

International Series in
Operations Research & Management Science

Raymond Bisdorff

Algorithmic Decision Making with Python Resources

From Multicriteria Performance
Records to Decision Algorithms via
Bipolar-Valued Outranking Digraphs



 Springer

International Series in Operations Research & Management Science

Founding Editor

Frederick S. Hillier, Stanford University, Stanford, CA, USA

Volume 324

Series Editor

Camille C. Price, Department of Computer Science, Stephen F. Austin State University, Nacogdoches, TX, USA

Editorial Board Members

Emanuele Borgonovo, Department of Decision Sciences, Bocconi University, Milan, Italy

Barry L Nelson, IEMS, C210, Northwestern University, Evanston, IL, USA

Bruce W. Patty, Veritec Solutions, MILL VALLEY, CA, USA

Michael Pinedo, Stern School of Business, New York University, New York, NY, USA

Robert J. Vanderbei, Princeton University, Princeton, NJ, USA

Associate Editor

Joe Zhu, Foisie Business School, Worcester Polytechnic Institute, Worcester, MA, USA

Uncorrected Proof

Algorithmic Decision Making with Python Resources

24

25

26

From Multicriteria Performance Records
to Decision Algorithms via Bipolar-Valued
Outranking Digraphs

27

28

29



ISSN 0884-8289 ISSN 2214-7934 (electronic)

International Series in Operations Research & Management Science

ISBN 978-3-030-90927-7 ISBN 978-3-030-90928-4 (eBook)

32

<https://doi.org/10.1007/978-3-030-90928-4>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG.
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

*This book is dedicated to my colleague and
dear friend, the late Prof. Marc Roubens.*

49

50

The reader will find in this monograph a series of tutorials and advanced topics originally written over the last decade as documentation parts for the DIGRAPH3 collection of Python modules. These programming resources— like the `outrankingDigraphs` module—were essentially used both for the computational verification of decision algorithms and for the preparation and illustration of a master’s course on algorithmic decision theory taught at the University of Luxembourg from 2010 to 2020. Some resources, like the `randomNumbers` module, served for preparing and illustrating the lectures of a computational statistics course. Curious readers will also discover some resources— the `arithmetics` module—used for preparing and illustrating a first semester course on discrete mathematics.

The DIGRAPH3 Python programming resources are useful in the field of algorithmic decision theory and more specifically for the outranking approach of multiple-criteria decision aiding (MCDA). In this latter scientific field, we address essentially three kinds of usage.

First, we present algorithms and illustrate computing tools for solving either a multiple-criteria first choice selection or a dual last choice rejection problem. We also tackle the problem of how to list a set of items with multiple incommensurable performance criteria either from the best to the worst (ranking problem) or from the worst to the best (ordering problem). Finally, we present order-statistical algorithms for relative or absolute quantiles rating of multiple-criteria performance records.

It is necessary to mention that the DIGRAPH3 resources do not provide a professional Python software library. The collection of Python modules I describe in this book was not built following any professional software development methodology. The design of classes and methods was kept as simple and elementary as was opportune for the author. Sophisticated and cryptic overloading of classes, methods and variables is more or less avoided all over. A simple copy, paste and ad hoc customisation development strategy was generally preferred. As a consequence, the DIGRAPH3 modules keep a large part of independence. Furthermore, the development of the DIGRAPH3 modules being spread over two decades, the programming style did evolve with growing experience and the changes and enhancement coming

up with the ongoing new releases of the standard Python3 libraries. The required 33 backward compatibility necessarily introduced so with time some notation and 34 programming technique changes. 35

The purpose of this book is to present in a single monograph the scientific 36 enhancements the author has contributed over the past two decades to the outranking 37 approach-based multiple-criteria decision aiding field and that are either left 38 unpublished or published in very specialised media only and difficult to access. 39

This monograph should provide the reader with a self-contained series of 40 tutorials which explain how to solve multiple-criteria selection, as well as ranking or 41 rating decision problems. If successful in this aim, the curious reader will effectively 42 install the DIGRAPH3 programming resources on their laptop and try out and redo 43 for themselves the proposed computations. 44

The material in this book is valuable for master's students and doctoral candidates 45 in computer science, mathematics, engineering sciences or computational man- 46 agement sciences taking a course on algorithmic decision theory, multiple-criteria 47 decision aiding or decision analysis. Some experience in computer programming, in 48 particular with Python, will assist the reader, but it is not a prerequisite. The many 49 coding examples shown throughout the text are purposely kept elementary from a 50 programming point of view. 51

Chapters presenting algorithms for ranking multiple-criteria performance records 52 from best to worst—especially when facing big performance tableaux—will be of 53 computational interest for designers of web recommender systems. 54

Similarly, the relative and absolute quantiles-rating algorithms, discussed and 55 illustrated in several chapters, will be of practical interest for public or private 56 performance auditors. 57

Finally, the monograph does not provide any mathematical developments or 58 proofs. Those readers interested in the mathematical background of our decision 59 algorithms are invited to consult the references provided at chapter level. Full 60 texts of most of these references may be downloaded from the open access 61 <https://orbi.lu/> repository of the University of Luxembourg. 62

Acknowledgements

63

This monograph contains many ideas, methods and tools that are not the author's 64 alone. They have been shared with and enhanced by friends, colleagues and many 65 students. To name the most relevant: *Pascal Bouvry, Denis Bouyssou, Luis C Dias, 66 Claude Lamboray, Patrick Meyer, Vincent Mousseau, Alex Olteanu, Marc Pirlot, 67 the late Bernard Roy, Ulrich Sorger, Alexis Tsoukias, Thomas Veneziano and, 68 especially, the late Marc Roubens.* 69

The UL HPC team, and more specifically *Valentin Plugaru* and *Sébastien Varette*,
helped with mastering the multiprocessing experiments on the HPC platforms. The
Springer publishing team very kindly assisted me eventually with the manuscript
preparation.

The help by everybody is gratefully acknowledged.

Luxembourg, Luxembourg
Autumn 2021

Raymond Bisdorff 75
76

Introduction

1

The Editing Strategy

2

In the five parts of this monograph, the reader will find several series of tutorials 3 and advanced topics that present and illustrate computational methods and tools 4 mainly useful in the field of multiple-criteria decision aiding and decision analysis. 5 These methods and tools were designed and implemented first in Python2 and 6 then in Python3 by the author over the last two decades for supporting both the 7 computational verification and validation of decision algorithms as well as the 8 preparation and illustration of a master's course on algorithmic decision theory. 9

Each chapter illustrates a specific preference modelling aspect, like building 10 a best choice recommendation, ranking or rating a set of potential decision 11 alternatives, or computing the winner of an election. In order to keep parts and 12 chapters more or less self-contained, definitions and explanations of major concepts, 13 like bipolar-valued digraphs, multiple-criteria performance tableaux and outranking 14 situations, may appear several times in the monograph. 15

Explicit Python programming examples, purposely kept elementary, are shown 16 in numerous terminal session style listings. A complete list of the numbered listings, 17 shown over all the chapters, is printed in the Appendix. These programming 18 examples were all checked against errors with the `doctest` module of the standard 19 Python3 library and should work effectively as such either, in a Python3 interactive 20 terminal console, or for sure in an `ipython` console.¹ Note that the layout of 21 console `print(...)` outcomes has been edited in some listings for easing their 22 reading. Some chapters will rely on a given data file that is made available in the 23 `examples` directory of the DIGRAPH3 resources. 24

For similarly easing their reading, most chapters do not provide mathematical 25 developments and proofs. Readers interested in such details are invited to consult 26 the references listed separately at the end of each chapter. The author's references 27

¹ IP[i]: IPython interactive computing, <https://ipython.org>.

provide full text access to preprints on the open access <https://orbilu.uni.lu/> repository of the University of Luxembourg. 28
29

Readers interested in the technical aspects of the organisation and implementation of the collection of DIGRAPH3 Python modules are invited to consult the extensive reference manual: <https://digraph3.readthedocs.io/en/latest/techDoc.html>, 31
32 assisted by a search page <https://digraph3.readthedocs.io/en/latest/search.html> covering the whole DIGRAPH3 documentation. 33
34

Organisation of the Book 35

The content of the monograph is divided into five parts. 36

Part I presents three chapters introducing the DIGRAPH3 programming resources and the main formal objects discussed in this book, namely *bipolar-valued digraphs* and, in particular, *outranking digraphs*. 37
38
39

In Chap. 1, the reader will gain contact with the DIGRAPH3 Python resources. 40
First are given the installation instructions and the list of the main DIGRAPH3 41
Python modules with their purpose. A Python terminal session using the root 42
digraphs module eventually illustrates how to generate, save and inspect a 43
random crisp digraph. 44

Chapter 2 introduces the bipolar-valued digraph model—the root type of all 45
our digraph models. A randomly bipolar-valued digraph instance is generated. 46
Drawing the digraph, separating its asymmetric and symmetric parts, or its border 47
and inner parts, is illustrated. The initial digraph instance may be reconstructed by 48
epistemic disjunctive fusion from these respective parts. Dual, converse and co-dual 49
transforms as well as symmetric and transitive closures are presented. Complete, 50
empty and indeterminate digraphs are eventually presented. 51

Chapter 3 presents the bipolar-valued outranking digraph—the main formal 52
object used and discussed in this monograph. After illustrating its hybrid type— 53
it is conjointly a multiple-criteria performance tableau and a bipolar-valued relation 54
modelling outranking situations between the given performance records—pairwise 55
comparisons and the recoding of the digraph characteristic valuation are illustrated. 56
The co-dual transform of the outranking digraph renders the corresponding strict 57
outranking digraph, i.e. its asymmetric part. 58

Part II illustrates in eight methodological chapters multiple-criteria performance 59
evaluation models and decision algorithms. These chapters are mostly problem 60
oriented. 61

Chapter 4 presents the RUBIS best choice recommender system. The approach 62
is illustrated with building a best office-location recommendation. We show how to 63
explore a given performance tableau and compute the corresponding outranking 64
digraph. After presenting the pragmatic principles that govern our best choice 65
recommendation algorithm we solve the best office-location choice problem. 66

Chapter 5 illustrates a way of creating a new multiple-criteria performance 67
tableau by editing a given template file containing five decision alternatives, three 68

AQ3

AQ4

decision objectives and six performance criteria. We discuss in detail how to edit 69 the decision alternatives, the decision objectives, the family of performance criteria, 70 and, finally, the evaluations of the decision alternatives on the performance criteria. 71

Chapter 6 describes the DIGRAPH3 resources for generating random multiple- 72 criteria performance tableaux. These random performance tableaux instances, 73 mainly meant for illustration and training purposes, were serving the preparation 74 and illustration of the algorithmic decision theory course lectures. The random 75 generators propose several useful models like a cost-benefit tableau, a three 76 objectives—economic, societal and environmental—tableau, and an academic per- 77 formance tableau. 78

Chapter 7 is more specifically devoted to handling linear voting profiles and 79 computing the winner of such election results like the simple majority or the 80 instant runoff winner. By following CONDORCET’s recipe, we consider pairwise 81 comparisons of election candidates and balance the number of times the first 82 beats the second against the number of times the second beats the first in order 83 to obtain a majority margins digraph, in fact a bipolar-valued digraph. When the 84 voters express contradictory linear voting profiles, one may naturally observe cyclic 85 social preferences without seeing any paradox in this situation. Finally, the chapter 86 presents a more politically realistic generator for random linear voting profiles by 87 taking into account pre-election polls. 88

Chapter 8 introduces several algorithms for solving multiple-criteria ranking 89 problems via bipolar-valued outranking digraphs. The COPELAND, NETFLOWS, 90 KEMENY, SLATER, KOHLER and the RANKEDPAIRS ranking rules are illustrated 91 with the help of a random outranking digraph. The fitness of their respective ranking 92 result is measured with a bipolar-valued version of KENDALL’s ordinal correlation 93 index. 94

Chapter 9 applies order statistics for sorting a set X of n potential decision 95 alternatives, evaluated on m incommensurable performance criteria, into q quantile 96 equivalence classes. The sorting algorithm is based on pairwise outranking charac- 97 teristics involving the quantile class limits observed on each criterion. Thus, we may 98 implement a weak ordering algorithm of complexity $O(nmq)$. 99

Chapter 10 addresses the problem of rating multiple-criteria performance records 100 of a set of potential decision alternatives with respect to performance quan- 101 tiles learned from similar decision alternatives observed in the past. We show 102 how to incrementally compute performance quantiles from incoming performance 103 tableaux. New performance records may now be rated with respect to such historical 104 quantile norms. 105

Chapter 11 tackles the ranking of big multiple-criteria performance tableaux with 106 thousands or millions of records. To effectively compute rankings from performance 107 tableaux of these sizes, the chapter proposes a collection of C-compiled and 108 optimised modules that may be run on Linux Debian HPC clusters as available, 109 for instance, at the University of Luxembourg. 110

Part III delivers three realistic algorithmic decision-making case studies. 111

Chapter 12 presents a case study concerning the building of a best choice rec- 112 ommendation for Alice, a German student who wants some advice concerning the 113

choice of her future university studies. We present Alice's performance tableau— 114
 potential foreign language study programs, her decision objectives, performance 115
 criteria and performance evaluations—and build a best choice recommendation for 116
 her. A thorough robustness analysis confirms a very best choice. 117

In Chap. 13 we are resolving with our DIGRAPH3 resources a ranking decision 118
 problem based on published data from the Times Higher Education (THE) World 119
 University Rankings 2016 by Computer Science (CS) subject. We first have a look 120
 into the THE multiple-criteria ranking data with short Python scripts. In a second 121
 section, we relax the commensurability hypothesis of the ranking criteria and show 122
 how to similarly rank with multiple incommensurable performance criteria of solely 123
 ordinal significance. A third section is finally devoted to introduce quality measures 124
 for qualifying ranking results. 125

Chapter 14 presents and discusses how to rate with the help of our DIGRAPH3 126
 resources the apparent student enrolment quality of higher education institutions. 127
 The multiple-criteria performance tableau we use is inspired by a 2004 student sur- 128
 vey published by DER SPIEGEL magazine and concerning nearly 50,000 students, 129
 enrolled in 1 of 15 popular academic subjects, like German studies, life sciences, 130
 psychology, law or computer science. 131

In Chap. 15, we propose a series of decision problems of various difficulties 132
 which may serve as exercises and exam questions for an algorithmic decision theory 133
 or multiple-criteria decision analysis course. They cover selection, ranking and 134
 rating problems. 135

Part IV presents in five chapters more advanced topics showing some pearls of 136
 bipolar-valued epistemic logic. 137

Starting from a motivating decision problem about how to list, from the best 138
 to the worst, a set of movies that are star-rated by journalists and movie critics, 139
 Chap. 16 shows that KENDALL's ordinal correlation index tau can be extended to 140
 a relational bipolar-valued equivalence measure of bipolar-valued digraphs. This 141
 finding gives way, on the one hand, to measure the fitness and fairness of multiple- 142
 criteria ranking rules. On the other hand, it provides a tool for illustrating preference 143
 divergences between decision objectives and/or performance criteria. 144

We illustrate in Chap. 17, first, the concept of graph kernel, i.e. maximal 145
 independent set of vertices. In non-symmetric digraphs, the kernel concept becomes 146
 richer and separates into initial and terminal kernels. In, furthermore, lateralized 147
 outranking digraphs, initial and terminal kernels become separate and may deliver 148
 suitable first resp. last choice recommendations. After commenting the tractability 149
 of kernel computations, we close the chapter with the solving of bipolar-valued 150
 kernel equation systems. 151

In Chap. 18 we propose to link a qualifying significance majority for outranking 152
 situations with a required $\alpha\%$ -confidence level. We model therefore the significance 153
 weights as random variables following more or less widespread distributions around 154
 a mean value that corresponds to the given deterministic significance weights. As 155
 the bipolar-valued random credibility of an outranking situation hence results from 156
 the simple sum of positive or negative independent random variables, we can apply 157

the central limit theorem (CLT) for computing the bipolar-valued likelihood that the expected significance majority margin will indeed be positive, respectively negative. 158 159

In Chap. 19, we study the robustness of the outranking digraph when the criteria 160 significance weights faithfully indicate solely an order of importance. The required 161 cardinal significance weights of the performance criteria represent actually the 162 ‘Achilles’ heel of the outranking approach. Rarely will indeed a decision maker be 163 cognitively competent for suggesting precise decimal-valued criteria significance 164 weights. This approach leads furthermore to the concept of unopposed or Pareto 165 efficient multi-objective choices. 166

In a social choice context, where decision objectives would match different 167 political parties, such Pareto efficient choices represent in fact multipartisan social 168 choices. Chapter 20 shows that they may judiciously deliver the primary selection 169 in a two-stage election system. The outranking model is based on bipolar approvals- 170 disapprovals of “*at least as well evaluated as*” statements. A similar approach is put 171 into practice with bipolar approval-disapproval voting systems. When converting 172 such approval-disapproval voting ballots into corresponding performance records, 173 one obtains a $(-1, 0, 1)$ -valued evaluative voting system. We eventually show in 174 this chapter that, in such bipolar voting systems, the election winner tends to be 175 among the more or less multipartisan candidates. 176

Part V illustrates in three chapters computational resources for working with 177 simple undirected graphs. 178

Chapter 21 introduces bipolar-valued undirected graphs and illustrates several 179 special graph models and algorithms like Q-coloring, maximal independent set 180 (MIS) and clique enumeration, line graphs and maximal matchings, grid graphs, 181 and n-cycle graphs with their non-isomorphic MISs. 182

Chapter 22 specifically addresses working with tree graphs and graph forests. We 183 illustrate how to generate and recognise random tree graphs and how to compute 184 the centres of a tree and draw a rooted and oriented tree. Finally, algorithms for 185 computing spanning trees and forests are presented. 186

Chapter 23 eventually presents some famous classes of BERGE graphs, namely 187 comparability, interval, permutation and split graphs. We first present an example of 188 an interval graph which is at the same time a triangulated, a comparability, a split 189 and a permutation graph. The importance of being an interval graph is illustrated 190 with *Claude Berge*’s mystery story. We discuss furthermore the generation of 191 permutation graphs and close with how to recognise that a given graph is in fact 192 a permutation graph. 193

Highlights

194

Contrary to what is generally thought, it is the preparation of the multiple-criteria 195 performance tableau that takes most of the decision analysis time, not running any 196 decision algorithms. Designing adequate performance evaluating criteria functions 197 for each decision objective and collecting meaningful and precise evaluations is 198

crucial for the success of the decision-making. This is a very critical and essential 199
step. Chapters 4, 5 and 12 illustrate and discuss in detail coherent multiple-criteria 200
performance tableaux. In order to discover more examples of potential performance 201
tableaux, we provide in Chap. 6 random generators for several common kinds of 202
performance tableaux. 203

Once the multiple-criteria performance tableau is ready, the thrilling step of 204
discovering the resulting outranking relation starts. Are there many chordless out- 205
ranking circuits? What is its degree of symmetry? What is its degree of transitivity? 206
If the number of potential decision alternatives is small—less than 30—can one try, 207
in the case of a selection problem, to compute prekernels in order to find potential 208
first- or last-choice decision alternatives? Chapters 4, 12 and 17 are illustrating and 209
discussing this challenging computational problem. 210

Comparing various ranking rules working on bipolar-valued outranking relations 211
constructed from performance tableaux of various kinds, cost-benefit, 3-objectives 212
and academic a.-o., has made us confident about the fact that convincing criteria for 213
judging the quality of a ranking result may not to be found alone by mathematical 214
properties, like KEMENY optimality or CONDORCET consistency. More useful 215
seems to be the fair balancing of decision objectives and performance criteria. In 216
this respect, it is the NETFLOWS ranking rule which appears to be most effective and 217
often gives fairly balanced multiple-criteria rankings. Chapter 8 on ranking rules, the 218
ranking and rating case studies of Chaps. 13 and 14, and Chap. 16 on bipolar-valued 219
relational equivalence of digraphs illustrate and discuss this important topic. 220

The bipolar-valued epistemic logic, in which our decision algorithms are comput- 221
ing and expressing their decision solutions, provides effective assistance for coping 222
with missing data and imprecise performance evaluations. Chapters 14 and 16 223
illustrate this advantage. An efficient robustness analysis becomes furthermore 224
available for handling, on the one side, uncertain criteria significance weights 225
leading in Chap. 18 to $\alpha\%$ -confident outranking digraphs. On the other side, 226
Chap. 19 illustrates how to compute robust outranking digraphs and decision 227
solutions when solely ordinal criteria significance weights are given. In Chap. 20, 228
the same kind of robustness analysis proposes strategies for tempering plurality 229
tyranny effects in social choice problems by favouring multipartisan candidates, like 230
two-stage elections with multipartisan primary selection of candidates or bipolar 231
approval-disapproval voting systems. 232

Noticing the efficiency of the bipolar-valued epistemic logical framework for 233
handling outranking digraphs, we could not resist making in Chaps. 21 and 22 234
an excursion into the domain of simple undirected graphs and tree graphs. The 235
beautiful book *Algorithmic Graph Theory and Perfect Graphs* by M. Ch. Golumbic 236
gave eventually the opportunity to tackle in the last chapter some famous classes of 237
BERGE graphs. 238

It is my hope that the reader, by going on, will find the same astonishment 239
and enchantment as I experienced when discovering the simplicity, efficiency and 240
elegance of handling bipolar-valued outranking digraphs and graphs with Python 241
programming resources. Extending the bipolar-valued epistemic logical framework 242
to other computational science domains will prove valuable, I am sure, for many 243
future scientific works. 244

Uncorrected Proof

Contents

1

Part I Introduction to the DIGRAPH3 Python Resources

1	Working with the DIGRAPH3 Python Resources	3	2
1.1	Installing the DIGRAPH3 Resources	3	3
1.2	Organisation of the DIGRAPH3 Python Modules	5	4
1.3	Starting a DIGRAPH3 Terminal Session	6	5
1.4	Inspecting a Digraph Object	8	6
	References	12	7
2	Working with Bipolar-Valued Digraphs	13	8
2.1	Random Bipolar-Valued Digraphs	13	9
2.2	Graphviz Drawings	15	10
2.3	Asymmetric and Symmetric Parts	16	11
2.4	Border and Inner Parts	17	12
2.5	Fusion by Epistemic Disjunction	18	13
2.6	Dual, Converse, and Codual Digraphs	20	14
2.7	Symmetric and Transitive Closures	21	15
2.8	Strong Components	22	16
2.9	CSV Storage	23	17
2.10	Complete, Empty, and Indeterminate Digraphs	24	18
	Notes	25	19
	References	26	20
3	Working with Outranking Digraphs	29	21
3.1	The Hybrid Outranking Digraph Model	29	22
3.2	The Bipolar-Valued Outranking Digraph	32	23
3.3	Pairwise Comparisons	33	24
3.4	Recoding the Characteristic Valuation Domain	35	25
3.5	The Strict Outranking Digraph	35	26
	Notes	36	27
	References	37	28

Part II Evaluation Models and Decision Algorithms

4 Building a Best Choice Recommendation	41	29
4.1 What Office Location to Choose?	41	30
4.2 The Given Performance Tableau	43	31
4.3 Computing the Outranking Digraph.....	45	32
4.4 Designing a Best Choice Recommender System	47	33
4.5 Computing the RUBIS Best Choice Recommendation	48	34
4.6 Weakly Ordering the Outranking Digraph	50	35
Notes.....	52	36
References.....	54	37
5 How to Create a New Multiple-Criteria Performance Tableau	55	38
5.1 Editing a Template File	55	39
5.2 Editing the Decision Alternatives	57	40
5.3 Editing the Decision Objectives	58	41
5.4 Editing the Family of Performance Criteria.....	59	42
5.5 Editing the Performance Evaluations.....	61	43
5.6 Inspecting the Template Outranking Relation	63	44
References.....	66	45
6 Generating Random Performance Tableaux	67	46
6.1 Introduction	67	47
6.2 Random Standard Performance Tableaux	68	48
6.3 Random Cost-Benefit Performance Tableaux.....	71	49
6.4 Random Three Objectives Performance Tableaux.....	74	50
6.5 Random Academic Performance Tableaux	79	51
Reference	82	52
7 Who Wins the Election?	83	53
7.1 Linear Voting Profiles	83	54
7.2 Computing the Winner	85	55
7.3 The Majority Margins Digraph	86	56
7.4 Cyclic Social Preferences	88	57
7.5 On Generating Realistic Random Linear Voting Profiles	91	58
References.....	95	59
8 Ranking with Multiple Incommensurable Criteria	97	60
8.1 The Ranking Problem	97	61
8.2 The COPELAND Ranking	100	62
8.3 The NETFLOWS Ranking	102	63
8.4 KEMENY Rankings.....	104	64
8.5 SLATER Rankings	107	65
8.6 The KOHLER Ranking-by-Choosing Rule	109	66
8.7 The RANKEDPAIRS Ranking Rule	112	67
References.....	113	68

9 Rating by Sorting into Relative Performance Quantiles	115	69
9.1 Quantile Sorting on a Single Performance Criterion	115	70
9.2 Sorting into Quantiles with Multiple Performance Criteria	116	71
9.3 The Sparse Pre-ranked Outranking Digraph Model	120	72
9.4 Ranking Pre-ranked Sparse Outranking Digraphs	123	73
References	124	74
10 Rating-by-Ranking with Learned Performance Quantile Norms	125	75
10.1 The Absolute Rating Problem	125	76
10.2 Incremental Learning of Historical Performance Quantiles	126	77
10.3 Rating-by-Ranking New Performances with Quantile Norms	129	78
References	136	79
11 HPC Ranking of Big Performance Tableaux	137	80
11.1 C-compiled Python Modules	137	81
11.2 Big Data Performance Tableaux	138	82
11.3 C-implemented Integer-Valued Outranking Digraphs	139	83
11.4 The Sparse Implementation of Big Outranking Digraphs	141	84
11.5 Quantiles Ranking of Big Performance Tableaux	144	85
11.6 HPC Quantiles Ranking Records	147	86
References	147	87

Part III Evaluation and Decision Case Studies

12 Alice's Best Choice: A Selection Case Study	151	88
12.1 The Decision Problem	152	89
12.2 The Performance Tableau	153	90
12.3 Building a Best Choice Recommendation	156	91
12.4 Robustness Analysis	161	92
References	163	93
13 The Best Academic Computer Science Depts: A Ranking Case Study	165	94
13.1 The THE Performance Tableau	165	96
13.2 Ranking with Multiple Criteria of Ordinal Significance	171	97
13.3 How to Judge the Quality of a Ranking Result?	178	98
References	184	99
14 The Best Students, Where Do They Study? A Rating Case Study	187	100
14.1 The Rating Problem	187	101
14.2 The 2004 Performance Quintiles	189	102
14.3 Rating-by-Ranking with Lower-Closed Quintile Limits	191	103
14.4 Rating by Quintiles Sorting	196	104
References	198	105
15 Exercises	199	106
15.1 Introduction	199	107
15.2 Who Will Receive the Best Student Award? (§)	199	108

15.3	How to Fairly Rank Movies? (§)	200	109
15.4	What Is Your Best Choice Recommendation? (§§)	202	110
15.5	Planning the Next Holiday Activity (§§§)	203	111
15.6	What Is the Best Public Policy? (§§)	205	112
15.7	A Fair Diploma Validation Decision (§§§§)	205	113
	References	206	114
Part IV Advanced Topics			
16	On Measuring the Fitness of a Multiple-Criteria Ranking	209	115
16.1	Listing Movies from Best Star-Rated to Worst	209	116
16.2	KENDALL's Ordinal Correlation Tau Index	212	117
16.3	Bipolar-Valued Relational Equivalence	214	118
16.4	Fitness of Ranking Heuristics	217	119
16.5	Illustrating Preference Divergences	219	120
16.6	Exploring the “ <i>better rated</i> ” and the “ <i>as well as rated</i> ” Opinions	220	122
	References	223	123
17	On Computing Digraph Kernels	225	124
17.1	What Is a Graph Kernel?	225	125
17.2	Initial and Terminal Kernels	228	126
17.3	Kernels in Lateralized Digraphs	232	127
17.4	Computing First and Last Choice Recommendations	236	128
17.5	Tractability of Kernel Computation	239	129
17.6	Solving Kernel Equation Systems	242	130
	Notes	248	131
	References	249	132
18	On Confident Outrankings with Uncertain Criteria Significance Weights	133	
	18.1 Modelling Uncertain Criteria Significance Weights	251	134
	18.2 Bipolar-Valued Likelihood of Outranking Situations	253	136
	18.3 Confidence level Of Outranking Digraphs	255	137
	References	260	138
19	Robustness Analysis of Outranking Digraphs	261	139
19.1	Cardinal or Ordinal Criteria Significance Weights?	261	140
19.2	Qualifying the Stability of Outranking Situations	263	141
19.3	Computing the Stability Denotation of Outranking Situations	267	142
19.4	Robust Bipolar-Valued Outranking Digraphs	269	143
19.5	Characterising Unopposed Multiobjective Outranking Situations	272	145
19.6	Computing Pareto Efficient Multiobjective Choices	275	146
	References	277	147

20	Tempering Plurality Tyranny Effects in Social Choice	279	148
20.1	Introduction	279	149
20.2	Two-Stage Elections with Multipartisan Primary Selection	280	150
20.3	Bipolar Approval–Disapproval Voting Systems	287	151
20.4	Pairwise Comparison of Approval–Disapproval Votes	290	152
20.5	Three-Valued Evaluative Voting Systems	293	153
20.6	Favouring Multipartisan Candidates	296	154
	References	300	155
Part V Working with Undirected Graphs			
21	Bipolar-Valued Undirected Graphs	303	156
21.1	Implementing Simple Graphs	303	157
21.2	Q-Coloring of a Graph	306	158
21.3	MIS and Clique Enumeration	307	159
21.4	Line Graphs and Maximal Matchings	309	160
21.5	Grids and the ISING Model	311	161
21.6	Simulating METROPOLIS Random Walks	311	162
21.7	Computing the Non-isomorphic MISs of the n -Cycle Graph	313	163
	References	318	164
22	On Tree Graphs and Graph Forests	319	165
22.1	Generating Random Tree Graphs	319	166
22.2	Recognising Tree Graphs	322	167
22.3	Spanning Trees and Forests	324	168
22.4	Maximum Determined Spanning Forests	326	169
	References	328	170
23	About Split, Comparability, Interval, and Permutation Graphs	329	171
23.1	A ‘multiply’ Perfect Graph	329	172
23.2	Who Is the Liar?	331	173
23.3	Generating Permutation Graphs	333	174
23.4	Recognising Permutation Graphs	336	175
	References	341	176
Index		343	177

List of Figures

1

Fig. 1.1	<i>The tutorial crisp digraph.</i> The <code>exportGraphViz()</code> method is depending on drawing tools from https://graphviz.org . On Linux Ubuntu or Debian you may try ‘ <code>sudo apt-get install graphviz</code> ’ to install them. Under MacOS there are ready <code>dmg</code> installers available	2
Fig. 1.2	<i>Browsing the relation map of the tutorial digraph.</i> + Indicates a certainly valid and – indicates a certainly invalid relation. Here we find confirmed again that our random digraph instance, <code>dg</code> , is indeed a crisp, i.e. 100% determined irreflexive digraph instance	3
Fig. 1.3	The circulant [1,3] digraph and the 3 grid digraph	4
Fig. 2.1	<i>The tutorial random valuation digraph.</i> Double links are drawn in bold black with an arrowhead at each end, whereas single asymmetric links are drawn in black with an arrowhead showing the direction of the link. Notice the undetermined relational situation ($r(6 \ S \ 2) = 0.00$) observed between nodes 6 and 2. The corresponding link is marked in grey with an open arrowhead in the drawing	5
Fig. 2.2	Asymmetric and symmetric part of the tutorial random valuation digraph	6
Fig. 2.3	<i>Border</i> and <i>inner</i> part of a linear order oriented by <i>terminal</i> and <i>initial</i> kernels	7
Fig. 2.4	Border and inner part of the tutorial random valuation digraph <code>rdg</code>	18
Fig. 2.5	Symmetric and transitive closure of the tutorial random valuation digraph <code>rdg</code>	22
		29

Fig. 2.6	<i>The valued relation table shown in a browser window.</i>	30
	Positive arcs are shown in green and negative arcs in red.	31
	Indeterminate—zero-valued—arcs, like the reflexive	32
	diagonal ones or the arc between nodes 6 and 2, are	33
	shown in grey	24 34
Fig. 3.1	<i>The strict (codual) outranking digraph.</i> It becomes	35
	readily clear now from the picture that both alternatives	36
	a_1 and a_7 are <i>not outranked</i> by any other alternatives.	37
	Hence, a_1 and a_7 appear as <i>weak</i> CONDORCET	38
	winners and may be recommended as potential <i>best</i>	39
	decision alternatives in this illustrative preference	40
	modelling example	36 41
Fig. 4.1	Unranked heatmap of the office choice performance	42
	tableau	45 43
Fig. 4.2	<i>Bipolar-valued adjacency matrix.</i> In the resulting	44
	outranking relation we may notice, on the one hand, that	45
	Alternative D is <i>positively outranking</i> all other potential	46
	office locations. On the other hand, alternatives A (the	47
	most expensive) and C (the cheapest) are <i>not outranked</i>	48
	by any other location	46 49
Fig. 4.3	<i>Best office choice recommendation from strict</i>	50
	<i>outranking digraph.</i> Notice that location A (Ave) (the	51
	most expensive) is appearing <i>incomparable</i> to all the	52
	other alternatives	51 53
Fig. 4.4	<i>Ranking-by-choosing the potential office locations.</i> In	54
	this <i>ranking-by-choosing</i> method, where we operate	55
	the <i>epistemic fusion</i> of iterated (strict) first and last	56
	choices, compromise alternative Dom is now ranked	57
	before compromise alternative Gar. The overall partial	58
	ordering result shows again the important fact that the	59
	most expensive location, Ave, and the cheapest location,	60
	Ces, due to their contradictory performances appear	61
	both <i>incomparable</i> with most of the other alternatives	52 62
Fig. 4.5	<i>The internal stability of a best choice recommendation</i>	63
	<i>in question.</i> The only kernel of this digraph is the pair	64
	$\{a, d\}$; yet, it is an ambiguous recommendation, as	65
	$\{a, d\}$ is conjointly an outranking and outranked choice.	66
	If the instability of the best choice recommendation is,	67
	however, not considered a problem then the choice $\{a, b\}$	68
	shows the most convincing strict outranking quality and	69
	could be considered in priority for recommendation as	70
	potential best choice candidates	53 71
Fig. 5.1	<i>The template outranking digraph.</i> The digraph bod	72
	models in fact a <i>partial order</i> on the five potential	73
	decision alternatives	64 74

