*. SIAM.

Trudeau, R. J. (2013). *Introduction to Graph Theory*. Courier Corporation.

Undiyaundeye, F. A. (2013). How children learn through play. *Journal of Emerging Trends in Educational Research and Policy Studies* 4.3, pp. 514–516.

Wikipedia: Branch & Cut (2024). https://en.wikipedia.org/wiki/Branch_and_cut. Accessed: 2024-02-06.

You've got Freight (2024). https://youvegotfreight.nl/en/. Accessed: 2024-03-02.

Acronyms

A
AI Artificial Intelligence. 4

B
BB Branch & Bound. xii, xv, xvi, 103–112, 114–131, 133–137, 145, 146, 150, 176, 228, 240, 254
BC Branch & Cut. xii, 133–136, 138, 140, 142, 144–146
BP Binary Program. 104, 105, 121, 122, 124, 125, 143, 150, 158, 166, 170, 176, 237, 239, 241, 246
BPP Bin Packing Problem. xiii, xvii, xx, 173, 179–190

C
CG Column Generation. xiii, 209, 229–231
CVRP Capacitated Vehicle Routing Problem. xx, 255, 257, 258

D
DAG Directed Acyclic Graph. xvii, 201–203
DARP Dial-A-Ride Problem. 264

E
ETA Estimated Time of Arrival. 269
EVPI Expected Value of Perfect Information. 277, 280, 281

G
GA Genetic Algorithm. 120
GAP Gate Assignment Problem. 46–48
GPS Global Positioning System. 269–271
GUB Generalized Upper Bound. 146

I
IATA International Air Transport Association. 203, 205
INFORMS INstitute For Operations Research and Management Science. 4
IP Integer Program. 104, 105, 115, 133, 135–137, 141

K
KP Knapsack Problem. xiii, xx, 121, 173–179, 281
KPI Key Performance Indicator. 3, 24, 25, 46, 124, 161, 176, 205, 276, 277

L
LIFO Last In First Out. 112
LNS Large Neighborhood Search. 120
LP Linear Program. xii, xiii, xvi, xix, 28, 59, 61, 63, 65, 67–69, 71, 73, 75, 77, 78, 80, 81, 88–91, 93, 97, 99–101, 103, 104, 113–117, 122, 123, 133–142, 146, 160, 209, 210, 212, 219, 223, 227, 229, 276

M
MCF Minimum Cost Flow. xiii, xx, 209, 232–235, 241
MILP Mixed Integer Linear Program. 104, 105, 108–110, 115, 118–120, 133–136, 138–140, 254, 259
MIR Mixed Integer Rounding. 146
MST Minimum Spanning Tree. xiii, xvii, xx, 209, 243–250

O
OR Operations Research. v–vii, x, xi, 3–10, 22, 45, 121, 124, 149, 173, 174, 193, 195, 200, 209, 227, 241, 243

P

PDP Pickup and Delivery Problem. xiii, 251, 263, 264
PMSP Parallel Machine Scheduling Problem. xii, xvi, xix, 149, 163–166

R
RLT Reformulation Linearization Technique. 146
RPG Role-Playing Game. 10

S
SAA Sample Average Approximation. 272, 275, 276
SCG Strong Chvatal–Gomory. 146
SDVRP Split Delivery Vehicle Routing Problem. xiii, 251, 260, 261
SMSP Single Machine Scheduling Problem. xii, xix, 149, 159–163
SP Shortest Path. xiii, xx, 209, 241–243, 270

T
TP Transportation Problem. xiii, xvii, xx, 209–225
TS Tabu Search. 120
TSP Traveling Salesman Problem. xiii, xx, 251–256, 265

U
ULD Unit Load Device. 188

V
VRP Vehicle Routing Problem. xiii, 251, 255, 257–265
VRPTW Vehicle Routing Problem with Time Windows. xiii, xx, 251, 258–260, 262, 263
VSS Value of Stochastic Solution. 277, 281

W
WS Wait-and-See. xx, 277, 280

Index

absolute value, 49
adjacencymatrix, 198
air cargo operations, 188
artificial variable, 71
assignment problem, 149
augmented form, 67

basic variable, 75
best bound, 107
best incumbent, 107
big-*M*, 33, 34, 37, 40

chromatic number, 237
closed Hamilton walk, 201
complete graph, 197
connected graph, 201
convex hull, 134
corner point, 66
cover inequality, 142
cutting stock problem, 187

decision variables, 23
degenerate solution, 150
degree of a vertex, 199
deterministic model, 269
directed graph, 195
dual simplex, 138
duality, 101, 230

either-or constraint, 34
Euler walk, 201

facility location problem, 169
feasible corner point, 67
feasible region, 60
fixed charge constraint, 40, 257

gamification, 7
Gomory cut, 136
graph coloring problem, 236
graph density, 197
graph theory, 193