Fig. 5.2	COPELAND ranked heatmap of the template performance tableau	75 65 76
Fig. 5.3	NETFLOWS ranked heatmap of the template performance tableau	77 65 78
Fig. 6.1	Browser view on random performance tableau instance	70 79
Fig. 6.2	Unordered heatmap of a random Cost-Benefit performance tableau	80 74 81
Fig. 6.3	Browser view on the given random three-objectives performance tableau	82 77 83
Fig. 6.4	<i>The strict outranking digraph oriented by first and last choice recommendations.</i> Policy p5 appears incomparable—in a strict outranking sense—to all the other six policies, whereas policies p1, p2, and p7 appear strictly outranked	84 85 86 87 78 88
Fig. 6.5	Browser view on the COPELAND ranked performance tableau	89 79 90
Fig. 6.6	Ranking the students in a performance heatmap view	81 91
Fig. 7.1	<i>Visualising an election result.</i> In the Figure we notice that the <i>majority margins</i> digraph from our example linear voting profile models in fact a linear ranking of the candidates: $c_1 > c_3 > c_2$, the same actually as modelled by the BORDA scores (see Listing 7.4 on page 85)	92 93 94 95 96 88 97
Fig. 7.2	<i>Cyclic social preferences.</i> $c_1 > c_2 > c_3 > c_1$, for instance	98 89 99
Fig. 7.3	<i>Visualising a linear voting profile in a NETFLOWS ranked heatmap.</i> As the number of voters is usually much larger than the number of eligible candidates, the voting heatmap is by default transposed into a voters \times candidates view. Notice that the importance weights of the voters are <i>negative</i> , which means that the preference direction of the criteria (in this case the individual voters) is <i>decreasing</i> (min), i.e. goes from lowest (best) rank to highest (worst) rank	100 101 102 103 104 105 106 107 91 108
Fig. 7.4	<i>Browsing the majority margins.</i> Light green cells contain the positive majority margins, whereas light red cells contain the negative majority margins	109 110 94 111
Fig. 8.1	<i>The strict outranking relation \succsim.</i> The relation shown here is, for instance, <i>not transitive</i> : alternative a_8 outranks alternative a_6 and alternative a_6 outranks a_4 , however, a_8 does not outrank a_4 . Furthermore, alternatives a_6 , a_7 , and a_8 show a cyclic outranking relation	112 113 114 115 116 99 117

Fig. 8.2	<i>Drawing of the weak COPELAND ranking.</i> The drawing shows the skeleton of the weak ranking produced by the corresponding ties of the COPELAND scores	118 119 104 120
Fig. 8.3	<i>Epistemic disjunction of optimal KEMENY rankings.</i> It is interesting to notice that both KEMENY rankings only differ in their respective positioning of alternative a_8 ; either before or after alternatives a_9 , a_4 , and a_1	121 122 123 106 124
Fig. 8.4	<i>Epistemic disjunction of optimal SLATER rankings.</i> What precise SLATER ranking result should we hence adopt?	125 126 108 127
Fig. 9.1	The relation map of a sparse outranking digraph	122 128
Fig. 10.1	<i>Showing updated quartiles limits per criterion.</i> The 0.25 column shows the first quartile (Q1) limits, the 0.50 column shows the second quartile (Q2) limits and the 0.75 column shows the third quartile (Q3) limits. Column 0.00 (respectively, 1.00) shows the minimum (respectively, maximum) performance on each criterion	129 130 131 132 133 129 134
Fig. 10.2	<i>Heatmap of absolute quartiles ranking.</i> The quartile equivalence classes appear lower-closed. No alternative is rated into the Q1 class ([0.00 – 0.25[) and no alternative is rated into the Q4 class ([0.75 – 1.00])	135 136 137 133 138
Fig. 10.3	Absolute quartiles rating digraph	134 139
Fig. 10.4	<i>Absolute deciles rating.</i> Decision alternatives a_{1001} and a_{1010} are now, as expected, rated in the 6th decile (D6), respectively, in the 7th decile (D7)	140 141 135 142
Fig. 11.1	<i>Sparse quartiles-sorting decomposed outranking relation (extract).</i> Legend: outranking for certain (\top); outranked for certain (\perp); more or less outranking (+); more or less outranked (-); indeterminate ()	143 144 145 143 146
Fig. 11.2	<i>HPC-UL Ranking Performance Records (Spring 2018).</i> On the big memory equipped Gaia-183 node with 64 cores we were able to rank one million, respectively, 6 million decision alternatives in about 2 minutes, respectively, 41 minutes	147 148 149 150 147 151
Fig. 12.1	Alice D., 19 years old German student finishing her secondary studies in Köln (Germany), desires to undertake foreign languages studies	152 153 151 154
Fig. 12.2	Alice's performance criteria	154 155
Fig. 12.3	Heatmap of Alice's performance tableau	155 156
Fig. 12.4	COPELAND ranked outranking relation map	157 157

Fig. 12.5	<i>Alice's best choice recommendation.</i> We notice that the <i>Graduate Interpreter</i> studies come first, followed by the <i>Qualified Translator</i> studies. Last come the <i>Chamber of Commerce</i> 's specialised studies. This confirms again the high significance that Alice attaches to the <i>attractiveness</i> of her further studies and of her future profession (see criteria AS and AP in Fig. 12.3 on page 155)	159
		164
Fig. 12.6	<i>Comparing the first and second best-ranked study programs.</i> The Köln alternative is <i>at least as well evaluated as</i> the Saarbrücken alternative on all the performance criteria, except the <i>Annual income</i> (of significance 2/24). Conversely, the Saarbrücken alternative is clearly <i>outperformed</i> from the geographical (0/6) as well as from the financial perspective (2/6)	165
		171
Fig. 12.7	<i>Unopposed partial ranking of the potential study programs.</i> Again, when <i>equally significant</i> performance criteria are assumed per decision objective, we observe in the figure here that $I - FHK$ remains the stable best choice, <i>independently</i> of the actual importance weights that Alice may wish to allocate to her four decision objectives	172
		178
Fig. 13.1	The THE ranking criteria	171
Fig. 13.2	Relation map of the robust outranking relation	180
Fig. 13.3	3D PCA plot of the pairwise criteria correlation table	181
Fig. 13.4	Extract of a heatmap browser view on the NETFLOWS ranking result	182
		183
Fig. 14.1	<i>Student enrolment quality scores per subject.</i> The light green, respectively, light red, figures indicate highest, respectively, lowest, score among the five universities	184
		186
Fig. 14.2	The fifteen academic subjects taken into account for assessing the student enrolment quality	187
		188
Fig. 14.3	Heatmap view of the quintiles rating-by-ranking result	190
Fig. 14.4	Drawing of the quintiles rating-by-ranking result	192
Fig. 14.5	<i>Disjunctive fusion of the KEMENY, COPELAND and NETFLOWS rankings.</i> They diverge in their rating-by-ranking of universities U2, U3, and U4	193
		195
Fig. 15.1	Star-rating of movies from February 2003	194
Fig. 16.1	Star-ratings of movies from September 2007	195
Fig. 16.2	Star-ratings of movies ranked with the NETFLOWS rule	196
Fig. 16.3	Pairwise valued correlation of the movie critics	197
Fig. 16.4	Asymmetric part of the " <i>at least as well star-rated as</i> " statements	198
		199
Fig. 16.5	3D PCA plot of the criteria ordinal correlation matrix	200
Fig. 16.6	Symmetric part of the " <i>at least as well star-rated as</i> " statements	201
		202
		222

Fig. 17.1	<i>A random MIS coloured in the random 3-regular graph r3g12. All non-blue vertices are covered by a blue vertex</i>	203 204 205
Fig. 17.2	<i>The dual of the chordless 6-circuit. No initial or terminal prekernel—weakly independent and dominant, respectively, absorbent subset of nodes—may be found in this kind of digraphs</i>	206 207 208 209 231
Fig. 17.3	Dual and converse transforms of the weak 6-circuit give the same digraph	210 211 231
Fig. 17.4	<i>A random digraph instance of order 7 and arc probability 0.3. The digraph instance is neither asymmetric ($a_3 \Leftrightarrow a_6$) nor symmetric ($a_2 \rightarrow a_1, a_1 \not\rightarrow a_2$), and the digraph is not transitive ($a_5 \rightarrow a_2 \rightarrow a_1$, but $a_5 \not\rightarrow a_1$)</i>	212 213 214 215 233 216
Fig. 17.5	A random digraph oriented by best covering initial and best covered terminal prekernels	217 235 218
Fig. 17.6	The performance tableau of a random outranking digraph instance	219 236 220
Fig. 17.7	<i>A random strict outranking digraph instance. All decision alternatives appear strictly better performing than alternative a_7. We call it a CONDORCET loser and it is an evident terminal prekernel candidate. On the other side, three alternatives: a_1, a_2, and a_4 are not dominated. They give together an initial prekernel candidate</i>	221 222 223 224 225 226 237 227
Fig. 17.8	<i>The strict outranking digraph oriented by its first and last choice recommendations. The grey arrows, like the one between alternatives a_4 and a_1, represent indeterminate preferential situations. Alternative a_1 appears hence to be rather incomparable to all the other, except alternative a_7</i>	228 229 230 231 232 238 233
Fig. 17.9	Heatmap with Copeland ranking of the performance tableau	234 239 235
Fig. 17.10	<i>Initial kernel $\{a_1, a_2, a_4\}$ restricted adjacency table. The outranking situation between alternatives a_4 and a_1 being indeterminate, this initial prekernel is only weakly independent</i>	236 237 238 244 239
Fig. 17.11	<i>Terminal prekernel $\{a_3, a_7\}$ restricted adjacency table. Again, we notice that this terminal prekernel is only weakly independent</i>	240 241 245 245 242
Fig. 17.12	<i>The strict outranking digraph oriented by its initial and terminal prekernels. Notice the indeterminate outranking situation between alternatives a_4 and a_1 and the same indeterminate outranking situation between alternatives a_3 and a_7</i>	243 244 245 246 246 246 246 247

Fig. 18.1	Four models of uncertain criteria significance weights	252	248
Fig. 18.2	Bipolar-valued outranking characteristic function	253	249
Fig. 18.3	Distribution of 10 000 random outranking characteristic values	255	251
Fig. 18.4	90%-confident strict outranking digraph oriented by its initial and terminal prekernels	252	253
Fig. 19.1	<i>Standard</i> versus <i>robust</i> strict outranking digraphs oriented by their initial and terminal prekernels	254	271
Fig. 19.2	Robust heatmap of the random 3-objectives performance tableau ordered by the NETFLOWS ranking rule	256	272
Fig. 19.3	Standard versus <i>unopposed</i> strict outranking digraphs oriented by first and last choice recommendations	258	276
Fig. 20.1	Relation table of multipartisan outranking digraph	283	260
Fig. 20.2	The linear ranking modelled by the majority margins digraph	287	261
Fig. 20.3	The bipolar-valued pairwise majority margins	292	263
Fig. 20.4	The pairwise <i>better approved than</i> majority margins	299	264
Fig. 21.1	Example simple graph instance	305	265
Fig. 21.2	3-Coloring and 2-coloring of the tutorial graph	307	266
Fig. 21.3	<i>A perfect maximum matching of the 8-cycle graph</i> Every vertex of the 8-cycle is adjacent to a matching edge. Odd cycle graphs do not admit any perfect matching	267	268
Fig. 21.4	Ising model of the 15×15 grid graph	312	271
Fig. 21.5	Symmetry axes of the four non-isomorphic MISs of the 12-cycle graph	317	273
Fig. 22.1	<i>Random tree graph instance of order 9.</i> One may distinguish vertices like v_1, v_2, v_4, v_5 , or v_9 of degree 1, called the <i>leaves</i> of the tree, and vertices like $v_3, v_6,$ v_7 , or v_8 of degree 2 or more, called the <i>nodes</i> of the tree	320	274
Fig. 22.2	Tree graph instance generated with the random PRÜFER code <code>['v5', 'v7', 'v2', 'v5', 'v3']</code>	321	278
Fig. 22.3	<i>Recognising a tree graph.</i> One may notice that vertex v_2 is actually situated in the <i>centre</i> of the tree with a neighbourhood depth of 2	323	281
Fig. 22.4	Drawing an oriented tree rooted at its centre	323	283
Fig. 22.5	Random spanning tree	324	284
Fig. 22.6	Random spanning forest	325	285
Fig. 22.7	Best determined spanning tree	327	286

Fig. 23.1	<i>A conjointly triangulated, comparability, interval, permutation and split graph</i>	287
		288
	In the figure here, one may readily recognise the essential characteristic of split graphs, namely being always splittable into two disjoint sub-graphs: an <i>independent choice</i> $\{v_6\}$ and a <i>clique</i> — $\{v_1, v_2, v_3, v_4, v_5, v_7, v_8\}$ —which explains their name	291
		292
		293
Fig. 23.2	Graph representation of the testimonies of the professors	332
		294
Fig. 23.3	The triangulated testimonies graph	333
		295
Fig. 23.4	The $[4,3,6,1,5,2]$ permutation graph	334
		296
Fig. 23.5	Minimal vertex colouring of the permutation graph	335
		297
Fig. 23.6	Coloured matching diagram of the permutation $[4,3,6,1,5,2]$	335
		298
		299
Fig. 23.7	The transitive orientation of the permutation graph	336
		300
Fig. 23.8	Random graph of order 8 generated with edge probability 0.4	337
		301
		302
Fig. 23.9	Transitive neighbourhoods of the graph g	338
		303
Fig. 23.10	<i>Transitive neighbourhoods of the dual graph $-g$</i>	304
	It is worthwhile noticing that the orientation of g is achieved with a single neighbourhood decomposition, covering all the vertices, whereas the orientation of the dual graph $-g$ here needs a decomposition into three subsequent neighbourhoods marked in black, red, and blue	305
		306
		307
		308
		309
		310
		311
Fig. 23.11	Isomorphic permutation graphs	340

List of Tables

1

Table 4.1	The potential new office locations	42	2
Table 4.2	The family of performance criteria	42	3
Table 4.3	Performance evaluations of the potential office locations	43	4
Table 10.1	Multi-criteria performances of two potential decision alternatives	125	5
Table 12.1	The potential study programs	152	7
Table 12.2	Alice's family of performance criteria	153	8
Table 14.1	Enrolment quality scores per academic scores	188	9
Table 15.1	Grades obtained by the students	200	10
Table 15.2	Performance evaluations of the potential TV sets	203	11
Table 15.3	The set of potential holiday activities	203	12
Table 15.4	The set of performance criteria	204	13
Table 15.5	The performance tableau	204	14

Listings

1

1.1	Generating a digraph instance	6	2
1.2	A stored digraph instance	7	3
1.3	Random crisp digraph object	8	4
1.4	Inspecting a <code>Digraph</code> object	10	5
1.5	Various <code>compute...()</code> methods	11	6
1.6	Circulant digraphs and $n \times m$ grid digraphs	11	7
2.1	Random bipolar-valued digraph instance	13	8
2.2	Example of random valuation digraph	14	9
2.3	Computing asymmetric and symmetric parts	16	10
2.4	Computing the asymmetric part of a bipolar-valued relation	16	11
2.5	Border and inner part of a linear order	17	12
2.6	Epistemic fusion of partial digraphs	18	13
2.7	Computing associated dual, converse and codual digraphs	20	14
2.8	Computing the dual, the converse, and the codual of a digraph	21	15
2.9	Symmetric and transitive closures	21	16
2.10	Computing the strong components in a digraph	22	18
2.11	Complete, empty, and indeterminate digraphs	24	19
3.1	Generating a random performance tableau	30	20
3.2	Inspecting the performance criteria	31	21
3.3	Inspecting the performance evaluations	31	22
3.4	Example of random bipolar-valued outranking digraph	32	23
3.5	Inspecting the valued adjacency table	32	24
3.6	Inspecting a pairwise multiple-criteria comparison	33	25
3.7	Pairwise comparison with considerable performance difference	34	26
3.8	Recoding the digraph valuation	35	28
4.1	Inspecting the <code>officeChoice</code> performance tableau	43	29
4.2	Inspecting the performance criteria	44	30

4.3	Computing a bipolar-valued outranking digraph.....	46	31
4.4	Computing the best choice recommendation.....	48	32
4.5	Inspecting pairwise comparison between alternatives G and D.....	49	33
4.6	Inspecting pairwise comparison between alternatives C and G.....	50	34
4.7	Ranking-by-choosing the outranking digraph.....	51	35
5.1	PerformanceTableau object template	55	36
5.2	Example of decision alternative description	57	37
5.3	Example of decision objectives' description	58	38
5.4	Example of performance criteria description	59	39
5.5	Example of cardinal <i>Costs</i> criterion	60	40
5.6	Editing performance evaluations.....	62	41
5.7	The template outranking relation	63	42
6.1	Generating a random performance tableau	69	43
6.2	Generating a random Cost-Benefit performance tableau	72	44
6.3	Generating a random three objectives performance tableau	75	45
6.4	Inspecting the three objectives	75	46
6.5	What is the public policy to recommend as best choice?	77	47
6.6	Generating a random academic performance tableau.....	80	48
6.7	Student performance summary statistics per course	80	49
6.8	Consensus quality of the students' ranking	82	50
7.1	Example of random linear voting profile	84	51
7.2	Showing linear voting profiles	84	52
7.3	Example instant run off winner	85	53
7.4	Example of BORDA rank scores	85	54
7.5	Rank analysis example with BORDA scores.....	86	55
7.6	Example of <i>Majority Margins</i> digraph	86	56
7.7	Example of cyclic social preferences	88	57
7.8	NETFLOWS ranked heatmap view on a voting profile	90	58
7.9	Rank analysis table with BORDA scores	90	59
7.10	Generating a linear voting profile with random polls	92	60
7.11	The uninominal and BORDA election winner	93	61
7.12	A majority margins digraph constructed from a linear voting profile	93	63
7.13	Ranking by iterating choosing the <i>first</i> and <i>last</i> remaining candidates	94	65
8.1	Random bipolar-valued strict outranking relation characteristics	98	67
8.2	Median cut polarised strict outranking relation characteristics	98	69
8.3	Computing a COPELAND ranking	101	70
8.4	Checking the ordinal quality of the COPELAND ranking	101	71
8.5	Computing a weak COPELAND ranking.....	102	72
8.6	Computing a NETFLOWS ranking	103	73

8.7	Checking the quality of the NETFLOWS ranking	103 74
8.8	Computing a KEMENY ranking	105 75
8.9	Optimal KEMENY rankings	105 76
8.10	Computing the epistemic disjunction of all optimal KEMENY rankings	77 105 78
8.11	Computing the consensus quality of the first KEMENY ranking	79 106 80
8.12	Computing the consensus quality of the second KEMENY ranking	81 107 82
8.13	Computing a SLATER ranking	107 83
8.14	Computing the epistemic disjunction of optimal SLATER rankings	84 109 85
8.15	Computing a KOHLER ranking	109 86
8.16	Ranking-by-choosing with iterated maximal NETFLOWS scores	87 110 88
8.17	Computing a RANKEDPAIRS ranking	112 89
9.1	Computing a quintiles sorting result	117 90
9.2	Inspecting the quantile limits	118 91
9.3	Computing a quintiles sorting result	118 92
9.4	Bipolar-valued sorting characteristics (extract)	119 93
9.5	Weakly ranking the quintiles sorting result	120 94
9.6	Computing a <i>pre-ranked</i> sparse outranking digraph	120 95
9.7	The quantiles decomposition of a pre-ranked outranking digraph	96 121 97
9.8	Functional binary relation characteristics	123 98
10.1	Computing performance quantiles from a given performance tableau	99 127 100
10.2	Printing out the estimated quartile limits	127 101
10.3	Generating 100 new random decision alternatives of the same model	102 128 103
10.4	Computing the absolute rating of 10 new decision alternatives	104 130 105
10.5	Performance tableau of the new incoming decision alternatives	106 130 107
10.6	Showing the limiting profiles of the rating quantiles	131 108
10.7	COPELAND ranking of new alternatives and historical quartile limits	109 131 110
10.8	Absolute quartiles rating result	132 111
10.9	Absolute deciles rating result	133 112
10.10	From deciles interpolated quartiles rating result	135 113
11.1	Big data performance tableau format	138 114
11.2	Constructing big bipolar-valued outranking digraphs	139 115

11.3	Constructing the sparse integer outranking digraph	141	116
11.4	The 6 components of a sparse outranking digraph.....	142	117
11.5	The <code>relation()</code> function of a sparse outranking digraph.....	143	118
11.6	Ranking the sparse integer outranking digraph	144	119
11.7	The ordered components of the sparse outranking digraph	145	120
11.8	Measuring the loss of quality with respect to the standard outranking digraph	121	
12.1	Alice's performance tableau	153	123
12.2	Computing Alice's outranking digraph.....	156	124
12.3	Inspecting polarised outranking situations	156	125
12.4	Alice's best choice recommendation	158	126
12.5	Alice's strict best choice recommendation	159	127
12.6	Weakly ranking by bipolar best-choosing and last-rejecting	160	128
12.7	Computing the 90% confident outranking digraph	161	129
12.8	Computing the 90%-confident best choice recommendation.....	161	130
12.9	Computing the unopposed outranking situations	163	131
13.1	Performance tableau of the.....	166	132
13.2	Printing the CS Departments.....	166	133
13.3	The THE ranking objectives	167	134
13.4	Computing the THE overall scores	169	135
13.5	Printing the ranked performance table.....	169	136
13.6	Inspecting the performance discrimination thresholds	171	137
13.7	Computing the robust outranking digraph.....	173	138
13.8	Inspecting outranking circuits	173	139
13.9	Showing the relation table with stability denotation.....	174	140
13.10	Computing a robust NETFLOWS ranking	175	141
13.11	Comparing the robust NETFLOWS ranking with the THE ranking	176	143
13.12	Comparing pairwise criteria performances	178	144
13.13	Measuring the quality of the NETFLOWS ranking result	179	145
13.14	Measuring the consensus quality of the NETFLOWS ranking result	180	147
13.15	Showing the ordinal correlation between the marginal criterion relations	148	
13.16	Computing the ordinal quality of the THE ranking	181	149
14.1	Inspecting stored historical performance quantiles	182	150
14.2	Estimated quintile limits of the 2004 survey	189	151
14.3	Showing the quintiling of the enrolment quality of the 5 universities	191	152
14.4	Computing the epistemic fusion of three rating-by-ranking results	192	153
14.5	Checking the consensus quality of the KEMENY ranking	194	155
14.6	Checking the consensus quality of the COPELAND ranking	195	157
		196	158

14.7	Showing quantiles sorting characteristics	197	159
14.8	Showing a quintiles rating-by-sorting result	197	160
16.1	Computing the average weighted number of stars per movie	211	161
16.2	Showing the movie from best- to worst-rated in a heatmap view	212	163
16.3	Computing a relational equivalence digraph	213	164
16.4	Two random bipolar-valued digraphs	214	165
16.5	Bipolar-valued equivalence digraph	215	166
16.6	Computing the ordinal correlation index from the equivalence digraph	216	167
16.7	Computing the valued ordinal correlation index	217	169
16.8	The bipolar-valued outranking digraph of the star-rated movies	217	171
16.9	Computing marginal criterion correlations with global NETFLOWS ranking	218	173
16.10	Computing the ordinal correlation between NETFLOWS and global outranking digraph	218	175
16.11	Bipolar ranking-by-choosing the movies	222	176
17.1	Generating a random 3-regular graph of order 12	226	177
17.2	Printing out all maximal independent sets of the random 3-regular graph	227	178
17.3	The prekernels of a complete digraph	228	180
17.4	The prekernels of the empty or indeterminate digraph	229	181
17.5	The prekernels of the 5-circuit digraph	230	182
17.6	The prekernels of the 6-circuit digraph	230	183
17.7	The prekernels of the dual of the 6-circuit digraph	231	184
17.8	The weak 6-circuit digraph	232	185
17.9	Generating a random digraph rd of order 7 and arc probability 0.3	233	186
17.10	Inspecting the properties of random digraph rd	234	188
17.11	Inspecting the prekernels of random digraph rd	234	189
17.12	Generating a random bipolar-valued outranking digraph	236	190
17.13	Computing the prekernels of the strict outranking digraph gcd	237	191
17.14	Computing a first and last choice recommendation from digraph gcd	237	193
17.15	Enumerating MISs by visiting only maximal independent choices (<i>A. Hertz</i>)	240	195
17.16	Generating all independent choices in a digraph	240	197
17.17	Computing dominant and absorbent prekernels	241	198
17.18	Verifying the kernel equation system on a tiny random digraph	242	199

17.19	Computing a dominant prekernel restricted adjacency table	244	201
17.20	Fixpoint iterations for initial prekernel $\{a_1, a_2, a_4\}$	245	202
18.1	Computing a 90% confident outranking digraph.....	256	203
18.2	90%-confident outranking relation with triangular distributed significance weights.....	204	257
18.3	99%-confident outranking relation	258	206
18.4	90%-confident outranking digraph with uniform variates	259	207
19.1	Generate a random 3-objectives performance tableau	261	208
19.2	The significance weights preorder	262	209
19.3	Example bipolar outranking digraph	263	210
19.4	Bipolar-valued outranking relation table with stability denotation	211	264
19.5	Comparison of alternatives p_2 and p_1	265	213
19.6	Comparison of alternatives p_7 and p_3	266	214
19.7	Computing a robust outranking digraph.....	269	215
19.8	Inspecting polarised outranking situations	270	216
19.9	Computing a robust performance heatmap view	271	217
19.10	Computing unopposed outranking situations	273	218
19.11	Computing unopposed outranking digraphs	274	219
19.12	Example of unopposed multiobjective outranking situation	275	220
19.13	Pareto efficient multiobjective choice	275	221
20.1	Example of a 3 parties voting profile	280	222
20.2	Converting a voting profile into a performance tableau	281	223
20.3	Computing unopposed multiobjective outranking situations	282	224
20.4	Computing unopposed multiobjective outranking situations	282	225
20.5	Recommending the secondary election winner	283	226
20.6	A divisive two-party example of a random linear voting profile	227	284
20.7	Example of ineffective primary multipartisan selection	285	228
20.8	Example of non-obvious secondary selection.....	286	229
20.9	Bipolar approval voting profiles	287	230
20.10	Inspecting an approval-disapproval ballot	288	231
20.11	Comparing the net approval and the NETFLOWS rankings.....	291	232
20.12	Computing the outranking digraph	293	233
20.13	Comparing the NETFLOWS and the Net Approval rankings	294	234
20.14	Computing a best social choice recommendation.....	294	235
20.15	A random approval-disapproval voting profile in a divisive political context.....	237	296
21.1	Generating a random graph instance	238	304
21.2	Stored instance of a random graph	240	304
21.3	Inspecting a graph instance	241	305
21.4	Conversion between graphs and digraphs	242	306
21.5	Computing a 3-coloring of the random graph g	243	306

21.6	Computing and printing the maximal independent sets of graph g	244
		307 245
21.7	Computing and printing the maximal independent sets of graph g	246
		308 247
21.8	Computing the line graph of the 5-cycle graph	248
21.9	Computing the MISs of the line graph of the 8-cycle graph.....	249
21.10	Computing maximum matchings in the 8-cycle graph	250
21.11	Simulating an Ising model on a 15×15 rectangular grid	251
21.12	Simulating random walks on a graph	252
21.13	Checking the quality of the MCMC sampler.....	253
21.14	Printing the transition probability distribution	254
21.15	Computing the MISs of the 12-cycle graph	255
21.16	Computing the MISs with the <code>perrinMIS</code> shell command	256
21.17	Computing the automorphism group generators	257
21.18	Computing the MIS orbits of the 12-cycle graph	258
22.1	Generating a random tree graph	259
22.2	Generating a tree graph with a random PRÜFER code	260
22.3	Recognising a tree graph	261
22.4	Computing the PRÜFER code of a tree graph instance.....	262
22.5	Computing the centres of a tree and drawing a rooted and oriented tree.....	263
		323 264
22.6	Generating uniform random spanning trees	265
22.7	Computing spanning forests over disconnected graphs.....	266
22.8	Generating randomly bipolar-valued graphs	267
22.9	Symmetric relation table	268
22.10	Computing best determined spanning forests	269
23.1	Testing perfect graph categories	270

AUTHOR QUERIES

- AQ1.** Please check the sentence starting “The required backward compatibility...” for clarity.
- AQ2.** Please check the sentence starting “The purpose of this book is to present...” for clarity.
- AQ3.** Please check if sentence starting “Drawing the digraph...” is okay
- AQ4.** Please check if edit to sentence starting “After illustrating its hybrid type...” is okay.
- AQ5.** in the sentence starting “In, furthermore, lateralized outranking...” please provide full form for term “resp.”.

Part I ₁

Introduction to the DIGRAPH3 Python ₂ Resources ₃

The first part contains three chapters for introducing the DIGRAPH3 software ₄ collection of Python programming resources. The first chapter is devoted to the ₅ installation of the DIGRAPH3 Python modules and running a first Python terminal ₆ session using the DIGRAPH3 ₃ resources. The second chapter introduces bipolar- ₇ valued digraphs, the root type of all available specialised digraphs. The third chapter ₈ finally introduces the main formal objects of this book, namely bipolar-valued ₉ outranking digraphs. ₁₀

Chapter 1	1
Working with the DIGRAPH3 Python Resources	2
	3

Contents	4
1.1 Installing the DIGRAPH3 Resources	3 5
1.2 Organisation of the DIGRAPH3 Python Modules	5 6
1.3 Starting a DIGRAPH3 Terminal Session	6 7
1.4 Inspecting a Digraph Object	8 8

Abstract The chapter is devoted to a first contact with the DIGRAPH3 Python resources. Following the installation instructions, we list the main Python modules with their purpose and eventually illustrate in a first Python terminal session how to generate, save, and inspect a random crisp digraph. 9
10
11
12

1.1 Installing the DIGRAPH3 Resources

Using the DIGRAPH3 Python modules is simple.¹ You only need to have installed on your system the Python programming language of version 3 (by default available under Linux and MacOS). 14
15
16

Several download options (easiest under Linux or MacOS) are given; either, by using a git client and clone a working copy from the `github.com` directory: 17
18

...\$ git clone <https://github.com/rbisdorff/Digraph3> 19

or from the `sourceforge.net` directory: 20

...\$ git clone <https://git.code.sf.net/p/digraph3/code> Digraph3 21

It is also possible, with a browser access, to download either, 22

- from the `github.com` link or 23
- from the `sourceforge.net` link, 24

¹ See the technical description of the DIGRAPH3 programming resources, Bisdorff (2021).

the latest distribution `zip` archive and extract it. 25

On Linux or MacOS, `.. $ cd` to the cloned or extracted `Digraph3` directory. 26
 The following shell command installs (with `sudo` !) the `DIGRAPH3` modules in the 27
 system-wide python environment: 28

```
.../Digraph3$ make install 29
```

Python-3.8 (or later) environment is recommended (see the `makefile` for 30
 adapting the `make install` command to your running python environment). 31

Whereas the following shell command installs the `DIGRAPH3` modules locally in 32
 an activated virtual python user environment: 33

```
.../Digraph3$ make installVenv 34
```

If the C-compiled modules for Big Data applications are required, it is necessary 35
 to previously install the `cython` compiler² package in the running Python environ- 36
 ment: 37

```
...$ python3 -m pip install cython 38
```

It is recommended to run a test suite: 39

```
.../Digraph3$ make tests 40
```

Test results are stored in the `Digraph3/test` directory. Notice that the 41
`python3 pytest` package is required: 42

```
...$ python3 -m pip install pytest 43
```

A verbose (with `stdout` not captured) `pytest` suite may be run as follows: 44

```
.../Digraph3$ make verboseTests 45
```

When the GNU parallel shell tool³ is installed and multiple cores are 46
 detected, the tests will be executed in multiprocessing mode: 47

```
.../Digraph3$ make pTests 48
```

Individual module `pytest` suites are also provided (see the `makefile`), like 49
 the one for the `outrankingDigraphs` module (see Chap. 2): 50

```
.../Digraph3$ make outrankingDigraphsTests 51
```

Dependencies

To be fully functional, the `DIGRAPH3` resources mainly need: 52

- The `graphviz` tools (Gansner and North 2000),⁴ and 54
- The `R` statistics resources⁵ to be installed; 55

² <https://cython.org>.

³ <https://www.gnu.org/software/parallel>.

⁴ <https://graphviz.org>.

⁵ <https://www.r-project.org>.

- When exploring digraph isomorphisms, the `nauty` isomorphism testing program is required (McKay and Piperno 2013). On Linux you may try: `sudo apt install nauty`. For MacOS, corresponding `dmg` installers are available for downloading; 56
- Two specific methods of the `OutrankingDigraph` class for clustering performance criteria or decision alternatives require furthermore the `calmat` matrix computing resource to be installed (see the `calmat` directory in the DIGRAPH3 resources). 60
61
62
63

1.2 Organisation of the DIGRAPH3 Python Modules

64

The main data handling modules of the DIGRAPH3 resources are the following:

- `digraphs`: main part of the DIGRAPH3 source code with the root `Digraph` class. 66
67
- `graphs`: resources for handling undirected graphs with the root `Graph` class and a bridge to the `digraphs` module resources. 68
69
- `perfTabs`: tools for handling multiple-criteria performance tableaux with root `PerformanceTableau` class. 70
71
- `outrankingDigraphs`: root module for handling outranking digraphs with the abstract root `OutrankingDigraph` class and the main `BipolarOutrankingDigraph` class.⁶ 72
73
74
- `votingProfiles`: classes and methods for handling voting ballots and computing election results with main `LinearVotingProfile` class. 75
76

Various random generators are provided by the following modules:

- `randomDigraphs`: various random digraph models like random crisp digraphs (`RandomDigraph` class) or random bipolar-valued digraphs (`RandomValuationDigraph` class). 78
79
80
- `randomPerfTabs`: various implemented random performance tableau models, like Cost-Benefit tableaux (`RandomCBPerformanceTableau` class) or 3-Objectives tableaux (`Random3ObjectivesPerformanceTableau` class). 81
82
83
84
- `randomNumbers`: additional random number generators, not available in the standard Python `random.py` library, like a discrete random variable (`DiscreteRandomVariable` class) or a Cauchy random variable (`CauchyRandomVariable` class) 85
86
87
88

⁶Notice that the `OutrankingDigraph` class defines a hybrid object type, inheriting conjointly from the `Digraph` class and the `PerformanceTableau` class.

Following modules provide tools for <i>sorting</i> , <i>ranking</i> , and <i>rating</i> problems:	89
1. <code>sortingDigraphs</code> : tools for solving sorting problems with the root <code>SortingDigraph</code> class and the main <code>QuantilesSortingDigraph</code> class;	90
2. <code>linearOrders</code> : tools for solving linearly ranking or ordering problems with the root <code>LinearOrder</code> class;	92
3. <code>transitiveDigraphs</code> : additional tools for handing transitive digraphs with root <code>TransitiveDigraph</code> class.	94
	95

Tools for specifically handling Big Data are eventually provided by the following modules:	96
	97

1. <code>performanceQuantiles</code> : incremental representation of large performance tableaux via binned cumulated density functions per criteria; Depends on the <code>randomPerfTabs</code> module.	98
2. <code>sparseOutrankingDigraphs</code> : sparse implementation design for bipolar-valued outranking digraphs of order >500.	101
3. <i>Cythonized</i> modules: C-compiled and optimised Python modules for handling big performance tableaux and bipolar outranking digraphs of order >1000.	103
	104

Readers interested in technical implementation details are invited to consult the reference manual of the DIGRAPH3 resources, where they will find the documentation and complete source code of all DIGRAPH3 modules, classes, and methods (Bisdorff 2021).	105
	106
	107
	108

1.3 Starting a DIGRAPH3 Terminal Session

After downloading the DIGRAPH3 resources, one may start an interactive Python3 terminal session in the <code>Digraph3</code> directory.	110
	111

```
$HOME/.../Digraph3$ python3
Python 3.9.6 (v3.9.6:db3ff76da1, Jun 28 2021, 11:49:53)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or
"license" for more information.
>>>
```

For exploring the classes and methods provided by the DIGRAPH3 modules enter the Python3 commands following the session prompts marked with `>>>` or `...`; the lines without a prompt are output from the Python3 interpreter. Python class names and Boolean parameters start by convention with a capital case; names of other Python objects, like modules, methods, and variables start with a lower case. All Python names and code are shown in a *typewriting* font.

Listing 1.1 Generating a digraph instance

```
1 >>> from randomDigraphs import RandomDigraph
2 >>> dg = RandomDigraph(order=5, arcProbability=0.5, \
```

```

3 ...                               seed=101)          126
4 >>> dg                         127
5 *----- Digraph instance description -----* 128
6 Instance class      : RandomDigraph 129
7 Instance name       : randomDigraph 130
8 Digraph Order       : 5             131
9 Digraph Size        : 12            132
10 Valuation domain   : [-1.00; 1.00] 133
11 Determinateness (%) : 100.00      134
12 Attributes         : ['actions', 'valuationdomain', 135
13                  'relation', 'order', 'name', 136
14                  'gamma', 'notGamma'] 137

```

In Listing 1.1 on the facing page we import, for instance, from the `randomDigraphs` module the `RandomDigraph` class in order to generate a random digraph object `dg` of order 5 and arc probability of 50%. The resulting digraph of *order* 5—number of nodes called (decision) *actions*—and *size* 12—number of arcs—is completely determined (see Line 11).

The content of `dg` may be saved in a file named `tutorialDigraph.py`.

```

1 >>> dg.save('tutorialDigraph')      144
2 *--- Saving digraph in file: <tutorialDigraph.py> ---* 145

```

with the following content:

Listing 1.2 A stored digraph instance

```

1 from decimal import Decimal          147
2 from collections import OrderedDict 148
3 actions = OrderedDict([            149
4     ('a1', {'shortName': 'a1',          150
5         'name': 'random decision action'}), 151
6     ('a2', {'shortName': 'a2',          152
7         'name': 'random decision action'}), 153
8     ('a3', {'shortName': 'a3',          154
9         'name': 'random decision action'}), 155
10    ('a4', {'shortName': 'a4',          156
11        'name': 'random decision action'}), 157
12    ('a5', {'shortName': 'a5',          158
13        'name': 'random decision action'}), 159
14 ])                                160
15 valuationdomain = {                161
16     'min': Decimal('-1.0'),          162
17     'med': Decimal('0.0'),          163
18     'max': Decimal('1.0'),          164
19     'hasIntegerValuation': True, # representation format 165
20 }                                166
21 relation = {                      167
22     'a1': {'a1':Decimal('-1.0'), 'a2':Decimal('-1.0'), 168
23         'a3':Decimal('1.0'), 'a4':Decimal('-1.0'), 169
24         'a5':Decimal('-1.0'), }, 170
25     'a2': {'a1':Decimal('1.0'), 'a2':Decimal('-1.0'), 171
26         'a3':Decimal('-1.0'), 'a4':Decimal('1.0'), 172
27         'a5':Decimal('1.0'), }, 173

```

```

28     'a3': {'a1':Decimal('1.0'), 'a2':Decimal('-1.0'),
29             'a3':Decimal('-1.0'), 'a4':Decimal('1.0'),
30             'a5':Decimal('-1.0'),},
31     'a4': {'a1':Decimal('1.0'), 'a2':Decimal('1.0'),
32             'a3':Decimal('1.0'), 'a4':Decimal('-1.0'),
33             'a5':Decimal('-1.0'),},
34     'a5': {'a1':Decimal('1.0'), 'a2':Decimal('1.0'),
35             'a3':Decimal('1.0'), 'a4':Decimal('-1.0'),
36             'a5':Decimal('-1.0'),},
37 }
```

In the DIGRAPH3 resources, all digraph object instances are of root Digraph type (see the technical description) and contain at least the following attributes (see Listing 1.1 on page 6 Lines 12–14):

1. A name attribute, holding usually the actual name of the stored instance that was used to create the instance; 187
2. An ordered dictionary of digraph nodes called actions (decision alternatives) with at least a name attribute; 189
3. An order attribute containing the number of graph nodes (length of the actions dictionary) automatically added by the object constructor; 191
4. A logical characteristic valuationdomain dictionary with three decimal entries: the minimum (−1.0, means certainly false), the median (0.0, means missing information), and the maximum characteristic value (+1.0, means certainly true); 193
5. A double dictionary called relation and indexed by an oriented pair of actions (nodes) and carrying a decimal characteristic value in the range of the previous valuation domain; 197
6. Its associated gamma attribute, a dictionary containing the direct successors, respectively, predecessors of each action, automatically added by the object constructor; 200
7. Its associated notGamma attribute, a dictionary containing the actions that are not direct successors, respectively, predecessors of each action, automatically added by the object constructor. 203

1.4 Inspecting a Digraph Object

206

Different show...() methods, like the showShort() method, now reveal us that dg is a crisp, irreflexive, and connected digraph of order five (see Listing 1.3 Lines 1, 16, 26, 29). 207

Listing 1.3 Random crisp digraph object

```

1 >>> dg.showShort()
2 *----- show short -----*
3 Digraph      : tutorialDigraph
4 Actions      : OrderedDict([
```

```

5   ('a1', {'shortName': 'a1', 'name': 'random decision action'}), 214
6   ('a2', {'shortName': 'a2', 'name': 'random decision action'}), 215
7   ('a3', {'shortName': 'a3', 'name': 'random decision action'}), 216
8   ('a4', {'shortName': 'a4', 'name': 'random decision action'}), 217
9   ('a5', {'shortName': 'a5', 'name': 'random decision action'}), 218
10  ])
219
11 Valuation domain : 220
12   'min': Decimal('-1.0'), 221
13   'max': Decimal('1.0'), 222
14   'med': Decimal('0.0'), 'hasIntegerValuation': True 223
15  }
224
16 >>> dg.showRelationTable() 225
17 * ---- Relation Table ---- 226
18   S | 'a1' 'a2' 'a3' 'a4' 'a5' 227
19  -----|----- 228
20   'a1' | -1 -1 1 -1 -1 229
21   'a2' | 1 -1 -1 1 1 230
22   'a3' | 1 -1 -1 1 -1 231
23   'a4' | 1 1 1 -1 -1 232
24   'a5' | 1 1 1 -1 -1 233
25 Valuation domain: [-1;+1] 234
26 >>> dg.showComponents() 235
27 *--- Connected Components ---* 236
28 1: ['a1', 'a2', 'a3', 'a4', 'a5'] 237
29 >>> dg.showNeighborhoods() 238
30 Neighborhoods: 239
31   Gamma : 240
32   'a1': in => {'a2', 'a4', 'a3', 'a5'}, out => {'a3'} 241
33   'a2': in => {'a5', 'a4'}, out => {'a1', 'a4', 'a5'} 242
34   'a3': in => {'a1', 'a4', 'a5'}, out => {'a1', 'a4'} 243
35   'a4': in => {'a2', 'a3'}, out => {'a1', 'a3', 'a2'} 244
36   'a5': in => {'a2'}, out => {'a1', 'a3', 'a2'} 245
37   Not Gamma : 246
38   'a1': in => set(), out => {'a2', 'a4', 'a5'} 247
39   'a2': in => {'a1', 'a3'}, out => {'a3'} 248
40   'a3': in => {'a2'}, out => {'a2', 'a5'} 249
41   'a4': in => {'a1', 'a5'}, out => {'a5'} 250
42   'a5': in => {'a1', 'a4', 'a3'}, out => {'a4'} 251

```

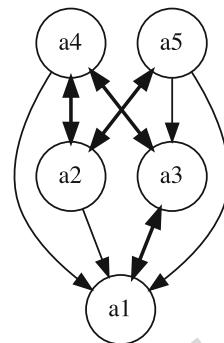
The `exportGraphViz()` method generates in the current working directory 252
 a `tutorialDigraph.dot` file and a `tutorialdigraph.png` picture of the 253
 tutorial digraph `dg` (see Fig. 1.1), if the `graphviz` tools are installed on your system 254
 (Gansner and North 2000). 255

```

1 >>> dg.exportGraphViz('tutorialDigraph') 256
2 *---- exporting a dot file do GraphViz tools -----* 257
3 Exporting to tutorialDigraph.dot 258
4 dot -Grankdir=BT -Tpng tutorialDigraph.dot -o 259
      tutorialDigraph.png 260

```

Fig. 1.1 The tutorial crisp digraph. The `exportGraphViz()` method is depending on drawing tools from <https://graphviz.org>. On Linux or Debian you may try `'sudo apt-get install graphviz'` to install them. Under MacOS there are ready `dmg` installers available



Digraph3 (graphviz), R. Bisdorff, 2020

Further methods are provided for inspecting this random Digraph object `dg`, like the following `showStatistics()` method.

Listing 1.4 Inspecting a Digraph object

```

1  >>> dg.showStatistics()                                     263
2  *----- general statistics -----*                         264
3  for digraph          : <tutorialDigraph.py>               265
4  order                : 5 nodes                           266
5  size                 : 12 arcs                          267
6  undetermined         : 0 arcs                           268
7  determinateness (%) : 100.0                           269
8  arc density          : 0.60                            270
9  double arc density   : 0.40                            271
10 single arc density   : 0.40                           272
11 absence density      : 0.20                           273
12 strict single arc density: 0.40                      274
13 strict absence density : 0.20                         275
14 nbr. of components   : 1                             276
15 nbr. of strong components : 1                        277
16 transitivity degree (%) : 60.0                         278
17                                : [0, 1, 2, 3, 4, 5]                      279
18 outdegrees distribution : [0, 1, 1, 3, 0, 0]           280
19 indegrees distribution : [0, 1, 2, 1, 1, 0]           281
20 mean outdegree        : 2.40                           282
21 mean indegree         : 2.40                           283
22                                : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]        284
23 symmetric degrees dist. : [0, 0, 0, 0, 1, 4, 0, 0, 0, 0] 285
24 mean symmetric degree : 4.80                           286
25 outdegrees concentration index : 0.1667            287
26 indegrees concentration index : 0.2333            288
27 symdegrees concentration index : 0.0333            289
28                                : [0, 1, 2, 3, 4, 'inf']           290
29 neighbourhood depths distribution: [0, 1, 4, 0, 0, 0] 291
30 mean neighbourhood depth   : 1.80                           292
31 digraph diameter         : 2                            293
32 agglomeration distribution : :                         294
  
```

Fig. 1.2 Browsing the relation map of the tutorial digraph. + Indicates a certainly valid and – indicates a certainly invalid relation. Here we find confirmed again that our random digraph instance, `dg`, is indeed a crisp, i.e. 100% determined irreflexive digraph instance

Relation Map

Ranking rule: Alphabetic

r(x,y)	a1	a2	a3	a4	a5
a1	–	–	+	–	–
a2	+	–	–	+	+
a3	+	–	–	+	–
a4	+	+	+	–	–
a5	+	+	+	–	–

Semantics	
+	certainly valid
+	valid
–	indeterminate
–	invalid
–	certainly invalid

```

33 a1 : 58.33
34 a2 : 33.33
35 a3 : 33.33
36 a4 : 50.00
37 a5 : 50.00
38 agglomeration coefficient : 45.00

```

295
296
297
298
299
300

The preceding `show...` methods usually rely upon corresponding compute methods, like: `computeSize()`, `computeDeterminateness()`, or `computeTransitivityDegree()`.

301
302
303

Listing 1.5 Various compute...() methods

```

1 >>> dg.computeSize()
2 12
3 >>> dg.computeDeterminateness(InPercents=True)
4 Decimal('100.00')
5 >>> dg.computeTransitivityDegree(InPercents=True)
6 Decimal('60.00')

```

304
305
306
307
308
309

Mind that `show...` methods output their results in the Python terminal console. We provide also some `showHTML...` methods which output their results in a system browser's tab or window (Fig. 1.2).

310
311
312

```

1 >>> dg.showHTMLRelationMap(relationName='r(x,y)', \
2 ... rankingRule=None)

```

313
314

Some special types of digraph instances, like the `CirculantDigraph` or the `GridDigraph` classes, are readily available (see Fig. 1.3 on the following page).

315
316

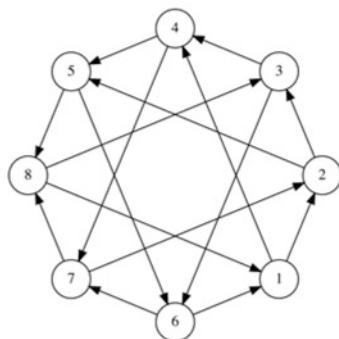
Listing 1.6 Circulant digraphs and $n \times m$ grid digraphs

```

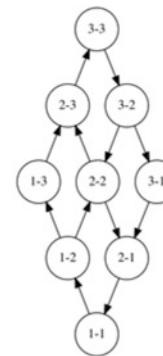
1 >>> from digraphs import CirculantDigraph, GridDigraph
2 >>> c8 = CirculantDigraph(order=8, circulants=[1,3])
3 >>> c8.exportGraphViz('c8')

```

317
318
319



Digraph3 (graphviz), R. Bisдорff, 2020



Digraph3 (graphviz), R. Bisдорff, 2020

Fig. 1.3 The circulant [1,3] digraph and the 3 grid digraph

```

4      *---- exporting a dot file for GraphViz tools ----*
5      Exporting to c8.dot
6      circo -Tpng c8.dot -o c8.png
7  >>> grid3 = GridDigraph(n=3, m=3, \
8          ...                           hasMedianSplitOrientation=True)
9  >>> grid3.exportGraphViz('grid3')
10     *---- exporting a dot file for GraphViz tools ----*
11     Exporting to grid3.dot
12     dot -Grankdir=BT -Tpng grid3.dot -o grid3.png

```

The next Chap. 2 introduces the fundamental *bipolar-valued digraph* model 329
 which is *root object type* to all the digraph models implemented in the DIGRAPH3 330
 modules (Bisдорff 2021). 331

References

- | | |
|---|-----|
| Bisдорff R (2021) Technical documentation of the Digraph3 collection of Python modules. https://digraph3.readthedocs.io/en/latest/techDoc.html | 333 |
| | 334 |
| Gansner E, North S (2000) An open graph visualization system and its applications to software 335
engineering. Softw Pract Exp 30(11):1203–1233. https://graphviz.org/documentation/ 336 | |
| McKay B, Piperno A (2013) Practical graph isomorphism, II. J Symb Comput 60:94–112. https://www.cs.sunysb.edu/~algorith/implement/nauty/implement.shtml 337 | 338 |

AUTHOR QUERIES

- AQ1.** Please check if the edit made to the sentence “Different show . . . () ...” is fine.
- AQ2.** Missing citation for Fig. 1.2 was inserted here. Please check if appropriate.

Uncorrected Proof

Chapter 2

Working with Bipolar-Valued Digraphs

1

2

Contents

2.1	Random Bipolar-Valued Digraphs	3
2.2	Graphviz Drawings	13 4
2.3	Asymmetric and Symmetric Parts	15 5
2.4	Border and Inner Parts	16 6
2.5	Fusion by Epistemic Disjunction	17 7
2.6	Dual, Converse, and Codual Digraphs	18 8
2.7	Symmetric and Transitive Closures	20 9
2.8	Strong Components	21 10
2.9	CSV Storage	22 11
2.10	Complete, Empty, and Indeterminate Digraphs	23 12
		24 13

Abstract The chapter introduces bipolar-valued digraphs, the fundamental root type of all the specialised digraphs implemented in the DIGRAPH3 modules. With the help of a randomly valued digraph, we illustrate some basic digraph manipulation methods, like drawing the digraph, dividing the digraph into its asymmetric and symmetric parts, separating the border from the inner part, computing associated dual, converse and codual digraphs, and operating symmetric and transitive closures.

2.1 Random Bipolar-Valued Digraphs

20

In Listing 2.1, we generate a uniformly random $[-1.0; +1.0]$ -valued digraph of order 7, denoted `rdg` and modelling, for instance, a binary relation $S(x, y)$ defined on the set of nodes of `rdg`. For this purpose, the DIGRAPH3 resources provide in the `randomDigraphs` module a specific `RandomValuationDigraph` class (Bisdorff 2021b).

Listing 2.1 Random bipolar-valued digraph instance

```
1 >>> from randomDigraphs import RandomValuationDigraph
2 >>> rdg = RandomValuationDigraph(order=7)
3 >>> rdg.save('tutRandValDigraph')
```

26

27

28

```

4  >>> from digraphs import Digraph
5  >>> rdg = Digraph('tutRandValDigraph')
6  >>> rdg
7  *----- Digraph instance description -----
8  Instance class      : Digraph
9  Instance name       : tutRandValDigraph
10 Digraph Order       : 7
11 Digraph Size        : 22
12 Valuation domain   : [-1.00;1.00]
13 Determinateness (%) : 75.24
14 Attributes          : ['name','actions','order',
15                           'valuationdomain','relation',
16                           'gamma','notGamma']

```

With the `save()` method (see Listing 2.1 on the previous page Line 3) we keep for future use a backup version of `rdg` which is saved into a file named `tutRandValDigraph.py` in the current working directory. The genuine `Digraph` class constructor can reload the `rdg` object from the stored file (Line 5). One can easily inspect the content of a `Digraph` object with its name `rdg` (Line 6). The digraph size 22 indicates the number of positively valued arcs (Line 11). The valuation domain is uniformly distributed in the interval $[-1.0; 1.0]$, the mean absolute arc valuation being $(0.7524 \times 2) - 1.0 = 0.5048$ (Line 13).

As mentioned in the previous Chap. 1, all objects of `Digraph` type contain at least the list of attributes shown here in Lines 14–16:

- A name (string) and a dictionary of `actions` (digraph nodes),
- An `order` (integer) attribute containing the number of nodes,
- A `valuationdomain` dictionary and a double dictionary `relation` representing the adjacency table of the digraph relation,
- A `gamma` and a `notGamma` dictionary containing the direct neighbourhood and not neighbourhood of each node.

The `Digraph` class provides some generic `show...` methods for exploring the content of a given `Digraph` object, like the `showRelationTable()`, the `showComponents()`, and the `showNeighborhoods()` methods.

Listing 2.2 Example of random valuation digraph

```

1  >>> rdg.showRelationTable()
2  * ---- Relation Table -----
3  r(xSy) | '1'   '2'   '3'   '4'   '5'   '6'   '7'
4  -----|-----
5  '1'   | 0.00  -0.48  0.70  0.86  0.30  0.38  0.44
6  '2'   | -0.22  0.00  -0.38  0.50  0.80  -0.54  0.02
7  '3'   | -0.42  0.08  0.00  0.70  -0.56  0.84  -1.00
8  '4'   | 0.44  -0.40  -0.62  0.00  0.04  0.66  0.76
9  '5'   | 0.32  -0.48  -0.46  0.64  0.00  -0.22  -0.52
10 '6'  | -0.84  0.00  -0.40  -0.96  -0.18  0.00  -0.22
11 '7'  | 0.88  0.72  0.82  0.52  -0.84  0.04  0.00
12 >>> rdg.showComponents()

```

```

13  *--- Connected Components ---* 74
14  1: ['1', '2', '3', '4', '5', '6', '7'] 75
15  >>> rdg.showNeighborhoods() 76
16  *--- Neighborhoods -----* 77
17  Gamma: 78
18  '1': in => {'5', '7', '4'}, out => {'5', '7', '6', '3', '4'} 79
19  '2': in => {'7', '3'}, out => {'5', '7', '4'} 80
20  '3': in => {'7', '1'}, out => {'6', '2', '4'} 81
21  '4': in => {'5', '7', '1', '2', '3'}, out => {'5', '7', '1', '6'} 82
22  '5': in => {'1', '2', '4'}, out => {'1', '4'} 83
23  '6': in => {'7', '1', '3', '4'}, out => set() 84
24  '7': in => {'1', '2', '4'}, out => {'1', '2', '3', '4', '6'} 85
25  Not Gamma: 86
26  '1': in => {'6', '2', '3'}, out => {'2'} 87
27  '2': in => {'5', '1', '4'}, out => {'1', '6', '3'} 88
28  '3': in => {'5', '6', '2', '4'}, out => {'5', '7', '1'} 89
29  '4': in => {'6'}, out => {'2', '3'} 90
30  '5': in => {'7', '6', '3'}, out => {'7', '6', '2', '3'} 91
31  '6': in => {'5', '2'}, out => {'5', '7', '1', '3', '4'} 92
32  '7': in => {'5', '6', '3'}, out => {'5'} 93

```

Positive characteristic values indicate the presence of an arc whereas negative ones indicate their absence. 0.0 values indicate an indeterminate situation: there may be or not be an arc (see Listing 2.2 on the facing page Lines 5–11). Mind that some Digraph class methods will ignore the *reflexive* links by considering that they are *indeterminate*, i.e. the characteristic value $r(x \, S \, x)$ for all action x is set to the *median*, i.e. *indeterminate* value 0.0 in this case (Bisdorff 2004).

2.2 Graphviz Drawings

101

An even better insight into the Digraph object `rdg` is given by looking in Fig. 2.1 at its `graphviz` drawing (Gansner and North 2000).¹

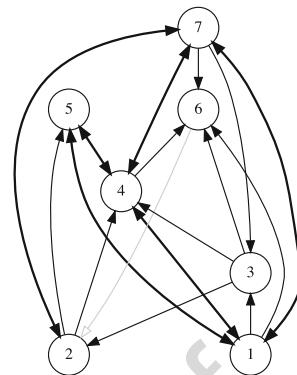
```

1 >>> rdg.exportGraphViz('tutRandValDigraph') 104
2  *--- exporting a dot file for GraphViz tools -----* 105
3  Exporting to tutRandValDigraph.dot 106
4  dot -Grankdir=BT -Tpng tutRandValDigraph.dot\ 107
   -o tutRandValDigraph.png 108

```

¹ The `exportGraphViz()` method is depending on drawing tools from the graphviz software (<https://graphviz.org/>). On Linux Ubuntu or Debian you may try to install them with `sudo apt-get install graphviz`. There are ready `dmg` installers for MacOS.

Fig. 2.1 The tutorial random valuation digraph. Double links are drawn in bold black with an arrowhead at each end, whereas single asymmetric links are drawn in black with an arrowhead showing the direction of the link. Notice the undetermined relational situation ($r(6 \rightarrow 2) = 0.00$) observed between nodes 6 and 2. The corresponding link is marked in grey with an open arrowhead in the drawing



Digraph3 (graphviz), R. Bisdorff, 2020

2.3 Asymmetric and Symmetric Parts

109

Extracting both the *symmetric* as well as the *asymmetric* part of digraph `rdg` may 110
be done with the help of two corresponding classes (see Listing 2.3). 111

Listing 2.3 Computing asymmetric and symmetric parts

```

1 >>> from digraphs import AsymmetricPartialDigraph, \
2 ...                               SymmetricPartialDigraph
3 >>> asymDg = AsymmetricPartialDigraph(rdg)
4 >>> asymDg.exportGraphViz()
5 >>> symDg = SymmetricPartialDigraph(rdg)
6 >>> symDg.exportGraphViz()

```

112 113 114 115 116 117

The relation constructors of the partial objects `asymDg` and `symDg` set to 118
the indeterminate characteristic value 0.0 all non-asymmetric, respectively, non- 119
symmetric links between nodes (see Fig. 2.2 on the next page). 120

In Listing 2.4 below is shown, for instance, the source code of the `relation` 121
constructor of the `AsymmetricPartialDigraph` class: 122

Listing 2.4 Computing the asymmetric part of a bipolar-valued relation

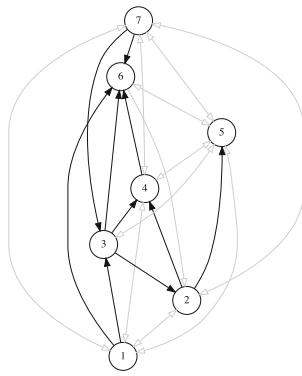
```

1 def _constructRelation(self):
2     actions = self.actions
3     Min = self.valuationdomain['min']
4     Max = self.valuationdomain['max']
5     Med = self.valuationdomain['med']
6     relationIn = self.relation
7     relationOut = {}
8     for a in actions:
9         relationOut[a] = {}
10        for b in actions:
11            if a != b:
12                if relationIn[a][b] >= Med and relationIn[b][a] <= Med:
13                    relationOut[a][b] = relationIn[a][b]
14                elif relationIn[a][b] <= Med and relationIn[b][a] >= Med:
15                    relationOut[a][b] = relationIn[a][b]

```

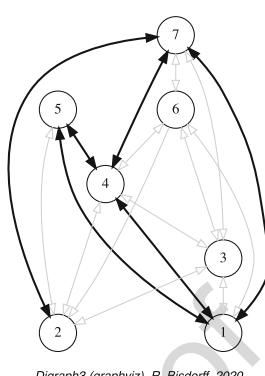
123 124 125 126 127 128 129 130 131 132 133 134 135 136 137

Asymmetric Part



Digraph3 (graphviz), R. Bisдорff, 2020

Symmetric Part



Digraph3 (graphviz), R. Bisдорff, 2020

Fig. 2.2 Asymmetric and symmetric part of the tutorial random valuation digraph

```

16         else:
17             relationOut[a][b] = Med
18         else: # reflexive links are ignored
19             relationOut[a][b] = Med
20     return relationOut

```

138
139
140
141
142

In Line 17 above all non-asymmetric situations are set to the *indeterminate*—
median—characteristic value. Mind by the way that all reflexive relations are
similarly set to this indeterminate value (see Line 19).

143
144
145

AQ1

2.4 Border and Inner Parts

146

We can also extract the *border*—the part of a digraph induced by the union of
its initial and terminal prekernels (see Chap. 17)—as well as, the *inner part*—
the complement of the border—with the help of two corresponding classes:
GraphBorder and GraphInner (see Listing 2.5).

147
148
149
150

Figure 2.3 on the following page shows the border and inner parts of the
linear ordering obtained from the tutorial random valuation digraph rdg with the
NETFLOWS ranking rule (see Sect. 8.3).

151
152
153

Listing 2.5 Border and inner part of a linear order

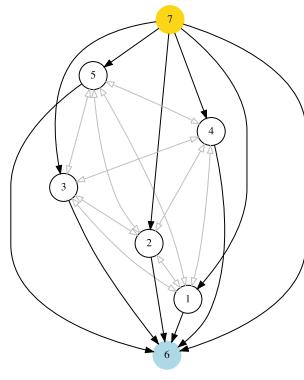
```

1 >>> from digraphs import GraphBorder, GraphInner
2 >>> from linearOrders import NetFlowsOrder
3 >>> nf = NetFlowsOrder(rdg)
4 >>> nf.netFlowsOrder
5     ['6', '4', '5', '3', '2', '1', '7']
6 >>> bnf = GraphBorder(nf)

```

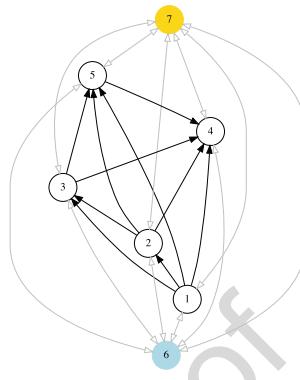
154
155
156
157
158
159

Border Part



Digraph3 (graphviz), R. Bischoff, 2020

Inner Part



Digraph3 (graphviz), R. Bischoff, 2020

Fig. 2.3 Border and inner part of a linear order oriented by terminal and initial kernels

```

7  >>> bnf.exportGraphViz(lastChoice=['6'],firstChoice=['7']) 160
8  >>> inf = GraphInner(nf) 161
9  >>> inf.exportGraphViz(lastChoice=['6'],firstChoice=['7']) 162

```

The drawings in Fig. 2.3 are oriented with the terminal node 6 (lastChoice parameter) and initial node 7 (firstChoice parameter) (see Listing 2.5 on the previous page Lines 7 and 9).

The class constructors of the partial digraphs `bnf` and `inf` (see Lines 6 and 8) set to the *indeterminate* characteristic value all links not in the *border*, respectively, *not* in the *inner* part (see Fig. 2.3). Being much *denser* than a linear order, the actual inner part of our tutorial random valuation digraph `rdg` is reduced to a single arc between nodes 3 and 4 (see Fig. 2.4 on the facing page right side). Indeed, a complete digraph on the limit has no inner part (privacy!) at all, whereas empty and indeterminate digraphs admit both, an empty border and an empty inner part.

2.5 Fusion by Epistemic Disjunction

The `Digraph` object `rdg` can be restored from both partial objects `asymDg` and `symDg`, or as well from the border `bg` and the inner part `ig`, with a *bipolar disjunctive fusion* operator `o-max`, also called *epistemic disjunction*, available via the `FusionDigraph` class.