Hamilton walk, 201
heuristic, 176, 254
Hungarian algorithm, 150

if-else statement, 53
infeasible corner point, 67
infeasible model, 65

K-out-of-N constraint, 37

lazy constraint, 250

max-min problem, 162
maximum flow problem, 225
min-max problem, 162
minimal cover, 143
minimum ratio test, 83
multigraph, 195
multiple optimal solutions, 62

non-basic variables, 75
North-West corner rule, 215

objective function, 24
open Hamilton walk, 201
optimality gap, 109
orthonormal matrix, 75

p-median problem, 166
piecewise linear function, 51

recourse problem, 270
root node, 105

sensitivity analysis, 100
serious game, 7
simplex tableau, 77
slack variable, 68
sparse matrix, 205
stochastic model, 269
stochasticity, 269
subgraph, 195
subtour elimination, 247
surplus variable, 72

transportation problem, 209
transportation simplex, 219

unbounded feasible region, 63
uncertainty, 269
undirected graph, 195

Vogel's method, 216

walk in a graph, 200
weighted graph, 203

zero-half cut, 145

Copyright of Figures

In this book, we tried our best to design our own figures, either using the TikZ LaTeX package or using the **Matplotlib** Python library (*Matplotlib: Visualization with Python* 2024). Oftentimes, these figures needed icons or other supporting images. Some other images were directly taken from websites or other sources. For this reason, we are listing in Table 1 all these supporting icons and images with the original source and the proper copyright.

Table 1: List of figures using images or logos from various sources. For each figure, we provide a brief description, the reference website, and the specified copyright.

Figure	Description	Copyright
Figure 1.1	The question mark and train icons were retrieved from iconoir.com	MIT
Figure 1.1	The map of the Milano subway system was retrieved from Wikipedia	CC BY-SA 4.0
Figure 2.1	Retrieved from Pixabay.com	CC0
Figure 4.2	Retrieved from Pixabay.com	CC0
Figure 4.5	Retrieved from Pixabay.com	CC0
Figure 5.1	Retrieved from Pixabay.com	CC0
Figure 5.5	The truck icon was retrieved from iconoir.com	MIT
Figure 10.6	Retrieved from Pixabay.com	CC0
Figure 11.7	Retrieved from Wikipedia	CC BY-SA 3.0
Figure 12.1	The industry and shop icons were retrieved from iconoir.com	MIT
Figure 12.9	Retrieved from Wikipedia	CC BY-SA 3.0

From theORy to application

learning to optimize with Operations Research in an interactive way

Alessandro Bombelli, Bilge Atasoy, Stefano Fazi, Doris Boschma

Synopsis: This book serves as a comprehensive roadmap for navigating the realm of Operations Research (OR). From laying down fundamental mathematical principles to crafting precise modeling techniques and their solution methods, it culminates in a panoramic view of OR models mirroring real-world operations. Delving into diverse applications-from assignment problems to network problems like graph coloring and minimum spanning trees, and navigating through routing problems that are very common in logistics-the book equips readers with practical insights. Each model is accompanied by meticulously detailed examples, seamlessly integrated with hyperlinked codes accessible via an open repository. Moreover, it introduces an engaging dimension with hyperlinks to three serious games replicating some cornerstone OR models, offering a playful yet educational environment for solo or group experimentation.






Alessandro Bombelli
Delft University of Technology, Faculty of Aerospace Engineering
Assistant professor in the Air Transport & Operations section

Bilge Atasoy
Delft University of Technology, Faculty of Mechanical Engineering
Associate professor in the Transport Engineering & Logistics section

Stefano Fazi
Delft University of Technology, Faculty of Technology, Policy, and Management
Assistant professor in the Transport and Logistics section

Doris Boschma
Delft University of Technology, Faculty of Technology, Policy, and Management
Worked as a designer and project leader at the Gamelab

© 2024 TU Delft Open
ISBN 978-94-6366-850-7
DOI: https://doi.org/ 10.59490/tb.94
textbooks.open.tudelft.nl

Cover image:
Close-up of the hexes defining the game board of Catan (previously known as The Settlers of Catan), one of the best-selling European board games. Figure by MorningbirdPhoto, retrieved here (https://pixabay.com/photos/board-game-settlers-of-catan-game-529581/) under license CC0