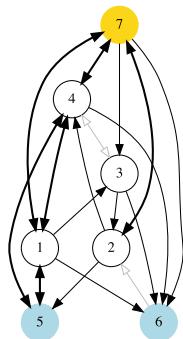
Listing 2.6 Epistemic fusion of partial digraphs

```

1  >>> from digraphs import FusionDigraph 178
2  >>> fusDg = FusionDigraph(asymDg,symDg,operator='o-max') 179
3  >>> # fusDg = FusionDigraph(bg,ig,operator='o-max') 180
4  >>> fusDg.showRelationTable() 181

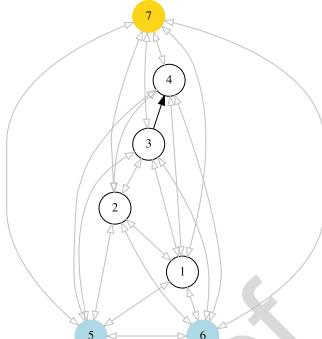
```

Border Part



Digraph3 (graphviz), R. Bisdorff, 2020

Inner Part



Digraph3 (graphviz), R. Bisdorff, 2020

Fig. 2.4 Border and inner part of the tutorial random valuation digraph rdg

* ----- Relation Table -----								182
r(xSy)	'1'	'2'	'3'	'4'	'5'	'6'	'7'	183

'1'	0.00	-0.48	0.70	0.86	0.30	0.38	0.44	185
'2'	-0.22	0.00	-0.38	0.50	0.80	-0.54	0.02	186
'3'	-0.42	0.08	0.00	0.70	-0.56	0.84	-1.00	187
'4'	0.44	-0.40	-0.62	0.00	0.04	0.66	0.76	188
'5'	0.32	-0.48	-0.46	0.64	0.00	-0.22	-0.52	189
'6'	-0.84	0.00	-0.40	-0.96	-0.18	0.00	-0.22	190
'7'	0.88	0.72	0.82	0.52	-0.84	0.04	0.00	191

The epistemic fusion operator $\circ\text{-max}$ (see Listing 2.6 Line 2) is defined as 192
follows: 193

Definition 2.1 (Disjunctive Epistemic Fusion Operator $\circ\text{-max}$) Let r and r' 194
characterise two bipolar-valued epistemic situations: 195

- $\circ\text{-max}(r, r') = \max(r, r')$ when both $r \geq 0.0$ and $r' \geq 0.0$, i.e. r and r' are 196
positively valid or indeterminate; 197
- $\circ\text{-max}(r, r') = \min(r, r')$ when both $r \leq 0.0$ and $r' \leq 0.0$, i.e. r and r' are 198
positively invalid or indeterminate; 199
- otherwise $\circ\text{-max}(r, r') = 0.0$, i.e. indeterminate. 200

Mind that the $\circ\text{-max}$ operator, like a mean operator, is *not associative* when 201
more than 2 operands are given. In order to make the $\circ\text{-max}$ fusion univocal, 202
the following rule is applied—first, all positive and negative terms are separately 203
aggregated—then the $\circ\text{-max}$ fusion is applied to both aggregates. 204

2.6 Dual, Converse, and Codual Digraphs

205

The `digraphs` module provides class constructors for computing the *dual*² 206 (negated relation), the *converse* (transposed relation), and the *codual* (transposed 207 and negated relation) of the `Digraph` instance `rdg` (see Listing 2.7 Lines 4, 16, 208 29). 209

Listing 2.7 Computing associated dual, converse and codual digraphs

```

1  >>> from digraphs import\                                210
2  ...                                DualDigraph, ConverseDigraph, CoDualDigraph 211
3  >>> # dual of rdg                                212
4  >>> ddg = DualDigraph(rdg)                                213
5  >>> ddg.showRelationTable()                                214
6      -r(xSy) | '1'   '2'   '3'   '4'   '5'   '6'   '7' 215
7      ----- 216
8      '1' | 0.00  0.48 -0.70 -0.86 -0.30 -0.38 -0.44 217
9      '2' | 0.22  0.00  0.38 -0.50  0.80  0.54 -0.02 218
10     '3' | 0.42  0.08  0.00 -0.70  0.56 -0.84  1.00 219
11     '4' | -0.44 0.40  0.62  0.00 -0.04 -0.66 -0.76 220
12     '5' | -0.32 0.48  0.46 -0.64  0.00  0.22  0.52 221
13     '6' | 0.84  0.00  0.40  0.96  0.18  0.00  0.22 222
14     '7' | 0.88 -0.72 -0.82 -0.52  0.84 -0.04  0.00 223
15 >>> # converse of rdg                                224
16 >>> cdg = ConverseDigraph(rdg)                                225
17 >>> cdg.showRelationTable()                                226
18     * ---- Relation Table ---- 227
19     r(ySx) | '1'   '2'   '3'   '4'   '5'   '6'   '7' 228
20     ----- 229
21     '1' | 0.00 -0.22 -0.42  0.44  0.32 -0.84  0.88 230
22     '2' | -0.48 0.00  0.08 -0.40 -0.48  0.00  0.72 231
23     '3' | 0.70 -0.38  0.00 -0.62 -0.46 -0.40  0.82 232
24     '4' | 0.86  0.50  0.70  0.00  0.64 -0.96  0.52 233
25     '5' | 0.30  0.80 -0.56  0.04  0.00 -0.18 -0.84 234
26     '6' | 0.38 -0.54  0.84  0.66 -0.22  0.00  0.04 235
27     '7' | 0.44  0.02 -1.00  0.76 -0.52 -0.22  0.00 236
28 >>> # codual of rdg                                237
29 >>> cddg = CoDualDigraph(rdg)                                238
30 >>> cddg.showRelationTable()                                239
31     * ---- Relation Table ---- 240
32     -r(ySx) | '1'   '2'   '3'   '4'   '5'   '6'   '7' 241
33     ----- 242
34     '1' | 0.00  0.22  0.42 -0.44 -0.32  0.84 -0.88 243
35     '2' | 0.48  0.00 -0.08  0.40  0.48  0.00 -0.72 244
36     '3' | -0.70 0.38  0.00  0.62  0.46  0.40 -0.82 245
37     '4' | -0.86 -0.50 -0.70  0.00 -0.64  0.96 -0.52 246
38     '5' | -0.30 -0.80  0.56 -0.04  0.00  0.18  0.84 247

```

² Not to be confused with the dual graph of a plane graph g that has a vertex for each face of g . Here we mean the *less than* (strict converse) relation corresponding to a *greater or equal* relation, or the *less than or equal* relation corresponding to a (strict) *better than* relation.

39	'6'	-0.38 0.54 -0.84 -0.66 0.22 0.00 -0.04	248
40	'7'	-0.44 -0.02 1.00 -0.76 0.52 0.22 0.00	249

Computing the *dual*, respectively, the *converse* of a digraph, may also be done 250 with prefixing the `__neg__` (`-`) or the `__invert__` (`~`) operator. The *codual* of 251 a `DiGraph` object can, hence, as well be computed with a *composition* (in either 252 order) of both operations (see Line 3 below). 253

Listing 2.8 Computing the dual, the converse, and the codual of a digraph

1	>>> ddg = -rdg	# dual of rdg	254
2	>>> cdg = ~rdg	# converse of rdg	255
3	>>> cddg = ~(-rdg) # = -(~(rdg)) codual of rdg		256
4	>>> (-(~rdg)).showRelationTable()		257
5	* ----- Relation Table -----		258
6	-r(ySx) '1' '2' '3' '4' '5' '6' '7'		259
7	----- -----		260
8	'1' 0.00 0.22 0.42 -0.44 -0.32 0.84 -0.88		261
9	'2' 0.48 0.00 -0.08 0.40 0.48 0.00 -0.72		262
10	'3' -0.70 0.38 0.00 0.62 0.46 0.40 -0.82		263
11	'4' -0.86 -0.50 -0.70 0.00 -0.64 0.96 -0.52		264
12	'5' -0.30 -0.80 0.56 -0.04 0.00 0.18 0.84		265
13	'6' -0.38 0.54 -0.84 -0.66 0.22 0.00 -0.04		266
14	'7' -0.44 -0.02 1.00 -0.76 0.52 0.22 0.00		267

2.7 Symmetric and Transitive Closures

268

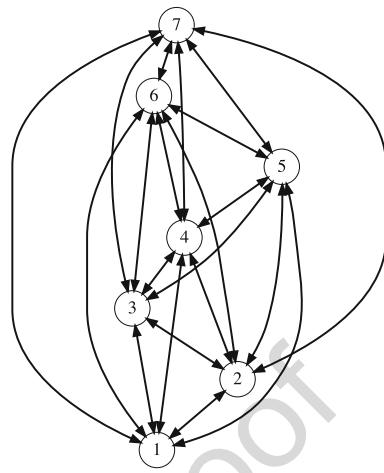
Symmetric and transitive closures, by default in-site methods, are also available 269 (see Fig. 2.5). Note that it is a good idea, before going ahead with these 270 in-site operations, that irreversibly modifies the original `rdg` object, to previously 271 make a backup version of `rdg`. The simplest storage method, always 272 provided by the generic `DiGraph.save()` method, writes out in a named 273 file the Python content of the `DiGraph` object in string representation (see 274 Sect. 1.3). 275

Listing 2.9 Symmetric and transitive closures

1	>>> rdg.save('tutRandValDigraph')	276
2	>>> rdg.closeSymmetric(InSite=True)	277
3	>>> rdg.closeTransitive(InSite=True)	278
4	>>> rdg.exportGraphViz('symTransClosure')	279

The `closeSymmetric()` method (see Listing 2.9 Line 2), of complexity 280 $O(n^2)$ where n denotes the digraph's order, changes, on the one hand, all single 281 pairwise links it may detect into double links by operating a disjunction of the 282 pairwise relations. On the other hand, the `closeTransitive()` method (see

Fig. 2.5 Symmetric and transitive closure of the tutorial random valuation digraph `rdg`



Digraph3 (graphviz), R. Bisdorff, 2020

Line 3) implements the *Roy-Warshall* transitive closure algorithm of complexity 283
 $O(n^3)$ (Roy 1959; Warshall 1962). 284

The same `closeTransitive()` with a `Reverse = True` flag may be 285
readily used for eliminating all transitive arcs from a transitive digraph instance. We 286
make usage of this feature when drawing HASSE diagrams of TransitiveDi- 287
graph objects. 288

2.8 Strong Components

As the original digraph `rdg` was connected (see above the result of the 290
`showShort()` command), both the symmetric and the transitive closures 291
operated together, will necessarily produce a single strong component, i.e. a 292
complete digraph. It is sometimes useful to collapse all strong components 293
in a given digraph and construct the so *collapsed* digraph. Using the 294
StrongComponentsCollapsedDigraph constructor here will render a 295
single hyper-node gathering all the original nodes (see Line 7 below). 296

Listing 2.10 Computing the strong components in a digraph

```

1  >>> from digraphs import StrongComponentsCollapsedDigraph 297
2  >>> sc = StrongComponentsCollapsedDigraph(rdg) 298
3  >>> sc.showAll() 299
4  *---- show detail ----* 300
5  Digraph : tutRandValDigraph_Scc 301
6  *---- Actions ----* 302
7  ['_7_1_2_6_5_3_4_'] 303
8  *---- Relation Table ----* 304
9      S | 'Scc_1' 305

```

```

10      -----|-----
11      'Scc_1' | 0.00
12  *---- strong Components ----*
13      short      content
14      'Scc_1'      '_7_1_2_6_5_3_4_'
15  *---- Neighborhoods ----*
16      Gamma      :
17      'frozenset({'7','1','2','6','5','3','4'})':
18          in => set(), out => set()
19      Not Gamma :
20      'frozenset({'7','1','2','6','5','3','4'})':
21          in => set(), out => set()

```

2.9 CSV Storage

318

Sometimes it is required to exchange the graph valuation data in CSV format with 319
 statistical software like **R**³ or **gretl**.⁴ For this purpose it is possible to export 320
 the digraph adjacency table into a CSV file. The characteristic valuation domain is 321
 hereby normalised by default to the range $[-1.0, 1.0]$ and the diagonal is put by 322
 default to the minimal value -1.0 . 323

```

1 >>> rdg = Digraph('tutRandValDigraph')
2 >>> rdg.saveCSV('tutRandValDigraph')
3 # content of file tutRandValDigraph.csv
4 "d","1","2","3","4","5","6","7"
5 "1",-1.0,0.48,-0.7,-0.86,-0.3,-0.38,-0.44
6 "2",0.22,-1.0,0.38,-0.5,-0.8,0.54,-0.02
7 "3",0.42,-0.08,-1.0,-0.7,0.56,-0.84,1.0
8 "4",-0.44,0.4,0.62,-1.0,-0.04,-0.66,-0.76
9 "5",-0.32,0.48,0.46,-0.64,-1.0,0.22,0.52
10 "6",0.84,0.0,0.4,0.96,0.18,-1.0,0.22
11 "7",-0.88,-0.72,-0.82,-0.52,0.84,-0.04,-1.0

```

It is possible to reload a **Digraph** instance from its previously saved CSV file 335
 content. 336

```

1 >>> from digraphs import CSVDigraph
2 >>> rdgcsv = CSVDigraph('tutRandValDigraph')
3 >>> rdgcsv.showRelationTable(ReflexiveTerms=False)
4 *---- Relation Table ----
5 r(xSy) | '1'   '2'   '3'   '4'   '5'   '6'   '7'
6 -----|-----
7 '1'   |   -   -0.48   0.70   0.86   0.30   0.38   0.44
8 '2'   |   -0.22   -   -0.38   0.50   0.80   -0.54   0.02
9 '3'   |   -0.42   0.08   -   0.70   -0.56   0.84   -1.00
10 '4'  |   0.44   -0.40   -0.62   -   0.04   0.66   0.76

```

³ <https://www.r-project.org/>.

⁴ The Gnu Regression, Econometrics and Time-series Library <http://gretl.sourceforge.net/>.

Fig. 2.6 The valued relation table shown in a browser window. Positive arcs are shown in green and negative arcs in red. Indeterminate—zero-valued—arcs, like the reflexive diagonal ones or the arc between nodes 6 and 2, are shown in grey

Tutorial random digraph

r(x S y)	1	2	3	4	5	6	7
1	0.00	-0.48	0.70	0.86	0.30	0.38	0.44
2	-0.22	0.00	-0.38	0.50	0.80	-0.54	0.02
3	-0.42	0.08	0.00	0.70	-0.56	0.84	-1.00
4	0.44	-0.40	-0.62	0.00	0.04	0.66	0.76
5	0.32	-0.48	-0.46	0.64	0.00	-0.22	-0.52
6	-0.84	0.00	-0.40	-0.96	-0.18	0.00	-0.22
7	0.88	0.72	0.82	0.52	-0.84	0.04	0.00

```

11   '5'      |  0.32 -0.48 -0.46  0.64   - -0.22 -0.52  347
12   '6'      | -0.84  0.00 -0.40 -0.96 -0.18   - -0.22  348
13   '7'      |  0.88  0.72  0.82  0.52 -0.84  0.04   -  349

```

It is as well possible with the `showHTMLRelationTable()` method to show a coloured version of the valued relation table in a system browser window (see Fig. 2.6).

```

1 >>> rdgcsv.showHTMLRelationTable(tableTitle="Tutorial random 353
      digraph") 354

```

2.10 Complete, Empty, and Indeterminate Digraphs

Let us finally mention some special universal classes of digraphs that are readily available in the `digraphs` module,⁵ like:

- the `CompleteDigraph` class,
- the `EmptyDigraph` class, and
- the `IndeterminateDigraph` class,

which set all pairwise characteristic values, respectively, to the *maximum*, the *minimum*, respectively, to the *median*—indeterminate—value.

Listing 2.11 Complete, empty, and indeterminate digraphs

```

1 >>> from digraphs import CompleteDigraph,EmptyDigraph,\ 363
2 ...           IndeterminateDigraph 364
3 >>> # the empty digraph 365
4 >>> e = EmptyDigraph(order=5) 366
5 >>> e.showRelationTable() 367
6   * ---- Relation Table ---- 368
7   S   |   '1'   '2'   '3'   '4'   '5' 369
8   --- -|----- 370
9   '1' | -1.00 -1.00 -1.00 -1.00 -1.00 371

```

⁵ See Bisdorff (2021a).

```

10  '2' | -1.00 -1.00 -1.00 -1.00 -1.00 372
11  '3' | -1.00 -1.00 -1.00 -1.00 -1.00 373
12  '4' | -1.00 -1.00 -1.00 -1.00 -1.00 374
13  '5' | -1.00 -1.00 -1.00 -1.00 -1.00 375
14 >>> e.showNeighborhoods() 376
15 Neighborhoods: 377
16   Gamma : 378
17   '1': in => set(), out => set() 379
18   '2': in => set(), out => set() 380
19   '5': in => set(), out => set() 381
20   '3': in => set(), out => set() 382
21   '4': in => set(), out => set() 383
22   Not Gamma : 384
23   '1': in => {'2','4','5','3'}, out => {'2','4','5','3'} 385
24   '2': in => {'1','4','5','3'}, out => {'1','4','5','3'} 386
25   '5': in => {'1','2','4','3'}, out => {'1','2','4','3'} 387
26   '3': in => {'1','2','4','5'}, out => {'1','2','4','5'} 388
27   '4': in => {'1','2','5','3'}, out => {'1','2','5','3'} 389
28 >>> # the indeterminate digraph 390
29 >>> i = IndeterminateDigraph() 391
30   * ----- Relation Table ----- 392
31   S | '1' '2' '3' '4' '5' 393
32   -----|----- 394
33   '1' | 0.00 0.00 0.00 0.00 0.00 395
34   '2' | 0.00 0.00 0.00 0.00 0.00 396
35   '3' | 0.00 0.00 0.00 0.00 0.00 397
36   '4' | 0.00 0.00 0.00 0.00 0.00 398
37   '5' | 0.00 0.00 0.00 0.00 0.00 399
38 >>> i.showNeighborhoods() 400
39 Neighborhoods: 401
40   Gamma : 402
41   '1': in => set(), out => set() 403
42   '2': in => set(), out => set() 404
43   '5': in => set(), out => set() 405
44   '3': in => set(), out => set() 406
45   '4': in => set(), out => set() 407
46   Not Gamma : 408
47   '1': in => set(), out => set() 409
48   '2': in => set(), out => set() 410
49   '5': in => set(), out => set() 411
50   '3': in => set(), out => set() 412
51   '4': in => set(), out => set() 413

```

Mind the subtle difference between the neighbourhoods of an *empty* and the 414 neighbourhoods of an *indeterminate* Digraph instance. In the first kind, the 415 neighbourhoods are known to be completely *empty* (see Listing [2.11 on the](#) 416 preceding page Lines 23–27) whereas, in the latter, *nothing is known* about the 417 actual neighbourhoods of the nodes (see Lines 47–51). These two cases illustrate 418 why in *bipolar-valued* digraphs, we may sometimes need both a `gamma` **and** a 419 `notGamma` attribute. 420

Notes

421

It was *Denis Bouyssou* who first suggested us end of the nineties, when we started 422 to work in Prolog on the computation of fuzzy digraph kernels with finite domain 423 constraint solvers, that the 50% criteria significance majority was a very special 424 value to be carefully taken into account. The converging solution vectors of the 425 fixpoint kernel equations furthermore confirmed this special status of the 50% 426 majority (see Chap. 17). These early insights led to the seminal articles on bipolar- 427 valued epistemic logic where we introduced split truth/falseness semantics for 428 a multi-valued logical processing of fuzzy preference modelling (Bisdorff 2000, 429 2002). The characteristic valuation domain remained, however, the classical fuzzy 430 [0.0; 1.0] valuation domain. 431

It is only in 2004, when we succeeded in assessing the stability of the outranking 432 digraph when solely ordinal criteria significance weights are given that it became 433 clear and evident for us that the characteristic valuation domain had to be shifted 434 to a $[-1.0; +1.0]$ -valued domain (see Chap. 19 and Bisdorff 2004). In this bipolar 435 valuation, the 50% majority threshold corresponds now to the median 0.0 value, 436 characterising with the correct zero value an epistemic indeterminateness—no 437 knowledge—situation. Furthermore, identifying truth and falseness directly by the 438 sign of the characteristic value revealed itself to be very efficient not only from 439 a computational point of view but also from scientific and semiotic perspectives. 440 A positive (respectively, negative) characteristic value now attest a logically valid 441 (respectively, invalid) statement and a negative affirmation now means a positive 442 refutation and vice versa. Furthermore, the median zero value gives way to 443 efficiently handling partial objects—like the border or the asymmetric part of a 444 digraph—and, even more important from a practical decision making point of view, 445 any missing data. 446

The bipolar $[-1.0; +1.0]$ -valued characteristic domain opened so the way 447 to important new operations and concepts, like the disjunctive epistemic fusion 448 operation seen in Sect. 2.5 on page 18 that confers the outranking digraph a logically 449 sound and epistemically correct definition (Bisdorff 2013). KENDALL's ordinal 450 correlation index could be extended to a bipolar-valued relational equivalence index 451 between digraphs (see Chap. 16 and Bisdorff 2012). Making usage of the bipolar- 452 valued Gaussian error function naturally led to defining a bipolar-valued likelihood 453 function, where a positive, respectively, negative, value gives the likelihood of an 454 affirmation, respectively, a refutation (see Chap. 18 and Bisdorff 2014). 455

In the following Chap. 3 we introduce the main formal object of this book, 456 namely *bipolar-valued outranking* digraphs. 457

References

458

Bisdorff R (2000) Logical foundation of fuzzy preferential systems with application to the electre 459 decision aid methods. Comput Oper Res 27:673–687. <http://hdl.handle.net/10993/23724> 460

- Bisdorff R (2002) Logical foundation of multicriteria preference aggregation. In: Bouyssou D, Jacquet-Lagrèze E, Perny P, Słowiński R, Vanderpooten D, Vincke P (eds) Aiding decisions with multiple criteria. Kluwer Academic, pp 379–403. <http://hdl.handle.net/10993/45313> 461
462
463
- Bisdorff R (2004) On a natural fuzzification of Boolean logic. In: Klement EP, Pap E (eds) Linz seminar on fuzzy set theory, mathematics of fuzzy systems, Bildungszentrum St. Magdalena, Linz (Austria), pp 20–26. <http://hdl.handle.net/10993/23721> 464
465
466
- Bisdorff R (2012) On measuring and testing the ordinal correlation between bipolar outranking relations. In: Mousseau V, Pirlot M (eds) DAP'2012 from multiple criteria decision aid to preference learning, University of Mons (Belgium), pp 91–100. <http://hdl.handle.net/10993/23909> 467
468
469
470
- Bisdorff R (2013) On polarizing outranking relations with large performance differences. *J Multi-Criteria Decis Anal* Wiley 20:3–12. <http://hdl.handle.net/10993/245> 471
472
- Bisdorff R (2014) On confident outrankings with multiple criteria of uncertain significance. In: Mousseau V, Pirlot M (eds) DAP'2014 from multiple criteria decision aid to preference learning, Ecole Centrale de Paris, pp 1–6. <http://hdl.handle.net/10993/23910> 473
474
475
- Bisdorff R (2021a) Documentation of the Digraph3 collection of Python modules for algorithmic decision theory. <https://digraph3.readthedocs.io/en/latest/> 476
477
- Bisdorff R (2021b) Technical documentation of the Digraph3 collection of Python modules. <https://digraph3.readthedocs.io/en/latest/techDoc.html> 478
479
- Gansner E, North S (2000) An open graph visualization system and its applications to software engineering. *Softw Pract Exp* 30(11):1203–1233. <https://graphviz.org/documentation/> 480
481
- Roy B (1959) Transitivity et connexité. *C R Acad Sci Paris* 249:216–218 482
- Marshall S (1962) A theorem on Boolean matrices. *J ACM* 9:11–12 483

AUTHOR QUERY

- AQ1.** Please check if the edit made to the sentence “Mind by the way that all...” is fine.

Uncorrected Proof

Chapter 3

Working with Outranking Digraphs

1

2

The rule for the combination of independent concurrent arguments takes a very simple form when expressed in terms of the intensity of belief ... It is this: Take the sum of all the feelings of belief which would be produced separately by all the arguments pro, subtract from that the similar sum for arguments con, and the remainder is the feeling of belief which ought to have the whole. This is a proceeding which men often resort to, under the name of balancing reasons

—C.S. Peirce, *The probability of induction* (1878)

11

Contents

12

3.1	The Hybrid Outranking Digraph Model	29	13
3.2	The Bipolar-Valued Outranking Digraph	32	14
3.3	Pairwise Comparisons	33	15
3.4	Recoding the Characteristic Valuation Domain	35	16
3.5	The Strict Outranking Digraph	35	17

Abstract In this chapter, we introduce the main formal object of this book, namely the bipolar-valued outranking digraph. With a randomly generated multiple-criteria performance tableau, we construct the corresponding bipolar-valued outranking relation from pairwise comparisons. The resulting bipolar-valued outranking characteristics may be recoded. Finally, the codual outranking digraph gives us the associated strict outranking relation.

18

19

20

21

22

23

3.1 The Hybrid Outranking Digraph Model

24

In the `outrankingDigraphs` module, the `BipolarOutrankingDigraph` class provides our standard *outranking digraph* constructor. Such an instance represents a *hybrid* object of both the `PerformanceTableau` type and the `Outran-`

25

26

27

kingDigraph type (Bisdorff 2021). A given BipolarOutrankingDigraph object contains hence always at least the following attributes:

1. `actions`: an ordered dictionary describing the potential decision actions or alternatives with `name` and `comment` attributes
2. `objectives`: a possibly empty ordered dictionary of decision objectives with `name` and `comment` attributes, describing the multiple preference dimensions involved in the decision problem
3. `criteria`: an ordered dictionary of performance criteria, i.e. *preferentially independent* and *non-redundant* decimal-valued evaluation functions used for assessing the performance of each potential decision action with respect to a decision objective
4. `evaluation`: a double dictionary gathering performance evaluations for each decision alternative on each criterion function
5. `valuationdomain`: a dictionary with three entries: the minimum (-1.0 , *certainly outranked*), the median (0.0 , *indeterminate*), and the maximum characteristic value ($+1.0$, *certainly outranking*)
6. `relation`: a double dictionary defined on the Cartesian product of the set of decision alternatives capturing the credibility of the pairwise *outranking situation* computed on the basis of the performance differences observed between couples of decision alternatives on the given family of criteria functions

Let us consider, for instance, a random bipolar-valued outranking digraph with seven decision actions denoted a_1, a_2, \dots, a_7 . We need therefore, first, to generate in Listing 3.1 a corresponding random performance tableau.

Listing 3.1 Generating a random performance tableau

```

1 >>> from perfTabs import RandomPerformanceTableau      52
2 >>> pt = RandomPerformanceTableau(numberOfActions=7, \      53
3 ...                                         seed=100)      54
4 >>> pt      55
5 *----- PerformanceTableau instance description -----* 56
6   Instance class      : RandomPerformanceTableau      57
7   Seed                : 100      58
8   Instance name       : randomperftab      59
9   Actions              : 7      60
10  Criteria             : 7      61
11  NaN proportion (%) : 6.1      62
12 >>> pt.showActions()      63
13 *----- show digraphs actions -----*      64
14   key: a1      65
15     name:      action 1      66
16     comment:   RandomPerformanceTableau() generated. 67
17   key: a2      68
18     name:      action 2      69
19     comment:   RandomPerformanceTableau() generated. 70
20   ...
21   ...

```

```

22  key: a7
23  name:      action 7
24  comment:   RandomPerformanceTableau() generated.

```

In this example, we consider a family of seven *equi-significant* cardinal *criteria functions* g_1, g_2, \dots, g_7 , measuring the performance of each alternative on a rational scale from 0.0 (worst) to 100.00 (best). In order to capture the evaluation procedures' potential *uncertainty* and *imprecision*, each criterion function g_1 to g_7 admits three performance *discrimination thresholds* of 2.5, 5.0, and 80.0 pts for warranting, respectively, any *indifference*, *preference*, or *considerable performance difference* situation.

Listing 3.2 Inspecting the performance criteria

```

1  >>> pt.showCriteria()
2  -----
3  g1 'RandomPerformanceTableau() instance'
4  Scale = [0.0, 100.0]
5  Weight = 1.0
6  Threshold ind : 2.50 + 0.00x ; percentile: 4.76
7  Threshold pref : 5.00 + 0.00x ; percentile: 9.52
8  Threshold veto : 80.00 + 0.00x ; percentile: 95.24
9  g2 'RandomPerformanceTableau() instance'
10 Scale = [0.0, 100.0]
11 Weight = 1.0
12 Threshold ind : 2.50 + 0.00x ; percentile: 6.67
13 Threshold pref : 5.00 + 0.00x ; percentile: 6.67
14 Threshold veto : 80.00 + 0.00x ; percentile: 100.00
15 ...
16 ...
17 g7 'RandomPerformanceTableau() instance'
18 Scale = [0.0, 100.0]
19 Weight = 1.0
20 Threshold ind : 2.50 + 0.00x ; percentile: 0.00
21 Threshold pref : 5.00 + 0.00x ; percentile: 4.76
22 Threshold veto : 80.00 + 0.00x ; percentile: 100.00

```

On criteria function g_1 , we observe, for instance, about 5% of *indifference* situations, about 90% of *preference* situations, and about 5% of *considerable performance difference* situations (see Lines 6–8 in Listing 3.2).

The individual *performance evaluation* of all decision alternatives on each criterion is gathered in a *performance table*.

Listing 3.3 Inspecting the performance evaluations

```

1  >>> pt.showPerformanceTableau()
2  -----
3  criteria | 'a1'  'a2'  'a3'  'a4'  'a5'  'a6'  'a7'
4  -----
5  'g1'    | 15.2  44.5  57.9  58.0  24.2  29.1  96.6
6  'g2'    | 82.3   43.9   NA    35.8  29.1  34.8  62.2
7  'g3'    | 44.2  19.1  27.7  41.5  22.4  21.5  56.9

```

8	'g4'	46.4	16.2	21.5	51.2	77.0	39.4	32.1	117
9	'g5'	47.7	14.8	79.7	67.5	NA	90.7	80.2	118
10	'g6'	69.6	45.5	22.0	33.8	31.8	NA	48.8	119
11	'g7'	82.9	41.7	12.8	21.9	75.7	15.4	6.0	120

It is noteworthy to mention the three *missing data* (NA) cases: action a3 is missing, for instance, an evaluation on criterion g2 (see Line 6 in Listing 3.3). 121
122

3.2 The Bipolar-Valued Outranking Digraph

123

Given the previous random performance tableau pt, the BipolarOutranking-
Digraph class constructor computes the corresponding *bipolar-valued outranking
digraph*. 124
125
126

Listing 3.4 Example of random bipolar-valued outranking digraph

```

1 >>> from outrankingDigraphs import \
2 ...                               BipolarOutrankingDigraph
3 >>> odg = BipolarOutrankingDigraph(pt)
4 >>> odg
5     ----- Object instance description -----
6     Instance class      : BipolarOutrankingDigraph
7     Instance name       : rel_randomperf
8     Actions             : 7
9     Criteria            : 7
10    Size                : 20
11    Determinateness (%) : 63.27
12    Valuation domain    : [-1.00;1.00]
13    Attributes          : [
14        'name', 'actions',
15        'criteria', 'evaluation', 'NA',
16        'valuationdomain', 'relation',
17        'order', 'gamma', 'notGamma', ...
18    ]

```

127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144

The resulting digraph contains 20 positive (valid) outranking relations. And, the mean majority criteria significance support of all the pairwise outranking situations is 63.3% (see Lines 10–11 in Listing 3.4). 145
146
147

We can inspect the complete $[-1.0, +1.0]$ -valued adjacency table with the showRelationTable() method(). 148
149

Listing 3.5 Inspecting the valued adjacency table

```

1 >>> odg.showRelationTable()
2     * ----- Relation Table -----
3     r(x,y) | 'a1'   'a2'   'a3'   'a4'   'a5'   'a6'   'a7'
4     -----|-----
5     'a1' | +1.00  +0.71  +0.29  +0.29  +0.29  +0.29  +0.00
6     'a2' | -0.71  +1.00  -0.29  -0.14  +0.14  +0.29  -0.57
7     'a3' | -0.29  +0.29  +1.00  -0.29  -0.14  +0.00  -0.29

```

150
151
152
153
154
155
156

```

8   'a4' | +0.00  +0.14  +0.57  +1.00  +0.29  +0.57  -0.43  157
9   'a5' | -0.29  +0.00  +0.14  +0.00  +1.00  +0.29  -0.29  158
10  'a6' | -0.29  +0.00  +0.14  -0.29  +0.14  +1.00  +0.00  159
11  'a7' | +0.00  +0.71  +0.57  +0.43  +0.29  +0.00  +1.00  160
12  Valuation domain: [-1.0; 1.0]  161

```

The BipolarOutrankingDigraph class constructor computes from the given performance tableau pt the characteristic value $r(x \succsim y)$ of a *pairwise outranking* relation $x \succsim y$ in a default *valuation domain* $[-1.0, +1.0]$ with the *median* value 0.0 acting as *indeterminate* characteristic value.

The semantics of $r(x \succsim y)$ are the following:

Definition 3.1 (Semantics of the Outranking Characteristic Function)

- When $r(x \succsim y) > 0.0$, it is *True* that x outranks y , i.e. alternative x is ‘*at least as well evaluated as*’ alternative y on a majority of criteria significance **and** there is no considerable negative performance difference observed in disfavour of x . 168
- When $r(x \succsim y) < 0.0$, it is *False* that x outranks y , i.e. alternative x is **only** ‘*evaluated at least as well as*’ alternative y on a minority of criteria significance **and** there is no considerable positive performance difference observed in favour of x . 171
- When $r(x \succsim y) = 0.0$, it is **indeterminate** whether x outranks y or not. 175

3.3 Pairwise Comparisons

176

From above given semantics, we notice in Line 5 in Listing 3.5 on the preceding page that $a1$ outranks $a2$: $r(a1 \succsim a2) + 0.71$, but not $a7$: $r(a1 \succsim a7) = 0.0$. In order to comprehend the characteristic values shown in the outranking relation table, we can inspect with the `showPairwiseComparison()` method the details of the pairwise multiple-criteria comparison between, for instance, alternatives $a1$ and $a2$.

Listing 3.6 Inspecting a pairwise multiple-criteria comparison

```

1  >>> odg.showPairwiseComparison('a1', 'a2')  183
2  *----- pairwise comparison -----*  184
3  Comparing actions : (a1, a2)  185
4  crit. wght. g(a1)  g(a2)  diff | ind  pref  r()  186
5  ----- 187
6  g1  1.00  15.17  44.51  -29.34 | 2.50  5.00  -1.00  188
7  g2  1.00  82.29  43.90  +38.39 | 2.50  5.00  +1.00  189
8  g3  1.00  44.23  19.10  +25.13 | 2.50  5.00  +1.00  190
9  g4  1.00  46.37  16.22  +30.15 | 2.50  5.00  +1.00  191
10  g5  1.00  47.67  14.81  +32.86 | 2.50  5.00  +1.00  192
11  g6  1.00  69.62  45.49  +24.13 | 2.50  5.00  +1.00  193
12  g7  1.00  82.88  41.66  +41.22 | 2.50  5.00  +1.00  194
13  Valuation in range: -7.00 to +7.00;  195
14  r(a1,a2): +5.00/7.00 = +0.71  +5.00  196

```

The outranking characteristic value $r(a1 \succ a2)$ represents the relative *majority margin* 197 resulting from the difference between the significance weights of the criteria 198 in favour and the significance weights of the criteria in disfavour of the statement 199 that alternative $a1$ is ‘*at least as well evaluated as*’ alternative $a2$. No considerable 200 performance difference being observed, no polarising situation is triggered in this 201 pairwise comparison. 202

Such a polarised situation is however observed when we compare the evaluations 203 of alternatives $a1$ and $a7$ with the `showPairwiseOutrankings()` method. 204

Listing 3.7 Pairwise comparison with considerable performance difference

```

1  >>> odg.showPairwiseOutrankings('a1','a7')          205
2  *----- pairwise comparison -----*               206
3  Comparing actions : (a1, a7)                   207
4  crit. wght. g(a1) g(a7) diff | ind pref r() | v veto 208
5  ----- 209
6  g1  1.00 15.17 96.58 -81.41 | 2.50 5.00 -1.00 | 80.00 -1.00 210
7  g2  1.00 82.29 62.22 +20.07 | 2.50 5.00 +1.00 |           211
8  g3  1.00 44.23 56.90 -12.67 | 2.50 5.00 -1.00 |           212
9  g4  1.00 46.37 32.06 +14.31 | 2.50 5.00 +1.00 |           213
10 g5  1.00 47.67 80.16 -32.49 | 2.50 5.00 -1.00 |           214
11 g6  1.00 69.62 48.80 +20.82 | 2.50 5.00 +1.00 |           215
12 g7  1.00 82.88 6.05 +76.83 | 2.50 5.00 +1.00 |           216
13 ----- 217
14 Valuation in range: -7.00 to +7.00; r(x,y)= +1/7 => 0.0 218
15 *----- pairwise comparison -----*               219
16 Comparing actions : (a1, a7)                   220
17 crit. wght. g(a7) g(a1) diff | ind pref r() | v veto 221
18 ----- 222
19 g1  1.00 96.58 15.17 +81.41 | 2.50 5.00 +1.00 | 80.00 +1.00 223
20 g2  1.00 62.22 82.29 -20.07 | 2.50 5.00 -1.00 |           224
21 g3  1.00 56.90 44.23 +12.67 | 2.50 5.00 +1.00 |           225
22 g4  1.00 32.06 46.37 -14.31 | 2.50 5.00 -1.00 |           226
23 g5  1.00 80.16 47.67 +32.49 | 2.50 5.00 +1.00 |           227
24 g6  1.00 48.80 69.62 -20.82 | 2.50 5.00 -1.00 |           228
25 g7  1.00 6.05 82.88 -76.83 | 2.50 5.00 -1.00 |           229
26 ----- 230
27 Valuation in range: -7.00 to +7.00; r(x,y)= -1/7 => 0.0 231

```

This time, we observe a $(1/7 + 1)/2 = 57.1\%$ majority of criteria significance 232 warranting an ‘*at least as well evaluated as*’ situation between alternative $a1$ 233 and alternative $a7$. Yet, we also observe a considerable *negative* performance 234 difference on criterion $g1$ (see Line 6 in Listing 3.7). Both contradictory facts 235 trigger eventually in Line 14 an *indeterminate* outranking situation. The inverse 236 polarisation effect appears when considering in Lines 19–25 the converse perform- 237 ance differences between alternative $a7$ and alternative $a1$. The considerable 238 better performing situation on criterion $g1$ makes doubtful the otherwise “*not at 239 least as well evaluated as*” situation (see Lines 19 and 27). 240

Notice that the occurrence in a pairwise comparison of conjointly considerable 241 positive and negative performance differences will also trigger an indeterminate 242 outranking situation. When observing at the same time a positive (respectively, 243 negative) “*at least as well evaluated as*” situation and one or more considerable 244 positive (respectively, negative) performance difference, the outranking situation 245 gets validated (respectively, invalidated) for certain (Bisdorff 2013). 246

3.4 Recoding the Characteristic Valuation Domain

247

All outranking digraphs, being of root `Digraph` type, inherit the methods available under this latter class. The characteristic valuation domain of a digraph can, for instance, be recoded with the `recodeValuation()` method to the *integer* range $[-7, +7]$, i.e. plus or minus the total significance weights of the family of criteria considered in this example instance `odg`. 249
250
251
252

Listing 3.8 Recoding the digraph valuation

```

1  >>> odg.recodeValuation(-7,+7)                                253
2  >>> odg.valuationdomain['hasIntegerValuation'] = True        254
3  >>> Digraph.showRelationTable(odg,ReflexiveTerms=False)        255
4  * ----- Relation Table -----                                256
5  r(x,y) | 'a1' 'a2' 'a3' 'a4' 'a5' 'a6' 'a7'                257
6  -----|-----                                258
7  'a1' | - +5 +2 +2 +2 0                                259
8  'a2' | -5 - -1 -1 +1 +2 -4                                260
9  'a3' | -1 +2 - -1 -1 0 -1                                261
10 'a4' | 0 +1 +4 - +2 +4 -3                                262
11 'a5' | -1 0 +1 0 - +2 -1                                263
12 'a6' | -1 0 +1 -1 +1 - 0                                264
13 'a7' | 0 +5 +4 +3 +2 0 -                                265
14 Valuation domain: [-7;+7]                                266

```

Notice in Listing 3.8 that the self comparison characteristics $r(x \succsim x)$ may be ignored by setting the `ReflexiveTerms` parameter to `False`. Mind that the trivial reflexive terms of outranking relations are ignored in some of the DIGRAPH3 methods. 267
268
269
270
271

3.5 The Strict Outranking Digraph

271

From theory, we know that a bipolar-valued outranking digraph is *weakly complete*, i.e. if $r(x \succsim y) < 0.0$, then $r(y \succsim x) \geq 0.0$. From this property follows that a bipolar-valued outranking digraph verifies the *coduality principle*: the *dual*¹—strict negation—of the *converse*—inverse—of the outranking relation $(x \succsim y)$ corresponds to its asymmetric *strict outranking* part $(x \succ y)$ (Bisdorff 2013, 2020). 272
273
274
275
276

Visualising the *codual (strict)* outranking digraph may be done with a `graphviz` drawing (Fig. 3.1).² 277
278

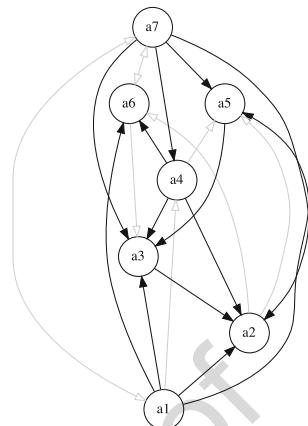
¹ Not to be confused with the dual graph of a plane graph g that has a vertex for each face of g . Here we mean the *less than* (strict converse) relation corresponding to a *greater or equal* relation, or the *less than or equal to* relation corresponding to a (strict) *better than* relation.

² The `exportGraphViz()` method is depending on drawing tools from the `graphviz` software (<https://graphviz.org/>). On Linux Ubuntu or Debian, you may try `sudo apt-get install graphviz` to install them. There are ready `dmg` installers for Mac OSX.

AQ1

AQ2

Fig. 3.1 The strict (codual) outranking digraph. It becomes readily clear now from the picture that both alternatives a_1 and a_7 are *not outranked* by any other alternatives. Hence, a_1 and a_7 appear as *weak* CONDORCET winners and may be recommended as potential *best* decision alternatives in this illustrative preference modelling example



Digraph3 (graphviz), R. Bisdorff, 2020

```

1  >>> cdodg = -(~odg)                                279
2  >>> cdodg.exportGraphViz('codualOdg')             280
3  *---- exporting a dot file for GraphViz tools -----* 281
4  Exporting to codualOdg.dot                         282
5  dot -Grankdir=BT -Tpng codualOdg.dot -o codualOdg.png 283

```

Many more tools for exploiting bipolar-valued outranking digraphs are available in the DIGRAPH3 resources (Bisdorff 2021). 284
285

Notes

The seminal work on outranking digraphs goes back to the seventies and eighties when *Bernard Roy* joined the just starting University Paris-Dauphine and founded there the ‘*Laboratoire d’Analyse et de Modélisation de Systèmes pour l’Aide à la Décision*’ (LAMSADE). The LAMSADE became the major site in the development of the outranking approach to multiple-criteria decision aiding (Roy and Bouyssou 1993). 292

The ongoing success of the original *outranking* concept stems from the fact that it is rooted in a sound pragmatism. The multiple-criteria performance tableau, necessarily associated with a given outranking digraph, is indeed convincingly objective and meaningful (Roy 1991). And, ideas from social choice theory gave initially the insight that a pairwise voting mechanism à la CONDORCET could provide an order-statistical tool for aggregating a set of preference points of view into what *Marc Barbut* called the *central* CONDORCET point of view (de Caritat, Marquis de Condorcet 1784 and Barbut 1980), in fact the median of the multiple preference points of view, at minimal absolute KENDALL’s ordinal correlation distance from all individual points of view (see Chap. 16). 301

Considering thus each performance criterion as a subset of unanimous voters 303 and balancing the votes in favour against considerable counter-performances in 304 disfavour gave eventually rise to the concept of *outranking situation*, a distinctive 305 feature of the Multiple-Criteria Decision Aiding approach (Bisdorff et al. 2015). A 306 modern definition would be an alternative x is said to *outrank* alternative y when— 307 a *significant majority* of criteria confirm that alternative x has to be considered 308 as *at least as well evaluated as* an alternative y (the *concordance argument*), 309 and—no discordant criterion opens to significant doubt the validity of the previous 310 confirmation by revealing a considerable counter-performance of alternative x 311 compared to y (the *discordance argument*). 312

If the concordance argument was always well received, the discordance argument 313 however, very confused in the beginning (Benayoun et al. 1966), could only be 314 handled in an epistemically correct and logically sound way by using a bipolar- 315 valued epistemic logic (see Definition 3.1 and Bisdorff 2013). The outranking 316 situation had consequently to receive an explicit negative definition: an alternative 317 x is said to *do not outrank* an alternative y when—a *significant majority* of criteria 318 confirm that alternative x has to be considered as *not at least as well evaluated as* 319 alternative y , and—no discordant criterion opens to significant doubt the validity 320 of the previous confirmation by revealing a considerable *better* performance of 321 alternative x compared to y . 322

Furthermore, the initial conjunctive aggregation of the concordance and discor- 323 dance arguments had to be replaced by a disjunctive epistemic fusion operation, 324 polarising in a logically sound and epistemically correct way the concordance with 325 the discordance argument. This way, bipolar-valued outranking digraphs gained two 326 very useful properties from a measure theoretical perspective. They are *weakly* 327 *complete*; incomparability situations are no longer attested by the absence of 328 positive outranking relations, but instead by epistemic indeterminateness. And 329 they verify the *coduality principle*: the negation of the epistemic ‘*at least as well* 330 *evaluated as*’ situation corresponds formally to the strict converse epistemic ‘*less* 331 *well evaluated than*’ situation. 332

In the next methodological Part II, we present and discuss multiple-criteria eval- 333 uation models and decision algorithms, like building a best choice recommendation, 334 determining the winner of an election and computing linear rankings or quantile 335 ratings with multiple incommensurable criteria. 336

References

337

- Barbut M (1980) Médianes, Condorcet et Kendall. *Math Sci Hum* 69:9–13 338
Benayoun R, Roy B, Sussmann B (1966) ELECTRE: une méthode pour guider le choix en présence 339
de points de vue multiples. *Tech. Rep.* 49, Société d’Economie et de Mathématique Appliquée, 340
Direction Scientifique 341
Bisdorff R (2013) On polarizing outranking relations with large performance differences. *J Multi- 342
Criteria Dec Anal Wiley* 20:3–12. <http://hdl.handle.net/10993/245> 343

- Bisdorff R (2020) Lecture 7: best multiple criteria choice: the Rubis outranking method. In: Lectures of the algorithmic decision theory course, University of Luxembourg. <http://hdl.handle.net/10993/37933> 344
- Bisdorff R (2021) Technical documentation of the Digraph3 collection of Python modules. <https://digraph3.readthedocs.io/en/latest/techDoc.html> 347
- Bisdorff R, Dias L, Meyer P, Mousseau V, Pirlot M (eds) (2015) Evaluation and decision models with multiple criteria: case studies. International handbooks on information systems. Springer, Berlin. <http://hdl.handle.net/10993/23698> 349
- de Caritat, Marquis de Condorcet J (1784) *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. Imprimerie royale, Paris 352
- Roy B (1991) The outranking approach and the foundations of electre methods. *Theory Decis* 31(1):49–73 354
- Roy B, Bouyssou D (1993) *Aide Multicritère à la Décision : Méthodes et Cas*. Economica, Paris 355
- Roy B, Bouyssou D (1993) *Aide Multicritère à la Décision : Méthodes et Cas*. Economica, Paris 356

AUTHOR QUERIES

- AQ1. Please check the sentence “From this property follows that ...” for clarity.
- AQ2. Missing citation for Fig. 3.1 was inserted here. Please check if appropriate.
- AQ3. Please check the term “à la CONDORCET” for correctness in sentence “And, ideas from social choice ...” and amend if required.

Uncorrected Proof

Part II ¹

Evaluation Models and Decision ²

Algorithms ³

The second and methodological part of the book presents in eight chapters multiple-⁴ criteria evaluation models and decision algorithms for selecting a best decision ⁵ alternative, for ranking the potential decision alternative from best to worst, and ⁶ for relative or absolute quantile rating with the help of bipolar-valued outranking ⁷ digraphs. We also show how to edit a new multiple-criteria performance tableau and ⁸ present several models of random performance tableau generators. The last chapter ⁹ is devoted to HPC ranking of big performance tableaux. ¹⁰

Chapter 4

Building a Best Choice Recommendation

1

2

*... The goal of our research was to design a resolution method ... 3
that is easy to put into practice, that requires as few and reliable 4
hypotheses as possible, and that meets the needs [of the decision 5
maker]... 6*

—(Benayoun *et al.* 1966) 7

Contents

8

4.1	What Office Location to Choose?	41	9
4.2	The Given Performance Tableau	43	10
4.3	Computing the Outranking Digraph.....	45	11
4.4	Designing a Best Choice Recommender System	47	12
4.5	Computing the RUBIS Best Choice Recommendation	48	13
4.6	Weakly Ordering the Outranking Digraph	50	14

Abstract This chapter presents the RUBIS best choice recommender system. Our 15
approach is illustrated with a best office location selection problem. We show how to 16
explore the given performance tableau and compute the corresponding outranking 17
digraph. After presenting the pragmatic principles that govern our best choice 18
recommendation algorithm we solve the best office location choice problem. 19

4.1 What Office Location to Choose?

20

A SME, specialised in printing and copy services, has to move into new offices, 21
and its CEO has gathered seven *potential new office locations* (see Table 4.1 on the 22
[following page](#)). 23

Three *decision objectives*, in order of decreasing importance, are guiding the 24
CEO's choice: 25

1. *maximise* the future turnover of the SME,

26

Table 4.1 The potential new office locations

ID	Name	Address	Comment	
A	Ave	Avenue de la liberté	High standing city centre	t3.1
B	Bon	Bonnevoie	Industrial environment	t3.2
C	Ces	Cessange	Residential suburb location	t3.3
D	Dom	Dommeldange	Industrial suburb environment	t3.4
E	Bel	Esch-Belval	New and ambitious urbanisation far from the city	t3.5
F	Fen	Fentange	Out in the countryside	t3.6
G	Gar	Avenue de la Gare	Main city shopping street	t3.7
				t3.8

Table 4.2 The family of performance criteria

Objective	ID	Name	Weight	Comment	
Yearly costs	Ct	Costs	45	Annual rent, charges, and cleaning	t6.1
Future turnover	Pr	Proximity	32	Distance from town centre	t6.2
Future turnover	V	Visibility	26	Circulation of potential customers	t6.3
Future turnover	St	Standing	23	Image and presentation	t6.4
Working conditions	W	Space	10	Working space	t6.5
Working conditions	Cf	Comfort	6	Quality of office equipment	t6.6
Working conditions	P	Parking	3	Available parking facilities	t6.7
					t6.8

2. *minimise* the future yearly costs induced by the moving, 27

3. *maximise* the new working conditions. 28

The decision consequences to take into account for evaluating the potential new office locations with respect to each one of the three objectives are modelled by the *family of performance criteria*¹ shown in Table 4.2 below. 29
30
31

In Table 4.2 we notice that the *Costs* criterion admits the highest significance 32 weight (45), followed by the *Future turnover* criteria (32, 26, 23). The *Working* 33 *conditions* criteria are the less significant (10, 6, 3).² It follows that the CEO 34 considers *maximising the future turnover* the most important objective ((32 + 26 + 35 23) = 81), followed by minimising the future yearly costs (45), and less important, 36 *maximising working conditions* ((10 + 6 + 3) = 19). 37

The evaluations of the seven potential new locations on each performance 38 criterion are gathered in a *performance table* shown in Table 4.3 on the next page. 39 All criteria, except the *Costs* Criterion, admit for evaluation a qualitative satisfaction 40 scale from 0% (weakest) to 100% (highest). One may thus notice that location 41 A (*Ave*) is the most expensive, but also 100% satisfying the *Proximity* as well as 42 the *Standing* criterion. Whereas location C (*Ces*) is the cheapest one; Providing, 43 however, no satisfaction at all on both the *Standing* and the *Working Space* criteria. 44

¹ See Roy (2000).

² These criteria weights were supposedly established with a swing weighing MCDA method (Keeney and Raiffa 1976).

Table 4.3 Performance evaluations of the potential office locations

Criterion	A	B	C	D	E	F	G	
Costs	35.0K€	17.8K€	6.7K€	14.1K€	34.8K€	18.6K€	12.0K€	t9.1
Proximity	100	20	80	70	40	0	60	t9.2
Visibility	60	80	70	50	60	0	100	t9.3
Standing	100	10	0	30	90	70	20	t9.4
Working space	75	30	0	55	100	0	50	t9.5
Working comfort	0	100	10	30	60	80	50	t9.6
Parking	90	30	100	90	70	0	80	t9.7
								t9.8

Concerning yearly costs, we suppose that the CEO is indifferent up to a 45 performance difference of 1000.00€, and he actually prefers a location when there 46 is at least a positive difference of 2500.00€. The evaluations observed on the six 47 qualitative criteria (measured in percentages of satisfaction) are very subjective and 48 rather imprecise. The CEO is hence *indifferent* up to a satisfaction difference of 49 10%, and he claims a significant *preference* when the satisfaction difference is at 50 least of 20%. Furthermore, a satisfaction difference of 80% represents for him a 51 *considerably large* performance difference, triggering the case given a *polarisation* 52 of the preferential situation (Bisdorff 2013). 53

In view of Table 4.3, what is now the office location we may recommend to the 54 CEO as **best choice**? 55

4.2 The Given Performance Tableau

56

The file `officeChoice.py`, stored in the `examples` directory of the 57 DIGRAPH3 resources, provides a corresponding `PerformanceTableau` object. 58 We can inspect its actual content with the computing resources provided by the 59 `perfTabs` module. 60

Listing 4.1 Inspecting the `officeChoice` performance tableau

```

1  >>> from perfTabs import PerformanceTableau
2  >>> pt = PerformanceTableau('officeChoice')
3  >>> pt
4  *----- PerformanceTableau instance description -----*
5  Instance class      : PerformanceTableau
6  Instance name       : officeChoice
7  Actions             : 7
8  Objectives          : 3
9  Criteria            : 7
10 NaN proportion (%) : 0.0
11 Attributes          : ['name', 'actions', 'objectives',
12                           'criteria', 'weightPreorder',
13                           'NA', 'evaluation']
14 >>> pt.showPerformanceTableau()

```

15	----- performance tableau -----*	75
16	Criteria 'Ct' 'Cf' 'P' 'Pr' 'St' 'V' 'W'	76
17	Weights 45.00 6.00 3.00 32.00 23.00 26.00 10.00	77
18	-----	78
19	'Ave' -35000.00 0.00 90.00 100.00 100.00 60.00 75.00	79
20	'Bon' -17800.00 100.00 30.00 20.00 10.00 80.00 30.00	80
21	'Ces' -6700.00 10.00 100.00 80.00 0.00 70.00 0.00	81
22	'Dom' -14100.00 30.00 90.00 70.00 30.00 50.00 55.00	82
23	'Bel' -34800.00 60.00 70.00 40.00 90.00 60.00 100.00	83
24	'Fen' -18600.00 80.00 0.00 0.00 70.00 0.00 0.00	84
25	'Gar' -12000.00 50.00 80.00 60.00 20.00 100.00 50.00	85

We thus recover all the input data shown in Sect. 4.1. Notice that the *negative* evaluations of the *Costs* criterion indicate a negative preference direction: the *lower* the costs, the *better* it is.

The `showCriteria()` method evaluates the actual preference discrimination we observe on each performance criterion.

Listing 4.2 Inspecting the performance criteria

1	>>> pt.showCriteria(IntegerWeights=True)	91
2	----- criteria -----*	92
3	Ct 'Costs'	93
4	Preference direction: min	94
5	Scale = (0.00, 50000.00)	95
6	Weight = 45	96
7	Threshold ind : 1000.00 + 0.00x ; percentile: 9.52	97
8	Threshold pref : 2500.00 + 0.00x ; percentile: 14.29	98
9	Cf 'Comfort'	99
10	Preference direction: max	100
11	Scale = (0.00, 100.00)	101
12	Weight = 6	102
13	Threshold ind : 10.00 + 0.00x ; percentile: 9.52	103
14	Threshold pref : 20.00 + 0.00x ; percentile: 28.57	104
15	Threshold veto : 80.00 + 0.00x ; percentile: 90.48	105
16	...	106
17	...	107

On the *Costs* criterion, 9.5% of the performance differences are considered insignificant and 14.3% below the preference discrimination threshold (see Lines 7–8 in Listing 4.2). On the qualitative *Comfort* criterion, we observe again 9.5% of insignificant performance differences (line 13). Due to the imprecision in the subjective evaluations, we notice here 28.6% of performance differences below the preference discrimination threshold (Line 14). Furthermore, $100.0 - 90.5 = 9.5\%$ of the performance differences are judged *considerably large* (Line 15) and will trigger hence a polarisation of the concerned outranking situations (Bisdorff 2013). Same information is available for all the other criteria. A colourful comparison of all the evaluations is shown in Fig. 4.1 on the facing page by the *heatmap* statistics, illustrating the respective quantile class of each evaluation. As the set of potential alternatives is tiny, we choose here a classification into performance quintiles.

Heatmap of Performance Tableau 'officeChoice'

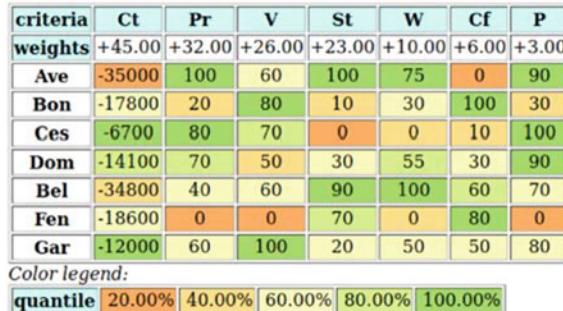


Fig. 4.1 Unranked heatmap of the office choice performance tableau

```

1 >>> pt.showHTMLPerformanceHeatmap(colorLevels=5, \
2 ...                                         rankingRule=None)

```

Location Ave shows extreme and contradictory evaluations: highest *Costs* and no *Working Comfort* on the one hand, and total satisfaction with respect to *Standing*, *Proximity*, and *Parking facilities* on the other hand. Similar, but opposite, situation is given for location Ces: unsatisfactory *Working Space*, no *Standing*, and no *Working Comfort* on the one hand, and lowest *Costs*, best *Proximity*, and *Parking facilities* on the other hand. Contrary to these contradictory alternatives, we observe two appealing compromise alternatives: locations Dom and Gar. Finally, location Fen is clearly the less satisfactory alternative of all. To help now the CEO choosing the best office location, we are going to compute pairwise outranking situations on the set of potential decision alternatives (see Bis dorff 2013).

4.3 Computing the Outranking Digraph

Definition 4.1 (Outranking Situation) For two potential decision alternatives x and y :

- ‘Alternative x outranks alternative y ’, denoted $(x \succsim y)$, is given when:
 1. A *majority* of criteria significance warrants that alternative x is *at least as well evaluated as* alternative y , and
 2. *No considerable* negative performance difference is observed on any criterion.
- ‘Alternative x does not outrank y ’, denoted $(x \not\succsim y)$, is given when:
 1. Only a *minority* of criteria significance warrants that alternative x is *at least as well evaluated as* alternative y , and
 2. *No considerable* positive performance difference is observed on any criterion.

Fig. 4.2 Bipolar-valued adjacency matrix. In the resulting outranking relation we may notice, on the one hand, that Alternative D is *positively outranking* all other potential office locations. On the other hand, alternatives A (the most expensive) and C (the cheapest) are *not outranked* by any other location

Valued Adjacency Matrix

$r(x \precsim y)$	A	B	C	D	E	F	G
A	-	0.00	1.00	0.30	0.78	0.00	0.00
B	0.00	-	0.00	-0.56	0.00	1.00	-0.60
C	0.00	0.00	-	0.46	0.00	1.00	0.10
D	0.10	0.56	0.02	-	0.46	1.00	0.25
E	0.52	0.00	0.00	-0.10	-	1.00	-0.42
F	0.00	-1.00	-1.00	-1.00	-1.00	-	-1.00
G	0.00	0.92	-0.10	1.00	0.54	1.00	-

Valuation domain: [-1.00; +1.00]

- Otherwise, the outranking situation between alternatives x and y is considered to be *indeterminate*. 143
144

The credibility of each pairwise outranking situation, denoted $r(x \succsim y)$, is measured in a bipolar significance valuation $[-1.0, 1.0]$, where *positive* terms $r(x \succsim y) > 0.0$ indicate a *validated outranking*, and *negative* terms $r(x \succsim y) < 0.0$ indicate an *invalidated outranking*, i.e. a *validated outranked* situation. The *median* value $r(x \succsim y) = 0.0$ represents an *indeterminate* situation (see Bisdorff 2004, 2013). 145
146
147
148
149

For computing such a bipolar-valued binary outranking relation from the given performance tableau pt , we use the `BipolarOutrankingDigraph` class from the `outrankingDigraphs` module. The corresponding `showHTMLRelationTable()` method shows here the resulting bipolar-valued adjacency matrix in a system browser window (see Fig. 4.2). 150
151
152
153
154

Listing 4.3 Computing a bipolar-valued outranking digraph

```
1 >>> from outrankingDigraphs import BipolarOutrankingDigraph 155
2 >>> bod = BipolarOutrankingDigraph(pt) 156
3 >>> bod.showHTMLRelationTable() 157
```

Alternative D gives a CONDORCET winner,³ whereas alternatives A (the most expensive) and C (the cheapest) give in fact *weak* CONDORCET winners. 158
159

```
1 >>> bod.computeCondorcetWinners () 160
2 [ 'D' ] 161
3 >>> bod.computeWeakCondorcetWinners () 162
4 [ 'A', 'C', 'D' ] 163
```

From theory, we know that outranking digraphs are *weakly complete*, i.e. for all x and y in X , $r(x \succsim y) < 0.0$ implies that $r(y \succsim x) \geq 0.0$. And, they verify the *coduality principle*: $r(x \not\succsim y) = r(y \not\succsim x)$ (Bisdorff 2013).⁴ 164
165
166

³ See Chap. 7 on computing the winner of an election.

⁴ Not to be confused with the dual graph of a plane graph g that has a vertex for each face of g . Here we mean the *less than* (strict converse) relation corresponding to a *greater or equal* relation, or the *less than or equal* relation corresponding to a (strict) *better than* relation.

Definition 4.2 (Strict Outranking Situation) For two potential decision alternatives x and y , ‘ x strictly outranks alternative y ’, denoted $(x \succ y)$, when ‘ x outranks alternative y ’ $((x \succ y) > 0.0)$ and “ y does not outrank alternative x ” $((y \succ x) < 0.0)$.

Following from the coduality principle, the strict outranking digraph `bodcd` is built from the outranking digraph `bod` with a codual transform (see Sect. 2.6).

```

1 >>> bodcd = ~(-bod)  # codual transform          173
2 >>> bodcd
3 *----- Object instance description -----*          174
4 Instance class          : BipolarOutrankingDigraph 175
5 Instance name           : converse-dual-rel_officeChoice 176
6 Actions                 : 7                         177
7 Criteria                : 7                         178
8 Size                    : 10                        179
9 Determinateness (%)     : 72.38                     180
10 Valuation domain        : [-1.00;1.00]          181

```

4.4 Designing a Best Choice Recommender System

183

Solving a best choice problem consists traditionally in finding *the* unique best decision alternative. We adopt here instead a modern recommender system’s approach which shows a non-empty subset of decision alternatives which contains by construction the potential best alternative(s).

The five *pragmatic principles* for computing such a *best choice recommendation* (BCR) are the following:

- P1: *Elimination for well motivated reasons*; each eliminated alternative has to be strictly outranked by at least one alternative in the BCR.
- P2: *Minimal size*; the BCR must be as limited in cardinality as possible.
- P3: *Efficient and informative*; the BCR must not contain a self-contained sub-recommendation.
- P4: *Effectively better*; the BCR must not be ambiguous in the sense that it may not be both a first choice as well as a last choice recommendation.
- P5: *Maximally determined*; the BCR is, of all potential best choice recommendations, the most determined one in the sense of the epistemic characteristics of the bipolar-valued outranking relation.

Let X be the set of potential decision alternatives. Let Y be a non-empty subset of X , called a *choice* in the strict outranking digraph $G(X, r(\succ))$. We can now qualify a BCR Y in following terms:

- Y is called strictly *outranking* (respectively, *outranked*) when for all not selected alternative x there exists an alternative $y \in X$ retained such that $r(y \succ x) > 0.0$ (respectively, $r(y \preceq x) > 0.0$). Such a choice verifies principle P1.

- Y is called *weakly independent* when for all $x \neq y$ in Y we observe $r(x \succsim y) \leq 0.0$. Such a choice verifies principles P3 (*internal stability*). 206
- Y is conjointly a strictly *outranking* (respectively, *outranked*) and *weakly independent* choice. Such a choice is called an *initial* (respectively, *terminal*) 208
prekernel.⁵ The initial prekernel now verifies principles P1, P2, P3, and P4. 209
- To finally verify principle P5, we recommend among all potential initial pre- 211
kernels, a *most determined* one, i.e. a strictly *outranking* and *weakly independent* 212
choice supported by the highest criteria significance. And in this most determined 213
initial prekernel we eventually retain the alternative(s) that are included with 214
highest criteria significance.⁶ 215

Mind that a given strict outranking digraph may not always admit prekernels. 216
This is the case when the digraph contains chordless circuits of odd length (see 217
Chap. 17). Luckily, our strict outranking digraph `bodcd` here does not show any 218
chordless outranking circuits; a fact we can check with the `computeChordless- 219
Circuits()` method followed by the `showChordlessCircuits()` 220
method.⁷ 221

```
1 >>> bodcd.computeChordlessCircuits () 222
2 []
3 >>> bodcd.showChordlessCircuits () 223
4 No circuits observed in this digraph. 224
5
```

When observing chordless odd outranking circuits, we need to break them open 226
with the `BrokenCocsDigraph` class at their weakest link, before enumerating 227
the prekernels (Bisdorff 2021). 228

We are ready now for building a best choice recommendation. 229

4.5 Computing the RUBIS Best Choice Recommendation

The `showBestChoiceRecommendation()` method computes the RUBIS best 231
choice recommendation directly from the outranking digraph `bod`. By default this 232
method is operating on the *codual* (strict) outranking digraph where chordless odd 233
circuits have been broken up (see the `CoDual` and `BrokenCocs` parameters in 234
Listing 4.4 Line 2): 235

Listing 4.4 Computing the best choice recommendation

```
1 >>> bod.showBestChoiceRecommendation (\ 236
2 ... CoDual=True, BrokenCocs=True # default 237
3 settings\ 238
```

⁵ See Chap. 17 on computing kernels in digraphs.

⁶ See Sect. 17.6.

⁷ The `computeChordlessCircuits()` and `showChordlessCircuits()` methods are separate because there are various methods available for enumerating the chordless circuits in a digraph (Bisdorff 2010).

```

3 ... ChoiceVector = True) 239
4 * --- First and last choice recommendation(s) ---* 240
5 (in decreasing order of determinateness) 241
6 Credibility domain: [-1.00,1.00] 242
7 === >> potential first choice(s) 243
8 * choice : ['A', 'C', 'D'] 244
9 independence : 0.00 245
10 dominance : 0.10 246
11 absorbency : 0.00 247
12 covering (%) : 41.67 248
13 determinateness (%) : 50.59 249
14 - characteristic vector = { 250
15 'D': +0.02, 'G': 0.00, 'C': 0.00, 'A': 0.00, 251
16 'F': -0.02, 'E': -0.02, 'B': -0.02, } 252
17 === >> potential last choice(s) 253
18 * choice : ['A', 'F'] 254
19 independence : 0.00 255
20 dominance : -0.52 256
21 absorbency : 1.00 257
22 covered (%) : 50.00 258
23 determinateness (%) : 50.00 259
24 - characteristic vector = { 260
25 'G': 0.00, 'F': 0.00, 'E': 0.00, 'D': 0.00, 261
26 'C': 0.00, 'B': 0.00, 'A': 0.00, } 262

```

It is interesting to notice in Line 8 above that the RUBIS *first choice recommendation* consists actually in the previously mentioned set of weak CONDORCET winners: A, C, and D (see Fig. 4.2 on page 46). In the corresponding prekernel characteristic vector (see Lines 3, 15 and Sect. 17.6), representing the bipolar credibility degree with which each alternative may indeed be included in, or excluded from this recommendation, we find that alternative D is the only positively validated one, whereas both extreme alternatives—A (the most expensive) and C (the cheapest)—stay in an indeterminate situation (see Bisдорff 2006; Bisдорff et al. 2006). They may *be or not be* potential best choices. Notice furthermore that compromise alternative G, while not actually being included in this strictly outranking prekernel, shows as well an indeterminate situation with respect to *being or not being* recommended as potential best choice. Alternatives B, E, and F are all negatively included, i.e. positively excluded, from this best choice recommendation (see Line 16).

To inspect why alternative D is the only positive best choice recommendation, we shall compare now the evaluations of alternatives D and G in a pairwise perspective.

Listing 4.5 Inspecting pairwise comparison between alternatives G and D

```

1 >>> bod.showPairwiseComparison('G','D') 279
2 *----- pairwise comparison -----* 280
3 Comparing actions : ('G', 'D') 281
4 crit. wght. g(x) g(y) diff. | ind. pref. concord. 282
5 ====== 283
6 Costs 45.00 -12000.00 -14100.00 +2100.00 | 1000.00 2500.00 +45.00 284
7 Comf. 6.00 50.00 30.00 +20.00 | 10.00 20.00 +6.00 285
8 Park. 3.00 80.00 90.00 -10.00 | 10.00 20.00 +3.00 286
9 Prox. 32.00 60.00 70.00 -10.00 | 10.00 20.00 +32.00 287

```

```

10  Stdg. 23.00    20.00    30.00   -10.00 |   10.00   20.00  +23.00  288
11  Visi. 26.00   100.00    50.00  +50.00 |   10.00   20.00  +26.00  289
12  Spac. 10.00    50.00    55.00   -5.00 |   10.00   20.00  +10.00  290
13  =====
14  Valuation in range: -145.00 to +145.00; global concordance: +145.00  291

```

In Listing 4.5 on the preceding page, we notice that, with the given preference discrimination thresholds, alternative G is actually *certainly at least as well evaluated as* alternative D: $r(G \succsim D) = +145/145 = +1.0$.

Yet, we must as well acknowledge in Listing 4.6 that the cheapest alternative C is in fact *strictly outranking* alternative G: $r(C \succsim G) = +15/145 > 0.0$, and $r(G \succsim C) = -15/145 < 0.0$.

Listing 4.6 Inspecting pairwise comparison between alternatives C and G

```

1 >>> bod.showPairwiseComparison('C','G')
2 *----- pairwise comparison -----
3 Comparing actions : (C,G)/(G,C)
4 crit. wght. g(x) g(y) diff. | ind. pref. (C,G) / (G,C)
5 =====
6 'C' 45.00 -6700.00 -12000.00 +5300.00 | 1000.00 2500.00 +45.00/-45.00 304
7 'Cf' 6.00 10.00 50.00 -40.00 | 10.00 20.00 -6.00/+6.00 305
8 'P' 3.00 100.00 80.00 +20.00 | 10.00 20.00 +3.00/-3.00 306
9 'Pr' 32.00 80.00 60.00 +20.00 | 10.00 20.00 +32.00/-32.00 307
10 'St' 23.00 0.00 20.00 -20.00 | 10.00 20.00 -23.00/+23.00 308
11 'V' 26.00 70.00 100.00 -30.00 | 10.00 20.00 -26.00/+26.00 309
12 'W' 10.00 0.00 50.00 -50.00 | 10.00 20.00 -10.00/+10.00 310
13
14 Valuation in range: -145 to +145; r(C >= G) / r(G >= c) : +15.00/-15.00 312

```

Let us finally notice in Listing 4.4 on page 48 Line 18 that both alternatives A and F are reported as potential last choice recommendation. Yet, this last choice recommendation appears to be globally indeterminate (Lines 25–26). This confirms the *incomparability* status of alternative A (see Fig. 4.3 on the next page).

```

1 >>> bodcd.exportGraphViz(fileName='bestOfficeChoice', \
2 ...                         firstChoice=['C','D'], \
3 ...                         lastChoice=['F'])
4 *---- exporting a dot file for GraphViz tools -----
5 Exporting to bestOfficeChoice.dot
6 dot -Grankdir=BT -Tpng bestOfficeChoice.dot \
7 -o bestOfficeChoice.png 323

```

4.6 Weakly Ordering the Outranking Digraph

To get a global insight in the overall strict outranking situations, we may use the RankingByChoosingDigraph class imported from the transitive-Digraphs module for computing a *ranking-by-choosing* result from the codual, i.e. the strict outranking digraph instance bodcd (see above). If the computing node supports multiple processor cores, *first* and *last* choosing iterations may be run in parallel (see Line 4 in Listing 4.7 on the facing page).

Fig. 4.3 Best office choice recommendation from strict outranking digraph. Notice that location A (Ave) (the most expensive) is appearing *incomparable* to all the other alternatives

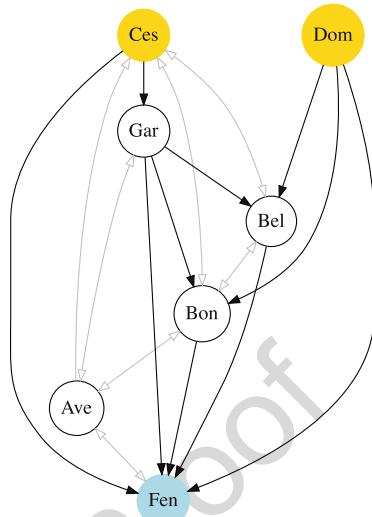


Diagram3 (graphviz), R. Bisdorff, 2020

Listing 4.7 Ranking-by-choosing the outranking digraph

```

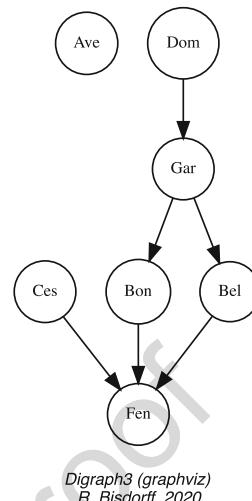
1  >>> from transitiveDigraphs import\
2      ...                               RankingByChoosingDigraph
3  >>> rbc = RankingByChoosingDigraph(bodcd)
4  Threading ... # multiprocessing if 2 cores are available
5  Exiting computing threads
6  >>> rbc.showRankingByChoosing()
7  Ranking by Choosing and Rejecting
8  1st ranked ['D']
9      2nd ranked ['C', 'G']
10     2nd last ranked ['B', 'C', 'E']
11     1st last ranked ['A', 'F']
12  >>> rbc.exportGraphViz('officeChoiceRanking')
13  *---- exporting a dot file for GraphViz tools -----
14  Exporting to officeChoiceRanking.dot
15  dot -Grankdir=TB -Tpng officeChoiceRanking.dot\
16          -o officeChoiceRanking.png

```

The best choice recommendation hence depends on the very importance the CEO is attaching to each of his decision objectives. In the given setting here, where he considers that *maximising the future turnover* is the most important objective followed by *minimising the Costs* and, less important, *maximising the working conditions*, location D represents actually the *best compromise*. However, if *Costs* do not play much a role, it would be perhaps better to decide to move to the most advantageous location A; or if, on the contrary, *Costs* do matter a lot, moving to the cheapest alternative C could definitely represent a more convincing recommendation (Fig. 4.4).

Fig. 4.4

Ranking-by-choosing the potential office locations. In this *ranking-by-choosing* method, where we operate the *epistemic fusion* of iterated (strict) first and last choices, compromise alternative Dom is now ranked before compromise alternative Gar. The overall partial ordering result shows again the important fact that the most expensive location, Ave, and the cheapest location, Ces, due to their contradictory performances appear both *incomparable* with most of the other alternatives



It might be worth editing the criteria significance weights in the `office Choice.py` data file in such a way that: 356
357

- All three decision objectives are considered *equally important*, and 358
- All criteria under each objective are considered *equi-significant*. 359

What will become the best choice recommendation under this working hypothesis?⁸ 360
361

In the next Chap. 5 we precisely show how to edit a new performance tableau 362
from a given template file. 363

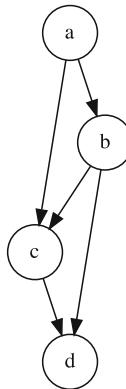
Notes

364

Following a seminar presentation in 2005 at the LAMSADE,⁹ where the author 365
promoted the use of kernels of the outranking digraph as suitable candidates 366
for delivering best choice recommendations (Bisdorff 2005), a critical discussion 367
started about the methodological requirement for a convincing best choice rec- 368
ommendation to be internally stable (pragmatic principle P3). Denis Bouyssou 369
illustrated his doubts with the potential outranking digraph shown in Fig. 4.5 on 370
the next page. 371

⁸ See also the notes of Lecture 7 from the MICS Algorithmic Decision Theory course (Bisdorff 2020).

⁹ Laboratoires d'Analyse et de Modélisation de Systèmes d'Aide à la Décision, Université Paris-Dauphine, UMR 7243 CNRS.



Digraph3 (graphviz), R. Bisendorff, 2020

Fig. 4.5 The internal stability of a best choice recommendation in question. The only kernel of this digraph is the pair {a, d}; yet, it is an ambiguous recommendation, as {a, d} is conjointly an outranking and outranked choice. If the instability of the best choice recommendation is, however, not considered a problem then the choice {a, b} shows the most convincing strict outranking quality and could be considered in priority for recommendation as potential best choice candidates

His commentary was the following: Adding alternative d to the set of potential best choice candidates is not convincing as there exists in the given digraph the node b, which is better evaluated than d. The argument that the incomparability between a and d should favour d as potential best choice is interesting but another hypothesis could be that b perhaps outranks a. In this latter case, it seems clear that the actual best choice recommendation should be reduced to node b, unless one disposes of other information, like a performance tableau and/or the actual computation method of the outranking situations. In any case, one has to be very clear about the available information when judging a best choice procedure.

It became thereafter obvious for us all that both the lack of a specific performance tableau as well as the lack of a precisely defined algorithm for computing valid outranking situations do not allow to judge if a given digraph does indeed model a potential outranking relation. In our present bipolar-valued epistemic approach, a valid outranking digraph instance, following from a given performance tableau and the disjunctive epistemic fusion construction of the outranking relation (see Chap. 3), will necessarily verify the weak completeness condition and the coduality principle. As a consequence, incomparability situations are now modelled by epistemic indeterminateness not by the actual absence of a reciprocal outranking relation.

The digraph put forward by *D. Bouyssou* in the October 2005 discussion is not weakly complete—node a is not outranking node d and vice versa—and does hence not represent, in our present sense, a valid outranking digraph instance. Yet, it may be a partial tournament and as such it could be a strict outranking digraph,

i.e. the asymmetric part—the codual—of a valid outranking digraph. In this case, 395
 nodes a and d —the kernel of the strict outranking digraph—would actually for sure 396
 outrank each other and, hence, represent both indifferently the natural best choice 397
 recommendation. However, in this not strict codual digraph, node a becomes also 398
 the unique CONDORCET winner—outranking for sure all other nodes—and gives 399
 hence the evident unique best choice recommendation. 400

Only after 2013, when the weak completeness and the coduality properties 401
 of the outranking digraph were discovered, it became obvious that the initial 402
 prekernels of the strict outranking digraph, coupled with the solution of the 403
 corresponding kernel equation system, were in fact delivering the most convincing 404
 best choice recommendations (see Chap. 17, Sect. 20.5 and Bis dorff 2013). It stays 405
 an interesting open mathematical problem to show (or not) that both necessary 406
 conditions—weak completeness and coduality—are also sufficient for qualifying 407
 any bipolar-valued digraph as potential instance of an outranking digraph. 408

References

409

- Benayoun R, Roy B, Sussmann B (1966) ELECTRE: une méthode pour guider le choix en présence 410
 de points de vue multiples. Tech. Rep. 49, Société d'Economie et de Mathématique Appliquée, 411
 Direction Scientifique 412
- Bis dorff R (2004) On a natural fuzzification of Boolean logic. In: Klement EP, Pap E (eds) Linz 413
 seminar on fuzzy set theory, mathematics of fuzzy systems, Bildungszentrum St. Magdalena, 414
 Linz (Austria), pp 20–26. <http://hdl.handle.net/10993/23721> 415
- Bis dorff R (2005) Exploitation en problématique du choix d'une relation de surclassement valuée. 416
<http://hdl.handle.net/10993/47659> 417
- Bis dorff R (2006) On enumerating the kernels in a bipolar-valued digraph. Ann Lamsade 6:1–38. 418
<http://hdl.handle.net/10993/38741> 419
- Bis dorff R (2010) Enumerating chordless circuits in directed graphs. In: ORBEL24-2010, 24th 420
 annual conference of the Belgian Operational Research Society (ORBEL aka Sogesci- 421
 B.V.W.B.), January 28–29, Liège (BE), Université de Liège (BE), pp 1–12. <http://hdl.handle.net/10993/23926> 422
- Bis dorff R (2013) On polarizing outranking relations with large performance differences. J Multi- 423
 Criteria Decis Anal Wiley 20:3–12. <http://hdl.handle.net/10993/245> 424
- Bis dorff R (2020) Lecture 7: Best multiple criteria choice: the Rubis outranking method. In: 425
 Lectures of the algorithmic decision theory course, University of Luxembourg, <http://hdl.handle.net/10993/37933> 426
- Bis dorff R (2021) Technical documentation of the Digraph3 collection of Python modules. <https://digraph3.readthedocs.io/en/latest/techDoc.html> 427
- Bis dorff R, Pirlot M, Roubens M (2006) Choices and kernels from bipolar valued digraphs. Eur J 428
 Oper Res 175:155–170. <http://hdl.handle.net/10993/23720> 429
- Keeney R, Raiffa H (1976) Decisions with multiple objectives: preferences and value tradeoffs. 430
 Cambridge University Press 431
- Roy B (2000) A French-English decision aiding glossary. News! Eur Working Group Multicriteria 432
 Aid Decis 3(Suppl 1):1–10 433

AUTHOR QUERY

AQ1. Missing citation for Fig. 4.4 was inserted here. Please check if appropriate.

Uncorrected Proof

Chapter 5

How to Create a New Multiple-Criteria Performance Tableau

Contents

5.1	Editing a Template File	55	5
5.2	Editing the Decision Alternatives	57	6
5.3	Editing the Decision Objectives	58	7
5.4	Editing the Family of Performance Criteria	59	8
5.5	Editing the Performance Evaluations	61	9
5.6	Inspecting the Template Outranking Relation	63	10

Abstract The chapter illustrates a way of creating a new `PerformanceTableau` instance by editing a given template with five decision alternatives, three decision objectives, and six performance criteria. We discuss in detail editing the decision alternatives, the decision objectives, the family of performance criteria, and finally, the evaluations of the decision alternatives on the performance criteria.

5.1 Editing a Template File

For easing the editing of a new multiple-criteria performance tableau, we provide the following `perfTab_Template.py` file in the `examples` directory of the DIGRAPH3 resources.

Listing 5.1 `PerformanceTableau` object template

```
1 #####  
2 # Digraph3 documentation  
3 # Template for creating a new PerformanceTableau instance  
4 # (C) R. Bisdorff Mar 2021  
5 # Digraph3/examples/perfTab_Template.py  
6 #####  
7 from decimal import Decimal  
8 from collections import OrderedDict  
9 #####  
10 # edit the decision actions  
11 # avoid special characters, like '_', '/' or ':',  
12 # in action identifiers and short names  
13 actions = OrderedDict([
```

```

14 ('a1', { 33
15   'shortName': 'action1', 34
16   'name': 'decision alternative a1', 35
17   'comment': 'some specific features of this alternative', 36
18   }), 37
19   ... 38
20   ... 39
21 ])
22 #####
23 # edit the decision objectives 41
24 # adjust the list of performance criteria 42
25 # and the total weight (sum of the criteria weights) 43
26 # per objective 44
27 objectives = OrderedDict([
28   ('obj1', { 46
29     'name': 'decision objective obj1', 47
30     'comment': "some specific features of this objective", 48
31     'criteria': ['g1', 'g2'], 49
32     'weight': Decimal('6'), 50
33     }), 51
34   ... 52
35   ... 53
36 ])
37 #####
38 # edit the performance criteria 56
39 # adjust the objective reference 57
40 # Left Decimal of a threshold = constant part and 59
41 # right Decimal = proportional part of the threshold 60
42 criteria = OrderedDict([
43   ('g1', { 61
44     'shortName': 'crit1', 62
45     'name': "performance criteria 1", 63
46     'objective': 'obj1', 64
47     'preferenceDirection': 'max', 65
48     'comment': 'measurement scale type and unit', 66
49     'scale': (Decimal('0.0'), Decimal('100.0'), 67
50     'thresholds': {'ind': (Decimal('2.50'), Decimal('0.0')), 68
51       'pref': (Decimal('5.00'), Decimal('0.0')), 69
52       'veto': (Decimal('60.00'), Decimal('0.0'))}, 70
53     }, 71
54     'weight': Decimal('3'), 72
55   }), 73
56   ... 74
57   ... 75
58 ])
59 #####
60 # default missing data symbol = -999 78
61 NA = Decimal('-999')
62 #####
63 # edit the performance evaluations 82
64 # criteria to be minimized take negative evaluations 83
65 evaluation = { 84
66   'g1': { 85
67     'a1':Decimal("41.0"), 86
68     'a2':Decimal("100.0"), 87
69     'a3':Decimal("63.0"), 88
70     'a4':Decimal('23.0'), 89
71     'a5': NA, 90
72   }, 91
73   # g2 is of ordinal type and scale 0-10 92
74   'g2': { 93
75     'a1':Decimal("4"), 94
76     'a2':Decimal("10"), 95
77     'a3':Decimal("6"), 96
78     'a4':Decimal('2'), 97
79     'a5':Decimal('9'), 98
80   },

```

```

81 # g3 has preferenceDirection = 'min'          100
82 'g3': {                                     101
83     'a1':Decimal("-52.2"),                   102
84     'a2':NA,                                103
85     'a3':Decimal("-47.3"),                   104
86     'a4':Decimal('-35.7'),                  105
87     'a5':Decimal('-68.00'),                 106
88 },
89 ...
90 ...
91 }
92 #####
```

The template file, shown in Listing 5.1 on page 55, contains first the instructions to import the required standard Python Decimal and OrderedDict classes (see Lines 7–8). Four main sections are following: the potential decision *actions*, the decision *objectives*, the performance *criteria*, and finally the performance *evaluations*.

5.2 Editing the Decision Alternatives

Decision alternatives are stored in attribute *actions* under the OrderedDict format. The OrderedDict object keeps this initial order when iterating over the decision alternatives.¹

Required attributes of each decision alternative, besides the object identifier,² are: shortName, name, and comment (see Lines 15–17 in Listing 5.1 on page 55). The shortName attribute is essentially used when showing the performance tableau or the performance heatmap in a browser view.

The random performance tableau models, introduced in Chap. 6, use the actions attribute for storing special features of the decision alternatives. The *Cost-Benefit* model, for instance, uses a type attribute for distinguishing between *advantageous*, *neutral*, and *cheap* alternatives (see Sect. 6.3). The *3-Objectives* model keeps a detailed record of the performance profile per decision objective and the corresponding random generators per performance criteria as shown in Listing 5.2.

Listing 5.2 Example of decision alternative description

```

1 >>> from randomPerfTabs import \
132
2 ...           Random3ObjectivesPerformanceTableau
133
3 >>> t = Random3ObjectivesPerformanceTableau()
134
4 >>> t.actions
135
5     OrderedDict([
136
```

¹ See the [OrderedDict description in the Python documentation](#) (Python Software Foundation 2021).

² Mind that graphviz drawings require node identifier strings without any special characters like ‘_’ or ‘/’.

```

6   ('p01', {'shortName': 'p01',
7     'name': 'action p01 Eco~ Soc- Env+',
8     'comment': 'random public policy',
9     'Eco': 'fair',
10    'Soc': 'weak',
11    'Env': 'good',
12    'profile': {'Eco': 'fair',
13      'Soc': 'weak',
14      'Env': 'good'}
15    'generators': {'ec01': ('triangular', 50.0, 0.5),
16      'so02': ('triangular', 30.0, 0.5),
17      'en03': ('triangular', 70.0, 0.5),
18      ...
19    },
20  },
21  ...
22 ]
23 )

```

The second section of the template file concerns the decision *objectives*.

155

5.3 Editing the Decision Objectives

156

The minimal required attributes of the ordered decision *objectives* dictionary, 157
 besides the individual objective identifiers, are *name*, *comment*, *criteria*: the 158
 list of significant performance criteria, and *weight*: the importance of the decision 159
 objective. The latter attribute contains the sum of the *significance* weights of the 160
 objective's criteria list (see Lines 27–33 in Listing 5.1 on page 55). 161

The *objectives* attribute is methodologically useful for specifying the 162
 performance criteria significance in building decision recommendations. Mostly, 163
 we assume indeed that decision objectives are all equally important and the 164
 performance criteria are equi-significant per objective. This is, for instance, the 165
 default setting in the random 3-*Objectives* performance tableau model. 166

Listing 5.3 Example of decision objectives' description

```

1 >>> # t = Random3ObjectivesPerformanceTableau()
2 >>> t.objectives
3 OrderedDict([
4   ('Eco',
5     {'name': 'Economical aspect',
6      'comment': 'Random3ObjectivesPerformanceTableau generated',
7      'criteria': ['ec01', 'ec06', 'ec09'],
8      'weight': Decimal('48')}),
9   ('Soc',
10    {'name': 'Societal aspect',
11      'comment': 'Random3ObjectivesPerformanceTableau generated',
12      'criteria': ['so02', 'so12'],
13      'weight': Decimal('48')}),

```

```

14 ('Env',
15   {'name': 'Environmental aspect',
16   'comment': 'Random3ObjectivesPerformanceTableau generated',
17   'criteria': ['en03', 'en04', 'en05', 'en07',
18                 'en08', 'en10', 'en11', 'en13'],
19   'weight': Decimal('48'))} 180
20 ]) 181

```

The importance weight sums up to 48 for each one of the three example decision objectives shown in Listing 5.3 on the facing page so that the significance weight of each one of the 3 economic criteria is set to 16, of both societal criterion is set to 24, and of each one of the 6 environmental criteria is set to 8 (see Lines 8, 13 and 19).

Mind that the `objectives` attribute is always present in a `PerformanceTableau` object instance, even when empty. In this case, we consider that each performance criterion canonically represents its own decision objective. The criterion significance weight equals in this case the corresponding decision objective's importance weight.

The third section of the template file concerns now the *performance criteria*.

5.4 Editing the Family of Performance Criteria

In order to assess how well each potential decision alternative is satisfying a given decision objective, we need *performance criteria*, i.e. decimal-valued evaluation functions gathered in an ordered `criteria` dictionary. The required attributes (see Listing 5.4), besides the criteria identifiers, are the usual `shortName`, `name`, and `comment`. Specific for a criterion is furthermore the `objective` reference, the significance `weight`, the evaluation `scale` (minimum and maximum performance values), the `preferenceDirection` ('`max`' or '`min`'), and the `performance` discrimination `thresholds` attributes.

Listing 5.4 Example of performance criteria description

```

1 criteria = OrderedDict([
2   ('g1', {
3     'shortName': 'crit1',
4     'name': "performance criteria 1",
5     'comment': 'measurement scale type and unit',
6     'objective': 'obj1',
7     'weight': Decimal('3'),
8     'scale': (Decimal('0.0'), Decimal('100.0'),
9               'preferenceDirection': 'max',
10              'thresholds': {'ind': (Decimal('2.50'), Decimal('0.0')),
11                            'pref': (Decimal('5.00'), Decimal('0.0')),
12                            'veto': (Decimal('60.00'), Decimal('0.0'))},
13              },
14 },
15 ...
16 ...
17 )
18 ...
19 ...
20 ...
21 ...
22 ...
23 ...
24 ...
25 ...
26 ...
27 ...
28 ...
29 ...
30 ...
31 ...
32 ...
33 ...
34 ...
35 ...
36 ...
37 ...
38 ...
39 ...
40 ...
41 ...
42 ...
43 ...
44 ...
45 ...
46 ...
47 ...
48 ...
49 ...
50 ...
51 ...
52 ...
53 ...
54 ...
55 ...
56 ...
57 ...
58 ...
59 ...
60 ...
61 ...
62 ...
63 ...
64 ...
65 ...
66 ...
67 ...
68 ...
69 ...
70 ...
71 ...
72 ...
73 ...
74 ...
75 ...
76 ...
77 ...
78 ...
79 ...
80 ...
81 ...
82 ...
83 ...
84 ...
85 ...
86 ...
87 ...
88 ...
89 ...
90 ...
91 ...
92 ...
93 ...
94 ...
95 ...
96 ...
97 ...
98 ...
99 ...
100 ...
101 ...
102 ...
103 ...
104 ...
105 ...
106 ...
107 ...
108 ...
109 ...
110 ...
111 ...
112 ...
113 ...
114 ...
115 ...
116 ...
117 ...
118 ...
119 ...
120 ...
121 ...
122 ...
123 ...
124 ...
125 ...
126 ...
127 ...
128 ...
129 ...
130 ...
131 ...
132 ...
133 ...
134 ...
135 ...
136 ...
137 ...
138 ...
139 ...
140 ...
141 ...
142 ...
143 ...
144 ...
145 ...
146 ...
147 ...
148 ...
149 ...
150 ...
151 ...
152 ...
153 ...
154 ...
155 ...
156 ...
157 ...
158 ...
159 ...
160 ...
161 ...
162 ...
163 ...
164 ...
165 ...
166 ...
167 ...
168 ...
169 ...
170 ...
171 ...
172 ...
173 ...
174 ...
175 ...
176 ...
177 ...
178 ...
179 ...
180 ...
181 ...
182 ...
183 ...
184 ...
185 ...
186 ...
187 ...
188 ...
189 ...
190 ...
191 ...
192 ...
193 ...
194 ...
195 ...
196 ...
197 ...
198 ...
199 ...
200 ...
201 ...
202 ...
203 ...
204 ...
205 ...
206 ...
207 ...
208 ...
209 ...
210 ...
211 ...
212 ...
213 ...
214 ...
215 ...
216 ...
217 ...
218 ...
219 ...
220 ...
221 ...

```

In our bipolar-valued outranking approach, all performance criteria implement *decimal-valued* evaluation functions, where preferences are either *increasing* or *decreasing* with measured performances. In order to model a *coherent* performance tableau, the family of decision criteria must satisfy two methodological requirements:

1. *Independence*: Each decision criterion implements an evaluation that is *functionally independent* of the evaluation of the other decision criteria, i.e. the performance evaluated on one of the criteria does not *constrain* in any sense the performance evaluated on any other criterion. 227-230
 2. *Non-redundancy*: Each performance criterion is only *significant* for a *single* decision objective. 231-232

For taking into account any, usually *unavoidable, imprecision* of the performance evaluation procedures, we may specify three performance *discrimination thresholds*: an *indifference* (ind), a *preference* (pref), and a *considerable performance difference* (veto) threshold (see Listing 5.4 on the preceding page Lines 10–12). The left decimal number of a threshold description tuple indicates a *constant part*, whereas the right decimal number indicates a *proportional part*.

On the template performance criterion $g1$, shown in Listing 5.4 on the previous page, we observe, for instance, an evaluation scale from 0.0 to 100.0 with a constant *indifference* threshold of 2.5, a constant *preference* threshold of 5.0, and a constant *considerable performance difference* threshold of 60.0. The latter threshold will trigger, the case given, a *polarisation* of the outranking statement (Bisdorff 2013).

In a random *Cost-Benefit* performance tableau model we may obtain by default 244
the following description of a cardinal *Costs* criterion: 245

Listing 5.5 Example of cardinal *Costs* criterion

```

22     'thresholds': OrderedDict([
23         ('ind', (Decimal('1.49'), Decimal('0'))),
24         ('pref', (Decimal('3.7'), Decimal('0'))),
25         ('veto', (Decimal('67.71'), Decimal('0'))),
26     ])
27 },
28 ...
29 ...
30 ])

```

Criterion `c1` appears here to be a cardinal criterion to be minimised and significant for the *Costs* (`C`) decision objective (see Line 13 in Listing 5.5 on the facing page). The `showCriteria()` prints out the corresponding performance discrimination thresholds.

```

1 >>> tcb.showCriteria(IntegerWeights=True)
2 ----- criterion -----
3 c1 'Costs/random cardinal cost criterion'
4 Preference direction: min
5 Scale = (0.0, 100.0)
6 Weight = 6
7 Threshold ind : 1.49 + 0.00x ; percentile: 5.13
8 Threshold pref : 3.70 + 0.00x ; percentile: 10.26
9 Threshold veto : 67.71 + 0.00x ; percentile: 96.15
10 ...
11 ...

```

The *indifference* threshold on this criterion amounts to a constant value of 1.49 (Line 6 above). More or less 5% of the observed performance differences on this criterion appear hence to be *insignificant*. Similarly, with a preference threshold of 3.70, about 90% of the observed performance differences are preferentially *significant* (Line 7). Furthermore, $100.0 - 96.15 = 3.85\%$ of the observed performance differences appear to be *considerable* (Line 8) and will trigger, the case given, a *polarisation* of the corresponding outranking situations.

After the performance criteria descriptions, we are ready for recording the actual *performance evaluations*.

5.5 Editing the Performance Evaluations

300

The individual evaluations of each decision alternative on each decision criterion are recorded in a double *criterion* \times *action* dictionary called `evaluation` (see Listing 5.5). As we may encounter cases of missing data, we previously define a *missing data* symbol `NA` which is set to a value disjoint from all the measurement scales, by default `Decimal ('-999')` (see Line 2 in Listing 5.6 on the next page).

Listing 5.6 Editing performance evaluations

```

1 #-----
2 NA = Decimal('-999')
3 #-----
4 evaluation = {
5   'g1': {
6     'a1': Decimal("41.0"),
7     'a2': Decimal("100.0"),
8     'a3': Decimal("63.0"),
9     'a4': Decimal('23.0'),
10    'a5': NA, # missing data
11  },
12  ...
13  ...
14 # g3 has preferenceDirection = 'min'
15 'g3': {
16   'a1': Decimal("-52.2"), # negative grades
17   'a2': NA,
18   'a3': Decimal("-47.3"),
19   'a4': Decimal('-35.7'),
20   'a5': Decimal('-68.00'),
21 },
22  ...
23  ...
24 }
```

Notice in Listing 5.6 that on a criterion with `preferenceDirection = 'min'` all performance evaluations are recorded as *negative* values (Lines 16 and following).

We can now inspect below the eventually edited complete template performance tableau `perfTab_Template.py` with the `showPerformanceTableau()` method.

```

1 >>> from perfTabs import PerformanceTableau
2 >>> pt = PerformanceTableau('perfTab_Template')
3 >>> pt.showPerformanceTableau(ndigits=1)
4 *---- performance tableau ----*
5 Criteria | 'g1'   'g2'   'g3'   'g4'   'g5'   'g6'
6 Actions  | 3       3       6       2       2       2
7 *----|----*-----|-----*-----|-----*-----*-----*
8 'action1' | 41.0   4.0    -52.2  71.0   63.0   22.5
9 'action2' | 100.0  10.0   NA     89.0   30.7   75.0
10 'action3' | 63.0   6.0    -47.3  55.4   63.5   NA
11 'action4' | 23.0   2.0    -35.7  83.5   37.5   54.9
12 'action5' | NA     9.0    -68.0  10.0   88.0   75.0
```

Computing below the associated outranking digraph `bod` allows checking the potential presence of any polarised outranking situations.

```

1 >>> from outrankingDigraphs import BipolarOutrankingDigraph
2 >>> bod = BipolarOutrankingDigraph(pt)
3 >>> bod.showPolarisations()
4 *---- Negative polarisations ----*
```

```

5  number of negative polarisations : 1          354
6  1: r(a4 >= a2) = -0.44                      355
7    criterion: g1                                356
8    Considerable performance difference : -77.00  357
9    Veto discrimination threshold      : -60.00  358
10   Polarisation: r(a4 >= a2) = -0.44 ==> -1.00 359
11 *---- Positive polarisations ----*          360
12   number of positive polarisations : 1          361
13  1: r(a2 >= a4) = 0.56                      362
14    criterion: g1                                363
15    Considerable performance difference : +77.00 364
16    Counter-veto threshold      : 60.00       365
17   Polarisation: r(a2 >= a4) = 0.56 ==> +1.00 366

```

Indeed, due to the considerable positive performance difference (+77.00) observed on performance criterion g_1 , alternative a_2 “*for sure outranks*” alternative a_4 , respectively, a_4 “*does for sure not outrank*” a_2 .

5.6 Inspecting the Template Outranking Relation

370

In Listing 5.7, the `showRelationTable()` method prints out the outranking relation table.

371

372

Listing 5.7 The template outranking relation

```

1 >>> bod.showRelationTable()          373
2 *---- Relation Table ----          374
3   r   | 'a1'   'a2'   'a3'   'a4'   'a5'  375
4   --- | -----          376
5   'a1' | +1.00  -0.44  -0.22  -0.11  +0.06 377
6   'a2' | +0.44  +1.00  +0.33  +1.00  +0.28 378
7   'a3' | +0.67  -0.33  +1.00  +0.00  +0.17 379
8   'a4' | +0.11  -1.00  +0.00  +1.00  +0.06 380
9   'a5' | -0.06  -0.06  -0.17  -0.06  +1.00 381

```

One may notice in this outranking relation table above that decision alternative a_2 positively *outranks* all the other four alternatives (Line 6). Similarly, alternative a_5 is positively *outranked* by all the other alternatives (see Line 9). We can orient this way the *graphviz* drawing of the template outranking digraph.

382

383

384

385

```

1 >>> bod.exportGraphViz(fileName= 'template', \          386
2 ...                      firstChoice =['a2'], \          387
3 ...                      lastChoice=['a5'])          388
4 *---- exporting a dot file for GraphViz tools ----* 389
5   Exporting to template.dot                         390
6   dot -Grankdir=BT -Tpng template.dot -o template.png 391

```

In Fig. 5.1 on the following page, alternatives $action3$ (a_3) and $action4$ (a_4) appear actually *incomparable*. In Listing 5.7, their pairwise outranking characteristics show indeed the *indeterminate* value 0.00 (Lines 7–8).

392

393

394

Checking their pairwise comparison may be done with the `showPairwiseComparison()` method:

```

1 >>> bod.showPairwiseComparison('a3','a4') 397
2 *----- pairwise comparison -----* 398
3 Comparing actions : ('a3','a4') 399
4 crit. wght. g(a3) g(a4) diff | ind pref r() | 400
5 ----- 401
6 'g1' 3.00 63.00 23.00 +40.00 | 2.50 5.00 +3.00 | 402
7 'g2' 3.00 6.00 2.00 +4.00 | 0.00 1.00 +3.00 | 403
8 'g3' 6.00 -47.30 -35.70 -11.60 | 0.00 10.00 -6.00 | 404
9 'g4' 2.00 55.40 83.50 -28.10 | 2.09 4.18 -2.00 | 405
10 'g5' 2.00 63.50 37.50 +26.00 | 0.00 10.00 +2.00 | 406
11 'g6' NA 54.90 407
12 Outranking characteristic value: r(a3 >= a4) = +0.00 408
13 Valuation in range: -18.00 to +18.00 409

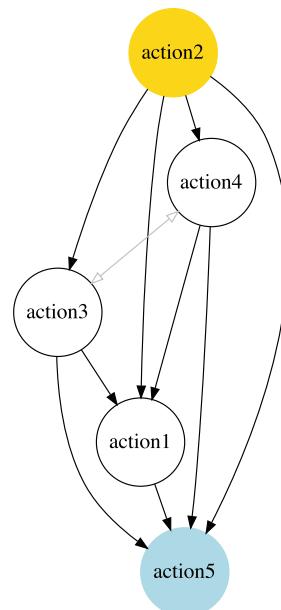
```

The incomparability situation between α_3 and α_4 results in fact from a perfect balancing of positive (+8) and negative (-8) criteria significance weights. 410
411

The five decision alternatives can finally be ranked with a heatmap browser view 412 following the COPELAND ranking rule (see Sect. 8.2), a rule which consistently 413 reproduces the partial outranking order shown in Fig. 5.1. 414

```
1 >>> bod.showHTMLPerformanceHeatmap(ndigits=1, \
2 ...     colorLevels=5, Correlations=True, \
3 ...     rankingRule='Copeland', \
4 ...     pageTitle=\
5 ...         'Heatmap of the template performance tableau')
```

Fig. 5.1 *The template outranking digraph.* The digraph bod models in fact a partial order on the five potential decision alternatives



criteria	crit4	crit1	crit3	crit2	crit6	crit5
weights	+2.00	+3.00	+6.00	+3.00	+2.00	+2.00
tau(*)	+0.60	+0.40	+0.35	+0.20	+0.10	-0.60
action2	89.0	100.0	NA	10.0	75.0	30.7
action3	55.4	63.0	-47.3	6.0	NA	63.5
action4	83.5	23.0	-35.7	2.0	54.9	37.5
action1	71.0	41.0	-52.2	4.0	22.5	63.0
action5	10.0	NA	-68.0	9.0	75.0	88.0

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%
----------	--------	--------	--------	--------	---------

(*) tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation
Outranking model: **standard**, Ranking rule: **Copeland**

Ordinal (Kendall) correlation between global ranking and global outranking relation: **+1.000**

Mean marginal correlation (a) : **+0.228**

Standard marginal correlation deviation (b) : **+0.322**

Ranking fairness (a) - (b) : **-0.094**

Fig. 5.2 COPELAND ranked heatmap of the template performance tableau

criteria	crit2	crit6	crit1	crit4	crit3	crit5
weights	+3.00	+2.00	+3.00	+2.00	+6.00	+2.00
tau(*)	+0.60	+0.50	+0.40	+0.20	-0.05	-0.20
action2	10.0	75.0	100.0	89.0	NA	30.7
action3	6.0	NA	63.0	55.4	-47.3	63.5
action5	9.0	75.0	NA	10.0	-68.0	88.0
action4	2.0	54.9	23.0	83.5	-35.7	37.5
action1	4.0	22.5	41.0	71.0	-52.2	63.0

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%
----------	--------	--------	--------	--------	---------

(*) tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation

Outranking model: **standard**, Ranking rule: **NetFlows**

Ordinal (Kendall) correlation between global ranking and global outranking relation: **+0.920**

Mean marginal correlation (a) : **+0.206**

Standard marginal correlation deviation (b) : **+0.286**

Ranking fairness (a) - (b) : **-0.081**

Fig. 5.3 NETFLOWS ranked heatmap of the template performance tableau

Due to an 8 against 7 *plurality tyranny* effect, the COPELAND ranking rule, 420
essentially based on crisp majority outranking counts, puts here alternative *action5* 421
(a5) last, despite its excellent evaluations observed on criteria g2, g5 and g6 422
(Fig. 5.2). 423

A slightly *fairer* ranking result may be obtained in Fig. 5.3 with the NETFLOWS 424
ranking rule (see Sect. 8.3). 425

```

1 >>> bod.showHTMLPerformanceHeatmap(ndigits=1, \
2 ...     colorLevels=5, Correlations=True, \
3 ...     rankingRule='NetFlows', \
4 ...     pageTitle= \
5 ...         'Heatmap of the template performance tableau')

```

It might be opportune to furthermore study the robustness of the apparent outranking 431
situations when assuming *uncertain* or solely *ordinal* criteria significance weights 432
(see Chaps. 18 and 19). 433

The next Chap. 6 describes DIGRAPH3 resources for generating random 434
multiple-criteria performance tableaux of various kinds like Cost-Benefit, three- 435
objectives, or academic tableaux. 436

References

437

Bisdorff R (2013) On polarizing outranking relations with large performance differences. J Multi- 438
Criteria Decis Anal Wiley 20:3–12. <http://hdl.handle.net/10993/245> 439
Python Software Foundation (2021) Python documentation. <https://docs.python.org/3/> 440

AUTHOR QUERIES

- AQ1.** Please check if the edit made to the sentence “The importance weight sums up...” is fine.
- AQ2.** Missing citation for Fig. 5.2 was inserted here. Please check if appropriate.

Uncorrected Proof

Chapter 6	1
Generating Random Performance	2
Tableaux	3

Contents	4
6.1 Introduction	67 5
6.2 Random Standard Performance Tableaux	68 6
6.3 Random Cost-Benefit Performance Tableaux	71 7
6.4 Random Three Objectives Performance Tableaux	74 8
6.5 Random Academic Performance Tableaux	79 9

Abstract The chapter describes the DIGRAPH3 resources for generating random multiple-criteria performance tableaux like a *Cost-Benefit* tableau, a three Objectives—*economic*, *societal*, and *environmental*—tableau, and an *academic* performance tableau. 10
11
12
13

6.1 Introduction

The `randomPerfTabs` module provides several classes for generating random performance tableaux models of different kind, mainly for the purpose of testing implemented methods and tools presented and discussed in the Algorithmic Decision Theory course lectures at the University of Luxembourg. This chapter introduces several of these performance tableau models. 15
16
17
18
19

The simplest class, called `RandomPerformanceTableau`, generates a set of n decision actions, a family of m real-valued performance criteria, ranging by default from 0.0 to 100.0, associated with default discrimination thresholds: 2.5 (ind), 5.0 (pref), and 60.0 (veto). The generated random evaluations are by default Beta (2, 2) distributed on each measurement scale. 20
21
22
23
24

One of the most useful models involving two decision objectives, named *Costs* (to be minimised), respectively, *Benefits* (to be maximised), is provided by the `RandomCBPerformanceTableau` class; its purpose being to generate more or less contradictory performances on these two, usually conflicting, objectives. 25
26
27
28
29
30

Many public policy decision problems often involve three more or less 31
 conflicting decision objectives taking into account *economical*, *societal* 32
 as well as *environmental* aspects. For this type of performance tableau 33
 model, the `randomPerfTabs` module provides a specific class, called 34
`Random3ObjectivesPerformanceTableau`. 35

Deciding which students, based on their grades obtained in a number of 36
 examinations, validate or not their academic studies, is the genuine decision 37
 practice of universities and academies. To thoroughly study these kind of decision 38
 problems, the `randomPerfTabs` module provides a corresponding performance 39
 tableau model, called `RandomAcademicPerformanceTableau`, which gath- 40
 ers grades obtained by a given number of students in a given number of weighted 41
 courses. 42

6.2 Random Standard Performance Tableaux

The `RandomPerformanceTableau` class, the simplest of the kind, specialises 44
 the generic `PerformanceTableau` class, and takes the following parameters: 45

- `numberOfActions` := number of decision actions. 46
- `numberOfCriteria` := number of performance criteria. 47
- `weightDistribution` :=
 '`random`' (default) | '`fixed`' | '`equisignificant`':
 - If '`random`', weights are uniformly selected randomly from the given 50
 weight scale; 51
 - If '`fixed`', the `weightScale` must provide a corresponding weights distribu- 52
 tion; 53
 - If '`equi-significant`', all criterion weights are put to unity. 54
- `weightScale` := `[Min, Max]` (default =`(1, numberOfCriteria)`). 55
- `IntegerWeights` := `True` (default) | `False` (normalised to proportions of 56
 `1.0`). 57
- `commonScale` := `[a, b]`; common performance measuring scales (default = 58
 `[0.0, 100.0]`). 59
- `commonThresholds` := `[(q0, q1), (p0, p1), (v0, v1)]`; `indifference(q)`, 60
 preference (`p`) and considerable performance difference (`v`) discrimina- 61
 tion thresholds. For each threshold type $x \in \{q, p, v\}$, the float $x0$ value rep- 62
 resents a *constant percentage* of the common scale and the float $x1$ value 63
 a *proportional value* of the actual performance measure. Default values are 64
 `[(2.5, 0.0), (5.0, 0.0), (60.0, 0, 0)]`. 65

- `commonMode` := common distribution of random performance measurements:¹ 66
67
 - ('beta', `None` (default setting), (α, β)), a beta generator with default $\alpha = 2$ 68
69 and $\beta = 2$ parameters.
 - ('uniform', `None`, `None`), uniformly distributed float values on the given 70
71 common scales' range [Min, Max];
 - ('normal', μ, σ), truncated Gaussian distribution, by default $\mu = (b-a)/2$ 72
73 and $\sigma = (b-a)/4$;
 - ('triangular', *mode*, *repartition*), generalised triangular distribution 74
75 with a probability repartition parameter specifying the probability mass 76
77 accumulated until the mode value. By default, *mode* = $(b-a)/2$ and *repartition* = 0.5.² 77
- `valueDigits` := `integer`, precision of performance measurements (2 decimal 78
79 digits by default).
- `missingDataProbability` := $0.0 \leq \text{float} \leq 1.0$; probability of missing 80
81 performance evaluation on a criterion for an alternative (default 0.025).
- `NA` := `Decimal` (default = -999); missing data symbol. 82

Code Example

83

Listing 6.1 Generating a random performance tableau

```

1 >>> from randomPerfTabs import RandomPerformanceTableau 84
2 >>> t = RandomPerformanceTableau(numberOfActions=21, \ 85
3 ...           numberOfCriteria=13, seed=100) 86
4 >>> t.actions 87
5 { 'a01': { 88
6   'comment': 'RandomPerformanceTableau() generated.', 89
7   'name': 'random decision action' 90
8 }, 91
9   'a02': { ... }, 92
10  ... 93
11 } 94
12 >>> t.criteria 95
13 { 'g01': { 96
14   'thresholds': { 97
15     'ind' : (Decimal('10.0'), Decimal('0.0')), 98
16     'veto': (Decimal('80.0'), Decimal('0.0')), 99
17     'pref': (Decimal('20.0'), Decimal('0.0'))}, 100
18   'scale': [0.0, 100.0], 101
19   'weight': Decimal('1'), 102
20   'name': 'RandomPerformanceTableau() instance', 103
21   'comment': "Arguments: weightDistribution=random, 104
22           weightScale=(1, 1); 105

```

¹ See Lecture 3 of the Computational Statistic Course (Bisdorff 2020).

² The `randomNumbers` module provides for this purpose the `ExtendedTriangularRandomVariable` class.

```

23                                     commonMode=None"
24     },
25     'g02': { ... },
26     ...
27 }
28 >>> t.NA
29     Decimal('-999')
30 >>> t.evaluation
31     {'g01': {'a01': Decimal('15.17'),
32                 'a02': Decimal('44.51'),
33                 'a03': Decimal('-999'), # missing evaluation
34                 ... },
35     ...
36 }
37 >>> t.showHTMLPerformanceTableau()

```

Highest and lowest evaluations on each criterion are marked in *light green*, respectively, in *light red*. Notice that missing (NA) evaluations are recorded in a performance tableau by default as `Decimal ('-999')` value (see Listing 6.1 on the preceding page Lines 28–29; Fig. 6.1).

106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124

AQ2

Performance table randomperf.tab

criteria	g01	g02	g03	g04	g05	g06	g07	g08	g09	g10	g11	g12	g13
weight	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
a01	15.17	46.37	82.88	41.14	59.94	41.19	58.68	44.73	22.19	64.64	34.93	42.36	17.55
a02	44.51	16.22	41.66	53.58	31.39	65.22	71.96	57.84	78.08	77.37	8.30	63.41	61.55
a03		21.53	12.82	56.93	26.80	48.03	54.35	62.42	94.27	73.57	71.11	21.81	56.90
a04	58.00	51.16	21.92	65.57	59.02	44.77	37.49	58.39	80.79	55.39	46.44	19.57	39.22
a05	24.22	77.01	75.74	83.87	40.85	8.55	85.44	67.34	57.40	39.08	64.83	29.37	96.39
a06	29.10	39.35	15.45	34.99	49.12	11.49	28.44	52.89	64.24	62.92	58.28	32.02	10.25
a07	96.58	32.06	6.05	49.56		66.06	41.64	13.08	38.31	24.82	48.39	57.03	42.91
a08	82.29	47.67	9.96	79.43	29.45	84.17	31.99	90.88	39.58	50.78	61.88	44.40	48.26
a09	43.90	14.81	60.55	42.37	6.72	56.14	34.20	51.54	21.79	79.13	50.95	93.16	81.89
a10	38.75	79.70	27.88	42.39	71.88	66.09	58.33	58.88	17.10	44.25	48.73	30.63	52.73
a11	35.84	67.48	38.81	33.75	26.87	64.10	71.95	62.72	NA	85.80	58.37	49.33	NA
a12	29.12	13.97	67.45	38.60	48.30	11.87		57.76	74.86	26.57	48.80	43.57	7.68
a13	34.79	90.72	38.93	57.38	64.14	97.86	91.16	43.80	33.68	38.98	28.87	63.36	60.03
a14	62.22	80.16	19.26	62.34	60.96	24.72	73.63	71.21	56.43	46.12	26.09	51.43	12.86
a15	44.23	69.62	94.95	34.95	63.46	52.97	98.84	78.74	36.64	65.12	22.46	55.52	68.79
a16	19.10	45.49	65.63	64.96	50.57	55.91	10.02	34.70	29.31	50.15	70.68	62.57	71.09
a17	27.73	22.03	48.00	79.38	23.35	74.03	58.74	59.42	50.95	82.27	49.20	43.27	38.61
a18	41.46	33.83	7.97	75.11	49.00	55.70	64.99	38.47	49.86	17.45	28.08	35.21	67.81
a19	22.41	NA	34.86	49.30	65.18	39.84	81.16		55.99	66.55	55.38	43.08	29.72
a20	21.52	69.98	71.81	43.74	24.53	55.39	52.67	13.67	66.80	57.46	70.81	5.41	76.05
a21	56.90	48.80	31.66	15.31	40.57	58.14	70.19	67.23	61.10	31.04	60.72	22.39	70.38

Fig. 6.1 Browser view on random performance tableau instance

6.3 Random Cost-Benefit Performance Tableaux

125

The `randomPerfTabs` module provides a `RandomCBPerformanceTableau` 126 class for generating *Costs* versus *Benefits* organised performance tableaux. The 127 random generator is following by default the directives below: 128

- Three types of decision actions are distinguished: *cheap*, *neutral*, and *expensive* 129 ones with an equal proportion of 1/3. Two types of weighted criteria are 130 also distinguished: *Costs* criteria to be *minimised*, and *Benefits* criteria to be 131 *maximised*, in the proportions 1/3, respectively, 2/3; 132
- Random performances on each type of criteria are drawn, either from an ordinal 133 scale [0; 10], or from a cardinal scale [0.0; 100.0], following a parametric 134 triangular law of mode: 30% performance for cheap, 50% for neutral, and 70% 135 performance for expensive decision actions, with constant probability repartition 136 0.5 on each side of the respective mode; 137
- Costs criteria use mostly cardinal scales (3/4), whereas Benefits criteria use 138 mostly ordinal scales (2/3); 139
- The sum of weights of the Costs criteria equals by default the sum of weights of 140 the Benefits criteria: `weighDistribution = 'equiobjective'`; 141
- On cardinal criteria, both of cost or of benefit type, following constant perfor- 142 mance discrimination quantiles are instantiated: 5% indifferent situations, 90% 143 preference situations, and 5% considerable performance difference situations. 144

Parameters

145

- If `numberOfActions == None`, a uniform random number between 10 and 146 31 of *cheap*, *neutral* or *advantageous* actions (equal 1/3 probability each type) 147 actions is instantiated. Minimal number of decision actions required is 3; 148
- If `numberOfCriteria == None`, a uniform random number between 5 and 149 21 of cost or benefit criteria (1/3, respectively, 2/3 probability) is instantiated; 150
- `weightDistribution := 'equisignificant'` (default) | 'equi- 151 objectives' | 'fixed' | 'random'; 152
- default `weightScale` for 'random' weight distribution is 1 - `number- 153 OfCriteria`; 154
- All *cardinal* criteria are evaluated with decimals between 0.0 and 100.0 whereas 155 *ordinal* criteria are evaluated with integers between 0 and 10. 156
- `commonThresholds` is obsolete. Preference discrimination is specified as 157 *percentiles* of concerned performance differences (see below). 158
- `commonPercentiles := {'ind': 5, 'pref': 10, 'veto': 95}` are 159 expressed in percents (reversed for vetoes), and only concern cardinal criteria. 160
- `missingDataProbability := 0.0 ≤ float ≤ 1.0`; probability of missing 161 performance evaluation on a criterion for an alternative (default 0.025). 162
- `NA := Decimal` (default = -999); missing data symbol. 163

Example Python Session

Listing 6.2 Generating a random Cost-Benefit performance tableau

```

1  >>> from randomPerfTabs import\ 165
2  ...      RandomCBPerformanceTableau 166
3  >>> t = RandomCBPerformanceTableau( 167
4  ...      numberActions=7,\ 168
5  ...      numberCriteria=5,\ 169
6  ...      weightDistribution='equiobjectives',\ 170
7  ...      commonPercentiles={'ind':0.05,\ 171
8  ...                      'pref':0.10,\ 172
9  ...                      'veto':0.95},\ 173
10 ...      seed=100) 174
11 >>> t.showActions() 175
12 *----- show decision action -----* 176
13     key: a1 177
14         short name: a1c 178
15         name: random cheap decision action 179
16     key: a2 180
17         short name: a2n 181
18         name: random neutral decision action 182
19     ... 183
20     key: a7 184
21         short name: a7a 185
22         name: random advantageous decision action 186
23 >>> t.showCriteria() 187
24 *----- criteria -----* 188
25     b1 'random ordinal benefit criterion' 189
26         Preference direction: max 190
27         Scale = (0, 10) 191
28         Weight = 3 192
29     ... 193
30     c1 'random cardinal cost criterion' 194
31         Preference direction: min 195
32         Scale = (0.0, 100.0) 196
33         Weight = 2 197
34         Threshold ind : 1.76 + 0.00x ; percentile: 9.5 198
35         Threshold pref : 2.16 + 0.00x ; percentile: 14.3 199
36         Threshold veto : 73.19 + 0.00x ; percentile: 95.2 200
37     ...} 201

```

In Listing 6.2 one may notice the three types of decision actions (see Lines 12–22), as well as the two types of criteria with either an *ordinal* or a *cardinal* performance measuring scale (Lines 24–36). In the latter case, by default about 5% of the random performance differences will be below the *indifference* and 10% below the *preference* discriminating threshold. About 5% will be *considerably large*. More statistics about the generated performance evaluations can be inspected with the `showStatistics()` method.

```

1 >>> t.showStatistics()                                     209
2     *----- Performance tableau summary statistics -----* 210
3     Instance name      : randomCBperftab                211
4     Actions          : 7                                212
5     Criteria         : 5                                213
6     Criterion name   : b1                             214
7     Criterion weight : 3                                215
8     criterion scale  : 0.00 - 10.00                  216
9     mean evaluation   : 5.14                           217
10    standard deviation: 2.64                         218
11    maximal evaluation: 8.00                         219
12    quantile Q3 (x_75): 8.00                         220
13    median evaluation: 6.50                           221
14    quantile Q1 (x_25): 3.50                           222
15    minimal evaluation: 1.00                         223
16    mean absolute difference: 2.94                  224
17    standard difference deviation: 3.74            225
18    ...
19     Criterion name   : c1                           227
20     Criterion weight : 2                                228
21     criterion scale  : -100.00 - 0.00                229
22     mean evaluation   : -49.32                         230
23     standard deviation: 27.59                         231
24     maximal evaluation: 0.00                           232
25     quantile Q3 (x_75): -27.51                         233
26     median evaluation: -35.98                         234
27     quantile Q1 (x_25): -54.02                         235
28     minimal evaluation: -91.87                         236
29     mean absolute difference: 28.72                  237
30     standard difference deviation: 39.02            238
31     ...

```

A heatmap view with five colour levels gives the result shown in Fig. 6.2 on the following page.

```

1 >>> t.showHTMLPerformanceHeatmap(colorLevels=5, \ 242
2     rankingRule=None, \ 243
3     pageTitle='Random Cost-Benefit Performance Tableau') 244

```

Such a performance tableau may be stored with the `save()` method and reloaded with the `PerformanceTableau` class:

```

1 >>> t.save('temp')                                     247
2     *----- saving performance tableau in XMCDA 2.0 format 248
3     -----*                                              249
4     File: temp.py saved !                            250
5 >>> from perfTabs import PerformanceTableau        251
5 >>> t = PerformanceTableau('temp')                  252

```



Fig. 6.2 Unordered heatmap of a random Cost-Benefit performance tableau

6.4 Random Three Objectives Performance Tableaux

253

The randomPerfTabs module provides a Random3ObjectivesPerformanceTableau class for generating random tableaux concerning public policies evaluated with respect to three decision objectives taking, respectively, into account *economical, societal* as well as *environmental* aspects. Each potential public policy is qualified randomly as performing *weakly* (–), *fairly* (~) or *well* (+) with respect to each one of the three objectives.

Generator Directives Are the Following

- `numberOfActions = 20` (default), minimal number required is 3;
- `numberOfCriteria = 13` (default),
- `weightDistribution = 'equiobjectives'` (default) | 'random' | 'equisignificant',
- `weightScale = (1,numberOfCriteria)`: only used when random criterion weights are requested,
- `integerWeights = True` (default): `False` gives normalised rational weights,
- `commonScale = (0.0,100.0)`,
- `commonThresholds = [(5.0, 0.0), (10.0, 0.0), (60.0, 0.0)]`: Performance discrimination thresholds may be set for 'ind', 'pref' and 'veto' thresholds,
- `commonMode = ['triangular','variable',0.5]`: random number generators of various other types ('uniform', 'beta') are available. If the mode of the 'triangular' distribution is set to 'variable', three modes at 0.3(–), 0.5(~), respectively, 0.7(+) of the common scale span are set at random for each coalition and action.
- `valueDigits = 2` (default): evaluations are encoded as decimals,

- `missingDataProbability` = 0.05 (default): random insertion of missing values with given probability, 278
279
 - `NA` := `Decimal` (default = `Decimal (' -999 ')`); missing data symbol, 280
 - `seed` = `None` (default). 281

Example Python Session

282

Listing 6.3 Generating a random three objectives performance tableau

```
1 >>> from randomPerfTabs import\  
2 ...             Random3ObjectivesPerformanceTableau  
3 >>> t = Random3ObjectivesPerformanceTableau()  
4 ...             numberOfActions=7, \  
5 ...             numberOfCriteria=13, \  
6 ...             weightDistribution='equiobjectives', \  
7 ...             seed=120)  
8 >>> t  
9 *----- PerformanceTableau instance description -----*  
10 Instance class      : Random3ObjectivesPerformanceTableau  
11 Seed                : 120  
12 Instance name       : random3ObjectivesPerfTab  
13 Actions              : 7  
14 Objectives           : 3  
15 Criteria             : 13  
16 NaN proportion (%) : 2.2  
17 Attributes           : ['name', 'valueDigits', 'BigData',  
18     'OrdinalScales', 'missingDataProbability',  
19     'negativeWeightProbability', 'randomSeed',  
20     'sumWeights', 'valuationPrecision', 'commonScale',  
21     'objectiveSupportingTypes', 'actions', 'objectives',  
22     'criteriaWeightMode', 'criteria',  
23     'weightPreorder', 'evaluation', 'NA']  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305
```

The default missing data probability is 5%. In our random instance here we observe 2.2% of missing evaluations (see Line 16). The 'equiobjectives' directive (see Line 6) results hence in a balanced total weight (72.00) for each decision objective as is made apparent below with the `showObjectives()` method.

Listing 6.4 Inspecting the three objectives

```
1 >>> t.showObjectives()  
2     *----- show objectives -----"  
3     Eco: Economical aspect  
4         ec01 criterion of objective Eco 18  
5         ec05 criterion of objective Eco 18  
6         ec06 criterion of objective Eco 18  
7         ec12 criterion of objective Eco 18  
8     Total weight: 72.00 (4 criteria)  
9     Soc: Societal aspect  
10        so02 criterion of objective Soc 24  
11        so11 criterion of objective Soc 24  
12        so13 criterion of objective Soc 24
```

```

13  Total weight: 72.00 (3 criteria) 323
14  Env: Environmental aspect 324
15    en03 criterion of objective Env 12 325
16    en04 criterion of objective Env 12 326
17    en07 criterion of objective Env 12 327
18    en08 criterion of objective Env 12 328
19    en09 criterion of objective Env 12 329
20    en10 criterion of objective Env 12 330
21  Total weight: 72.00 (6 criteria) 331

```

In Listing 6.4 on the previous page, we notice that four *equisignificant* criteria (ec01, ec05, ec06, and ec12) of significance weight 18 assess, for instance, the performance of the public policies from an *economic* point of view (Lines 3–8). Three *equisignificant* criteria of weight 24 do the same from a *societal* (Lines 9–13), and six of weight 12 from an *environmental* point of view (Lines 14–21).

Variable *triangular* modes: 0.3, 0.5, or 0.7 of the span of the measure scale, give a different performance status for each public policy with respect to the three decision objectives.

```

1 >>> t.showActions() 340
2   key: p1 341
3     short name: p1 342
4     name: action p1 Eco- Soc+ Env+ 343
5     profile: {'Eco': 'weak', 'Soc': 'good', 'Env': 'good'} 344
6   key: p2 345
7     short name: p2 346
8     name: action p2 Eco- Soc- Env- 347
9     profile: {'Eco': 'weak', 'Soc': 'weak', 'Env': 'weak'} 348
10   ...
11   key: p7 350
12     short name: p7 351
13     name: action p7 Eco+ Soc- Env- 352
14     profile: {'Eco': 'good', 'Soc': 'weak', 'Env': 'weak'} 353

```

Policy p1, for instance, will probably show *good* performances with respect to the *societal* and *environmental* aspects, and *weak* performances with respect to the *economic* aspect, whereas policy p2, being weak with respect to all the three objectives, will probably appear among the last recommended policies.

We may inspect in Fig. 6.3 on the facing page the given random three-objectives performance tableau with the `showHTMLPerformanceTableau()` method.

```

1 >>> t.showHTMLPerformanceTableau() 360

```

Light green cells show the highest and light red the lowest evaluations. Policy p1 obtains thus the lowest evaluation (17.09/100.00) on the environmental criterion en09, whereas policy p3 obtains the highest evaluations on four out of the six *environmental* criteria.

No trivial best choice becomes apparent when looking at the performance tableau shown in Fig. 6.3 on the next page. Let us therefore compute a RUBIS best choice recommendation (see Chap. 4).

Performance table random3ObjectivesPerfTab

criteria	ec01	ec05	ec06	ec12	en03	en04	en07	en08	en09	en10	so02	so11	so13
weight	18.00	18.00	18.00	18.00	12.00	12.00	12.00	12.00	12.00	12.00	24.00	24.00	24.00
p1	31.05	26.95	35.91	32.85	68.88	70.52	22.94	50.04	17.06	32.52	20.90	NA	62.37
p2	12.54	11.31	29.63	9.77	43.09	36.04	32.96	54.35	82.25	22.29	75.76	51.42	27.96
p3	30.70	58.45	38.31	24.89	72.36	87.02	30.57	80.17	90.38	86.60	54.51	43.21	6.58
p4	16.18	61.70	60.27	15.10	26.63	66.28	12.84	58.78	86.68	57.92	NA	71.91	70.18
p5	86.00	89.43	44.41	18.32	31.10	74.55	29.89	40.26	26.86	41.91	23.83	57.41	22.41
p6	77.65	94.24	24.75	65.64	3.85	36.15	47.36	81.22	72.54	80.54	27.25	37.04	34.11
p7	72.26	44.62	70.55	29.82	52.69	37.80	50.06	28.27	28.53	29.50	29.81	41.38	26.31

Fig. 6.3 Browser view on the given random three-objectives performance tableau

Listing 6.5 What is the public policy to recommend as best choice?

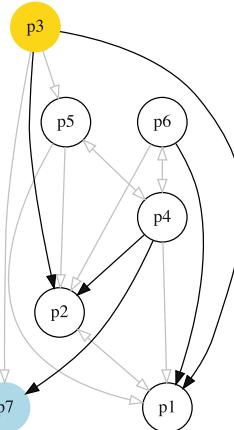
```

1 >>> from outrankingDigraphs import\          368
2 ...          BipolarOutrankingDigraph          369
3 >>> g = BipolarOutrankingDigraph(t)          370
4 >>> g.showBestChoiceRecommendation()          371
5 ****
6 Rubis best choice recommendation(s) (BCR)          372
7 (in decreasing order of determinateness)
8 Credibility domain: [-1.00,1.00]          373
9 === >> potential first choice(s)          374
10 * choice          : ['p3', 'p4', 'p5', 'p6']          375
11 independence          : 0.00          376
12 dominance          : 0.17          377
13 absorbency          : -1.00          378
14 covering (%)          : 41.67          379
15 determinateness (%) : 52.98          380
16 - most credible action(s) = { 'p3': 0.17, }          381
17 === >> potential last choice(s)          382
18 * choice          : ['p1', 'p2', 'p5', 'p7']          383
19 independence          : 0.00          384
20 dominance          : -0.44          385
21 absorbency          : 0.19          386
22 covered (%)          : 41.67          387
23 determinateness (%) : 50.79          388
24 - most credible action(s) = { 'p7': 0.06, }          389

```

Policy p3 gives the most credible first choice recommendation with the support of a 57.5% majority of significance, whereas policy p7 gives a credible last choice recommendation.

Policy p5 represents an ambiguous first as well as last choice candidate. A drawing of the strict outranking digraph oriented by first and last recommendations shows in Fig. 6.4 on the following page the complete preferential picture.



Digraph3 (graphviz), R. Bisdorff, 2020

Fig. 6.4 The strict outranking digraph oriented by first and last choice recommendations. Policy p_5 appears incomparable—in a strict outranking sense—to all the other six policies, whereas policies p_1 , p_2 , and p_7 appear strictly outranked

```

1  >>> (~(-g)).exportGraphViz (\ 398
2  ...           fileName='3ObjPerfTabBestChoice', \ 399
3  ...           firstChoice=['p3'], lastChoice=['p7']) 400
4  *----- exporting a dot file for GraphViz tools -----* 401
5  Exporting to 3ObjPerfTabBestChoice.dot 402
6  dot -Grankdir=BT -Tpng 3ObjPerfTabBestChoice.dot\ 403
7  ...           -o 3ObjPerfTabBestChoice.png 404

```

A heatmap view in Fig. 6.5 on the next page on the COPELAND ranked 405 performance tableau confirms the best choice recommendation (see Sect. 8.2). 406

```

1  >>> t.showHTMLPerformanceHeatmap (\ 407
2  ...           Correlations=True, \ 408
3  ...           colorLevels=5, ndigits=1, \ 409
4  ...           rankingRule='Copeland') 410

```

The COPELAND ranking, based on the bipolar outranking digraph, confirms 411 policy p_3 as first-ranked. Notice also that the three strict outranked policies do 412 effectively appear in the last positions. 413

Heatmap of Performance Tableau 'random3ObjectivesPerfTab'

criteria	en10	ec05	en09	en04	en08	ec01	ec12	ec06	so11	so02	en07	so13	en03
weights	+12.00	+18.00	+12.00	+12.00	+12.00	+18.00	+18.00	+18.00	+24.00	+24.00	+12.00	+24.00	+12.00
tau ^(*)	+0.79	+0.52	+0.45	+0.40	+0.38	+0.14	+0.12	+0.10	+0.02	-0.02	-0.10	-0.10	-0.17
p3	86.6	58.5	90.4	87.0	80.2	30.7	24.9	38.3	43.2	54.5	30.6	6.6	72.4
p4	57.9	61.7	86.7	66.3	58.8	16.2	15.1	60.3	71.9	NA	12.8	70.2	26.6
p6	80.5	94.2	72.5	36.1	81.2	77.7	65.6	24.8	37.0	27.2	47.4	34.1	3.9
p5	41.9	89.4	26.9	74.5	40.3	86.0	18.3	44.4	57.4	23.8	29.9	22.4	31.1
p7	29.5	44.6	28.5	37.8	28.3	72.3	29.8	70.5	41.4	29.8	50.1	26.3	52.7
p1	32.5	26.9	17.1	70.5	50.0	31.1	32.9	35.9	NA	20.9	22.9	62.4	68.9
p2	22.3	11.3	82.2	36.0	54.4	12.5	9.8	29.6	51.4	75.8	33.0	28.0	43.1

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%
----------	--------	--------	--------	--------	---------

(*) tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation

Outranking model: standard, Ranking rule: Copeland

Ordinal (Kendall) correlation between global ranking and global outranking relation: +0.896

Mean marginal correlation (a) : +0.161

Standard marginal correlation deviation (b) : +0.258

Ranking fairness (a) - (b) : -0.098

Fig. 6.5 Browser view on the COPELAND ranked performance tableau

6.5 Random Academic Performance Tableaux

414

The RandomAcademicPerformanceTableau class generates performance tableaux with random grades for a given number of students in different courses. 415
416

Generator Directives

- `numberOfStudents` := Integer (default 10), 418
- `numberOfCourses` := Integer (default 5), 419
- `weightDistribution` := 'equisignificant' | 'random' (default), 420
- `weightScale` := 1, 1 - `numberOfCourses` (default when random), 421
- `IntegerWeights` := Boolean (True = default), 422
- `commonScale` := (Integer,integer) (0, 20) (default), 423
- `ndigits` := Integer (default 0), 424
- `WithTypes` := Boolean (default False), 425
- `commonMode` := ('triangular',*xm*=14,*r*=0.25) (default), 426
- `commonThresholds` := 'ind':(0,0), 'pref':(1,0) (default), 427
- `missingDataProbability` := 0.0 (default), 428
- `NA` := Decimal (default = -999); missing data symbol. 429

When Parameter `WithTypes` is set to True, the students are randomly 430 allocated to one of the following four categories—weak (1/6), fair 431 (1/3), good (1/3), and excellent (1/3)—in the bracketed proportions. In a default 0–20 432 grading range, the random range of a weak student is 0–10, of a fair student 4– 433 16, of a good student 8–20, and of an excellent student 12–20. The random grading 434 generator follows in this case a double triangular probability law with *mode* (*xm*) 435 equal to the middle of the random range and median repartition (*r* = 0.5) of 436 probability each side of the mode. 437

Listing 6.6 Generating a random academic performance tableau

```

1 >>> from randomPerfTabs import RandomAcademicPerformanceTableau 438
2 >>> t = RandomAcademicPerformanceTableau(\ 439
3 ...     numberofStudents=11,\ 440
4 ...     numberofCourses=7, missingDataProbability=0.03,\ 441
5 ...     WithTypes=True, seed=100) 442
6 >>> t 443
7 *----- PerformanceTableau instance description -----* 444
8 Instance class : RandomAcademicPerformanceTableau 445
9 Seed : 100 446
10 Instance name : randstudPerf 447
11 Actions : 11 448
12 Criteria : 7 449
13 NA proportion(%) : 5.2 450
14 Attributes : ['randomSeed', 'name', 'actions', 451
15     'criteria', 'evaluation', 'weightPreorder'] 452
16 >>> t.showPerformanceTableau() 453
17 *---- performance tableau -----* 454
18 Courses | 'm1' 'm2' 'm3' 'm4' 'm5' 'm6' 'm7' 455
19 ECTS | 2 1 3 4 1 1 5 456
20 -----|----- 457
21 's01f' | 12 13 15 08 16 06 15 458
22 's02g' | 10 15 20 11 14 15 18 459
23 's03g' | 14 12 19 11 15 13 11 460
24 's04f' | 13 15 12 13 13 10 06 461
25 's05e' | 12 14 13 16 15 12 16 462
26 's06g' | 17 13 10 14 NA 15 13 463
27 's07e' | 12 12 12 18 NA 13 17 464
28 's08f' | 14 12 09 13 13 15 12 465
29 's09g' | 19 14 15 13 09 13 16 466
30 's10g' | 10 12 14 17 12 16 09 467
31 's11w' | 10 10 NA 10 10 NA 08 468
32 >>> t.weightPreorder 469
33 ['m2', 'm5', 'm6'], ['m1'], ['m3'], ['m4'], ['m7']] 470

```

The example random tableau, generated, for instance, above with missing DataProbability = 0.03, WithTypes = True and seed = 100 (see Listing 6.6 Lines 2–5), results in a set of two excellent (s05e and s07e), five good (s02g, s03g, s06g, s09g and s10g), three fair (s01f, s04f and s08f) and one weak (s11w) student. Notice that six students get a grade below the course validating threshold of 10 and we observe four missing grades (NA), two in course m5 and, one in courses m3 and m6 (see Lines 21–31).

A statistical summary of the students' grades obtained in the highest weighted course, namely m7, is shown with the showCourseStatistics() method.

Listing 6.7 Student performance summary statistics per course

```

1 >>> t.showCourseStatistics('m7') 480
2 *---- Summary performance statistics -----* 481
3 Course name : m7 482
4 Course weight : 5 483
5 Students : 11 484

```

6	Grading scale	:	0.00	-	20.00	485
7	Missing evaluations	:	0			486
8	Mean evaluation	:	12.82			487
9	Standard deviation	:	3.79			488
10	Maximal evaluation	:	18.00			489
11	Quantile Q3 (x_75)	:	16.25			490
12	Median evaluation	:	14.00			491
13	Quantile Q1 (x_25)	:	10.50			492
14	Minimal evaluation	:	6.00			493
15	Mean absolute difference	:	4.30			494
16	Standard difference deviation	:	5.35			495

In Listing 6.7 on the preceding page, the Course m7 evaluation statistics show a mean grade of 12.82 and a median grade of 14. Maximal (respectively, minimal) grade is 18 (respectively, 6).

With a ranked heatmap view on all the grades we now get in Fig. 6.6 a global pictures of the performance of all the 11 students. The ranking shown here, produced with the NETFLOWS ranking rule (see Sect. 8.3), reports a high correlation of +0.887 with the corresponding bipolar-valued outranking digraph (see Chap. 16).

The NETFLOWS ranking represents also a rather *fair weighted consensus* between the individual courses' marginal rankings as is made apparent with the showRankingConsensusQuality() method in Listing 6.8 on the next page.

criteria	m7	m4	m2	m3	m1	m6	m5
weights	+5.00	+4.00	+1.00	+3.00	+2.00	+1.00	+1.00
tau(*)	+0.73	+0.31	+0.29	+0.20	+0.11	+0.09	+0.07
s07e	17.00	18.00	12.00	12.00	12.00	13.00	NA
s02g	18.00	11.00	15.00	20.00	10.00	15.00	14.00
s09g	16.00	13.00	14.00	15.00	19.00	13.00	9.00
s05e	16.00	16.00	14.00	13.00	12.00	12.00	15.00
s06g	13.00	14.00	13.00	10.00	17.00	15.00	NA
s03g	11.00	11.00	12.00	19.00	14.00	13.00	15.00
s10g	9.00	17.00	12.00	14.00	10.00	16.00	12.00
s01f	15.00	8.00	13.00	15.00	12.00	6.00	16.00
s08f	12.00	13.00	12.00	9.00	14.00	15.00	13.00
s04f	6.00	13.00	15.00	12.00	13.00	10.00	13.00
s11w	8.00	10.00	10.00	NA	10.00	NA	10.00

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%
----------	--------	--------	--------	--------	---------

(*) tau: *Ordinal (Kendall) correlation between marginal criterion and global ranking relation*

Outranking model: **standard**, Ranking rule: **NetFlows**

Ordinal (Kendall) correlation between global ranking and global outranking relation: **+0.887**

Fig. 6.6 Ranking the students in a performance heatmap view

Listing 6.8 Consensus quality of the students' ranking

```

1 >>> from outrankingDigraphs import\
2 ...             BipolarOutrankingDigraph
3 >>> g = BipolarOutrankingDigraph(t)
4 >>> t.showRankingConsensusQuality(\
5 ...                 g.computeNetFlowsRanking())
6     Consensus quality of ranking: 506
7     ['s07', 's02', 's09', 's05', 's06', 's03', 's10', 507
8     's01', 's08', 's04', 's11'] 508
9     Criterion (weight): correlation 509
10    -----
11     'm7' (5) : +0.727 510
12     'm4' (4) : +0.309 511
13     'm2' (1) : +0.291 512
14     'm3' (3) : +0.200 513
15     'm1' (2) : +0.109 514
16     'm6' (1) : +0.091 515
17     'm5' (1) : +0.073 516
18 Summary: 517
19     Weighted mean marginal correlation (a) : +0.361 518
20     Standard deviation (b) : +0.248 519
21     Ranking fairness (a) - (b) : +0.113 520

```

The correlation with the marginal course rankings follows, except for course m2, the order of the given ECTS weights. Course m7, with the highest relative ECTS (5), is most present (+0.727) in the global NETFLOWS ranking and no marginal course ranking appears negatively correlated with this global ranking. The mean weighted correlation is +0.361.

Useful multiple-criteria ranking rules are presented and discussed in detail in Chap. 8. The next Chap. 7 is concerned with yet another kind of performance evaluation models which is discussed in social choice theory, namely *linear voting profiles*.

Reference

Bisdorff R (2020) Lecture 3: continuous random variables. In: Lectures of the computational statistics course, University of Luxembourg. <http://hdl.handle.net/10993/37870>

AUTHOR QUERIES

- AQ1. Please check the sentence “Deciding which students, based...” for clarity.
- AQ2. Missing citation for Fig. 6.1 was inserted here. Please check if appropriate.

Uncorrected Proof

Chapter 7

Who Wins the Election?

1

2

Contents

7.1	Linear Voting Profiles	83	4
7.2	Computing the Winner	85	5
7.3	The Majority Margins Digraph	86	6
7.4	Cyclic Social Preferences	88	7
7.5	On Generating Realistic Random Linear Voting Profiles	91	8

Abstract This chapter is more specifically devoted to handling linear voting profiles and computing the winner of such election results. By following CONDORCET’s recipe, we consider pairwise comparisons of election candidates and balance the number of times the first beats the second against the number of times the second beats the first. Thus we obtain the majority margins digraph, in fact a bipolar-valued digraph. When the voters express contradictory linear voting profiles one naturally observes cyclic social preferences without seeing any paradox in this situation. Finally we present a more politically realistic generator for random linear voting profiles which takes into account pre-election polls.

7.1 Linear Voting Profiles

18

The votingProfiles module provides resources like the LinearVoting-Profile class for handling election results (Bisdorff 2020a). To illustrates these resources let us consider elections involving a set of eligible candidates and a set of weighted voters, who express their voting preferences in a complete linear ranking (without ties) of the eligible candidates. The module provides a RandomLinearVotingProfile class for generating random instances of such LinearVotingProfile type. In the interactive Python session shown in Listing 7.1, a random linear voting profile is generated, for instance, for the election of three candidates by five voters:

Listing 7.1 Example of random linear voting profile

```

1  >>> from votingProfiles import\           28
2  ...                               RandomLinearVotingProfile\ 29
3  >>> lvp = RandomLinearVotingProfile(numberOfVoters=5,\ 30
4  ...                               numberOfCandidates=3,\ 31
5  ...                               RandomWeights=True)\ 32
6  >>> lvp.candidates\           33
7  OrderedDict([('c1', {'name': 'Candidate 1'}),\ 34
8  ('c2', {'name': 'Candidate 2'}),\ 35
9  ('c3', {'name': 'Candidate 3'}))\ 36
10 >>> lvp.voters\           37
11 OrderedDict([('v1', {'weight': 2}),\ 38
12 ('v2', {'weight': 3}),\ 39
13 ('v3', {'weight': 1}),\ 40
14 ('v4', {'weight': 5}),\ 41
15 ('v5', {'weight': 4}))\ 42
16 >>> lvp.linearBallot\           43
17 {'v1': ['c2', 'c1', 'c3'],\ 44
18 'v2': ['c3', 'c1', 'c2'],\ 45
19 'v3': ['c1', 'c3', 'c2'],\ 46
20 'v4': ['c1', 'c2', 'c3'],\ 47
21 'v5': ['c3', 'c1', 'c2']}\ 48

```

The `LinearVotingProfile` data concerning the eligible candidates and the voters is internally stored in two ordered dictionaries: attribute `candidates` (Line 6) and attribute `voters` (Line 10). Notice that in this random example, the five voters are weighted. The linear voting ballots are stored in a standard dictionary: attribute `linearBallot` (Line 16). These ballots can be inspected with the `showLinearBallots()` method.

Listing 7.2 Showing linear voting profiles

```

1  >>> lvp.showLinearBallots()\           55
2  voters(weight)      candidates rankings\ 56
3  v1(2):            ['c2', 'c1', 'c3']\ 57
4  v2(3):            ['c3', 'c1', 'c2']\ 58
5  v3(1):            ['c1', 'c3', 'c2']\ 59
6  v4(5):            ['c1', 'c2', 'c3']\ 60
7  v5(4):            ['c3', 'c1', 'c2']\ 61
8  nbr. of voters: 15\ 62

```

Editing of this linear voting profile may be done by saving first the data into a file, then edit this file, and reload it again.

```

1  >>> lvp.save(fileName='tutorialLinearVotingProfile1')\ 65
2  --- Saving linear profile in file:\ 66
3  <tutorialLinearVotingProfile1.py> ---*\ 67
4  >>> from votingProfiles import LinearVotingProfile\ 68
5  >>> lpv = LinearVotingProfile('tutorialLinearVotingProfile1')\ 69

```

7.2 Computing the Winner

70

The `computeUninominalVotes()` and the `computeSimpleMajority-
Winner()` methods compute *uninominal votes*, i.e. how many times a candidate
was ranked first, and who is consequently the *simple majority* (plurality) winner(s)
in this election.

```

1 >>> lvp.computeUninominalVotes()
2   {'c1': 6, 'c2': 2, 'c3': 7}
3 >>> lvp.computeSimpleMajorityWinner()
4   ['c3']
```

As we observe no absolute majority (8/15) of votes for any one of the
three candidates, one may look, with the `computeInstantRunoffWinner()`
method, for the *instant runoff* winner instead (Bisdorff 2020a).

Listing 7.3 Example instant run off winner

```

1 >>> lvp.computeInstantRunoffWinner(Comments=True)
2   Half of the Votes = 7.50
3   ==> stage = 1
4     remaining candidates ['c1', 'c2', 'c3']
5     uninominal votes {'c1': 6, 'c2': 2, 'c3': 7}
6     minimal number of votes = 2
7     maximal number of votes = 7
8     candidate to remove = c2
9     remaining candidates = ['c1', 'c3']
10  ==> stage = 2
11    remaining candidates ['c1', 'c3']
12    uninominal votes {'c1': 8, 'c3': 7}
13    minimal number of votes = 7
14    maximal number of votes = 8
15    candidate a1 obtains an absolute majority
16  Instant run off winner: ['c1']
```

In Listing 7.3 no candidate obtains at stage 1 an absolute majority of votes.
Candidate c_2 obtains the minimal number of votes (2/15) and is, hence, eliminated.
At stage 2, candidate c_1 eventually obtains an absolute majority of the votes (8/15)
and is hence elected.

One may also follow the *Chevalier de Borda*'s advice and, after a *rank analysis* of
the linear ballots, compute the *BORDA score*—the average rank—of each candidate
and hence determine the *BORDA winner(s)* (de Borda 1781).

Listing 7.4 Example of BORDA rank scores

```

1 >>> lvp.computeRankAnalysis()
2   {'c1': [6, 9, 0], 'c2': [2, 5, 8], 'c3': [7, 1, 7]}
3 >>> v.computeBordaScores()
4   OrderedDict([
5     ('c1', {'BordaScore': 24, 'averageBordaScore': 1.6}),
6     ('c3', {'BordaScore': 30, 'averageBordaScore': 2.0}),
7     ('c2', {'BordaScore': 36, 'averageBordaScore': 2.4})])
8 >>> lvp.computeBordaWinners()
9   ['c1']
```

In Listing 7.4, Candidate c_1 obtains the minimal BORDA score, followed by candidate c_3 and finally candidate c_2 . The corresponding BORDA *rank analysis table* may be printed out with the `showRankAnalysisTable()` method. 114
115
116

Listing 7.5 Rank analysis example with BORDA scores

```

1 >>> lvp.showRankAnalysisTable() 117
2 *---- Borda rank analysis tableau ----* 118
3 candi- | alternative-to-rank | Borda 119
4 dates | 1 2 3 | score average 120
5 -----|----- 121
6 'c1' | 6 9 0 | 24/15 1.60 122
7 'c3' | 7 1 7 | 30/15 2.00 123
8 'c2' | 2 5 8 | 36/15 2.40 124

```

In our randomly generated election results, we are lucky: Candidate c_1 is both the instant runoff winner and the BORDA winner (see Listings 7.3 on the previous page and 7.4). However, one could also follow the *Marquis de Condorcet*'s advice, and compute the *majority margins* obtained by voting for each individual pair of candidates (de Caritat, Marquis de Condorcet 1784). 125
126
127
128
129

7.3 The Majority Margins Digraph

Candidate c_1 , for instance, is ranked four times before and once behind candidate c_2 (see Listing 7.2 on page 84). Hence, due to the voters' weights, the corresponding *majority margin* $M(c_1, c_2)$ amounts to $(3 + 1 + 5 + 4) - (2) = +11$. These pairwise *majority margins* define on the set of candidates what we call a *majority margins digraph*. The `MajorityMarginsDigraph` class is handling such kind of digraphs. 131
132
133
134
135
136

Listing 7.6 Example of *Majority Margins* digraph

```

1 >>> from votingProfiles import MajorityMarginsDigraph 137
2 >>> mmdg = MajorityMarginsDigraph(lvp,\ 138
3 ... 139
4 >>> mmdg 140
5 *---- Digraph instance description ----* 141
6 Instance class : MajorityMarginsDigraph 142
7 Instance name : rel_randomLinearVotingProfile1 143
8 Digraph Order : 3 144
9 Digraph Size : 3 145
10 Valuation domain : [-15.00;15.00] 146
11 Determinateness (%) : 55.56 147
12 Attributes : ['name', 'actions', 'voters', 148
13 'ballot', 'valuationdomain', 149
14 'relation', 'order', 150
15 'gamma', 'notGamma'] 151
16 >>> mmdg.showAll() 152
17 *---- show detail ----* 153

```

```

18 Digraph : rel_randLinearVotingProfile1
19     *---- Candidates ----*
20     ['c1', 'c2', 'c3']
21     *---- Characteristic valuation domain ----*
22     {'max': Decimal('15.0'),
23      'med': Decimal('0'),
24      'min': Decimal('-15.0'),
25      'hasIntegerValuation': True}
26     *---- majority margins ----*
27     M(x,y) | 'c1' 'c2' 'c3'
28     -----|-----
29     'c1' | 0 11 1
30     'c2' | -11 0 -1
31     'c3' | -1 1 0
32     Valuation domain: [-15;+15]

```

Notice in Listing 7.6 Lines 29–31 that, in the case of linear voting profiles, 169 majority margins always verify a *zero-sum property*: $M(x, y) + M(y, x) = 0$ for 170 all candidates x and y . This is not true in general for arbitrary voting profiles. 171 The *majority margins* digraph of linear voting profiles defines in fact a *weak* 172 *tournament* and belongs, hence, to the class of *self-codual*¹ bipolar-valued digraphs 173 (see Sect. 2.6). 174

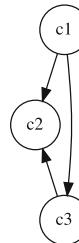
A candidate x , showing a positive majority margin $M(x, y)$, is beating candidate 175
 y with an absolute majority in a pairwise voting. Hence, a candidate showing only 176
positive terms in their row in the *majority margins* digraph relation table, beats 177
all other candidates with absolute majority of votes. CONDORCET recommends 178
to declare this candidate (is always unique, why?) the winner of the election. 179
Here we are lucky, it is again candidate `c1` who is the CONDORCET winner (see 180
Line 29 in Listing 7.6 on the preceding page). This result is confirmed below by the 181
`computeCondorcetWinners()` method. 182

```
1 >>> mmdg.computeCondorcetWinners()
2 ['c1']
```

By seeing the majority margins like a *bipolar-valued characteristic function* of a global voters' preference relation defined on the set of eligible candidates, we may reuse all operational resources of the generic Digraph class,² and especially its `exportGraphViz()` method for visualising in Fig. 7.1 on the following page the election result.

¹ The class of self-codual bipolar-valued digraphs consists of all weakly asymmetric digraphs, i.e. digraphs containing only asymmetric and/or indeterminate links. Limit cases consist of, on the one side, full tournaments with indeterminate reflexive links, and, on the other side, fully indeterminate digraphs. In this class, the converse (inverse \sim) operator is indeed identical to the dual (negation \neg) one.

² See Chan 2.



Digraph3 (graphviz), R. Bisendorff, 2020

Fig. 7.1 Visualising an election result. In the Figure we notice that the *majority margins* digraph from our example linear voting profile models in fact a linear ranking of the candidates: $c_1 > c_3 > c_2$, the same actually as modelled by the BORDA scores (see Listing 7.4 on page 85)

```

1  >>> mmdg.exportGraphViz(\          190
2      ...                     fileName='tutorialLinearBallots',\ 191
3      ...                     graphType='pdf') 192
4  *---- exporting a dot file for GraphViz tools -----* 193
5  Exporting to tutorialLinearBallots.dot 194
6  dot -Grankdir=BT -Tpng tutorialLinearBallots.dot \ 195
    -o tutorialLinearBallots.pdf 196
  
```

That a majority margins digraph models a linear ranking of the eligible candidates, as shown in Fig. 7.1, represents a very unlikely event. Usually, when aggregating linear ballots, there appear cyclic social preferences. 197
198
199

7.4 Cyclic Social Preferences

200

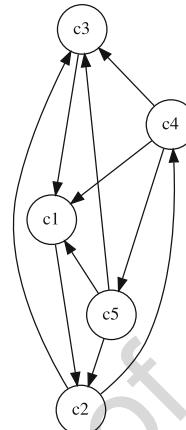
Let us consider, for instance, in Listing 7.7 the following linear voting profile v and 201
construct the corresponding majority margins digraph. 202

Listing 7.7 Example of cyclic social preferences

```

1  >>> v.showLinearBallots()          203
2  voters(weight): candidates rankings
3      v1(1): ['c1', 'c3', 'c5', 'c2', 'c4'] 204
4      v2(1): ['c1', 'c2', 'c4', 'c3', 'c5'] 205
5      v3(1): ['c5', 'c2', 'c4', 'c3', 'c1'] 206
6      v4(1): ['c3', 'c4', 'c1', 'c5', 'c2'] 207
7      v5(1): ['c4', 'c2', 'c3', 'c5', 'c1'] 208
8      v6(1): ['c2', 'c4', 'c5', 'c1', 'c3'] 209
9      v7(1): ['c5', 'c4', 'c3', 'c1', 'c2'] 210
10     v8(1): ['c2', 'c4', 'c5', 'c1', 'c3'] 211
11     v9(1): ['c5', 'c3', 'c4', 'c1', 'c2'] 212
12  >>> mmdg = MajorityMarginsDigraph(v) 213
13  >>> mmdg.showRelationTable(ReflexiveTerms=False) 214
14  * ---- Relation Table -----* 215
  
```

Fig. 7.2 Cyclic social preferences. $c_1 > c_2 > c_3 > c_1$, for instance



Digraph3 (graphviz), R. Bisdorff, 2020

15	$M(x, y)$	'c1'	'c2'	'c3'	'c4'	'c5'	217
16	-----	-----	-----	-----	-----	218	
17	'c1'	-	1	-1	-5	-3	219
18	'c2'	-1	-	1	1	-1	220
19	'c3'	1	-1	-	-3	-1	221
20	'c4'	5	-1	3	-	1	222
21	'c5'	3	1	1	-1	-	223
22	Valuation domain:	[-9; +9]					224

Now there does not exist anymore a completely positive row in the relation table (see Lines 17–21 in Listing 7.7). No one of the five candidates is beating all the others with an absolute majority of votes. There is no CONDORCET winner anymore. In fact, when looking in Fig. 7.2 at a graphviz drawing of this *majority margins* digraph, we may observe *cyclic* voters' preferences, like $c_1 > c_2 > c_3 > c_1$.

```

1 >>> mmdg.exportGraphViz('cycles',graphType='pdf')
2 ----- exporting a dot file for GraphViz tools -----
3 Exporting to cycles.dot
4 dot -Grankdir=BT -Tpng cycles.dot -o cycles.pdf

```

There may be many circuits appearing in a *majority margins* digraph. The computeChordlessCircuits method detects and enumerates all minimal chordless circuits in a Digraph instance.

```

1 >>> mmdg.computeChordlessCircuits()
2 [[['c2', 'c3', 'c1'], frozenset({'c2', 'c3', 'c1'})],
3  [['c2', 'c4', 'c5'], frozenset({'c2', 'c5', 'c4'})],
4  [['c2', 'c4', 'c1'], frozenset({'c2', 'c1', 'c4'})]]

```

In our example voting profile v , we actually obtain three chordless social preference circuits and determining the winner of this election result becomes non-trivial. There is, for instance, no more any instant runoff winner.

```

1 >>> v.computeInstantRunoffWinner(Comments=True) 244
2 Total number of votes = 9.000 245
3 Half of the Votes = 4.50 246
4 ==> stage = 1 247
5     remaining candidates ['c2', 'c5', 'c1', 'c4', 'c3'] 248
6     uninominal votes {'c2': 2.0, 'c5': 3.0, 'c1': 2.0, 'c4': 249
7         1.0, 'c3': 1.0} 250
8     minimal number of votes = 1.0 251
9     maximal number of votes = 3.0 252
10    candidate to remove = c3 253
11    candidate to remove = c4 254
12    remaining candidates = ['c2', 'c5', 'c1'] 255
13 ==> stage = 2 256
14     remaining candidates ['c2', 'c5', 'c1'] 257
15     uninominal votes {'c2': 3.0, 'c5': 3.0, 'c1': 3.0} 258
16     minimal number of votes = 3.0 259
17     maximal number of votes = 3.0 260
18     candidate to remove = c1 261
19     candidate to remove = c5 262
20     candidate to remove = c2 263
21     remaining candidates = [] 264
21 []

```

CONDORCET's approach for determining the winner of an election is *not decisive* 266
 in this example voting profile. One needs therefore, the case given, exploiting more 267
 sophisticated approaches for finding the winner of the election on the basis of the 268
 given linear ballots (see Chap. 8 and Bisdorff et al. 2008). 269

The NETFLOWS ranked heatmap view on voting profiles with the `showHTMLVotingHeatmap()` 270
`VotingHeatmap()` method is one such tool for showing convincing voting 271
 results (see Sect. 8.3). 272

Listing 7.8 NETFLOWS ranked heatmap view on a voting profile

```

1 >>> v.showHTMLVotingHeatmap(rankingRule='Netflows', \ 273
2 ...             colorLevels=3, Correlations=True) 274

```

It is worthwhile noticing in Fig. 7.3 on the facing page that the compromise 275
 NETFLOWS ranking $c4 > c5 > c2 > c3 > c1$, shown in this heatmap, results in 276
 the highest possible *ordinal correlation* index of +0.778 with the majority margins 277
 digraph (see Chap. 16). This NETFLOWS ranking result corresponds also to the 278
 BORDA scores ranking.³ 279

Listing 7.9 Rank analysis table with BORDA scores

```

1 >>> v.showRankAnalysisTable() 280
2     *---- Borda rank analysis tableau -----* 281
3     candi- | alternative-to-rank           | Borda 282
4     dates  | 1      2      3      4      5      | score  average 283

```

³ Mind that BORDA scores require the unrealistic working hypothesis that one knows how to precisely code in numbers the marginal linear ranks per voter.

Fig. 7.3 Visualising a linear voting profile in a NETFLOWS ranked heatmap.

As the number of voters is usually much larger than the number of eligible candidates, the voting heatmap is by default transposed into a voters \times candidates view. Notice that the importance weights of the voters are *negative*, which means that the preference direction of the criteria (in this case the individual voters) is *decreasing* (min), i.e. goes from lowest (best) rank to highest (worst) rank

Voting Heatmap

criteria	weight	tau*	c4	c5	c2	c3	c1
v5	-1.00	+0.60	1	4	2	3	5
v3	-1.00	+0.60	3	1	2	4	5
v8	-1.00	+0.40	2	3	1	5	4
v7	-1.00	+0.40	2	1	5	3	4
v6	-1.00	+0.40	2	3	1	5	4
v9	-1.00	+0.20	3	1	5	2	4
v4	-1.00	+0.00	2	4	5	1	3
v2	-1.00	-0.40	3	5	2	4	1
v1	-1.00	-0.80	5	3	4	2	1

Color legend:

quantile	33.33%	66.66%	100.00%
----------	--------	--------	---------

(* tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation

Outranking model: robust, Ranking rule: NetFlows

Ordinal (Kendall) correlation between

global ranking and global outranking relation: +0.778

5	-----	-----	-----	-----	-----	-----	-----	284	
6	'c4'	1	4	3	0	1	23	2.56	285
7	'c5'	3	0	3	2	1	25	2.78	286
8	'c2'	2	3	0	1	3	27	3.00	287
9	'c3'	1	2	2	2	2	29	3.22	288
10	'c1'	2	0	1	4	2	31	3.44	289

Let us eventually notice in the rank analysis table, shown in Listing 7.9 above, that the uninominal plurality winner would be, with three votes, candidate c5, whereas the best NETFLOWS and BORDA ranked candidate c1 obtains with candidate c3 only one vote.

But do represent our example linear voting profiles here realistic election outcomes?

7.5 On Generating Realistic Random Linear Voting Profiles

296

By default, the `RandomLinearVotingProfile` class generates random linear voting profiles where every candidate has the same uniform probability to be ranked at a certain position by all the voters. Each voter's random linear ballot is indeed generated via a uniform shuffling of the list of candidates.

297

In reality, political election data are quite different. There usually will be different favourite and marginal candidates for each political party. To simulate these aspects with our random generator, we are using two random exponentially distributed polls of the candidates and consider a bipartisan political landscape with a certain

298

299

300

301

302

303

304

random balance (default theoretical party repartition = 0.50) between the two sets 305
 of potential party supporters. A certain theoretical proportion (default = 0.1) will 306
 not support any party. 307

Let us generate in Listing 7.10 such a linear voting profile for an election with 308
 1000 voters and 15 candidates. 309

Listing 7.10 Generating a linear voting profile with random polls

```

1  >>> from votingProfiles import\
2  ...             RandomLinearVotingProfile
3  >>> lvp = RandomLinearVotingProfile(\ 310
4  ...             numberofCandidates=15,\ 311
5  ...             numberofVoters=1000,\ 312
6  ...             WithPolls=True,\ 313
7  ...             partyRepartition=0.5,\ 314
8  ...             other=0.1,\ 315
9  ...             seed=0.9189670954954139) 316
10 >>> lvp 317
11 *----- VotingProfile instance description -----* 318
12 Instance class : RandomLinearVotingProfile 319
13 Instance name  : randLinearProfile 320
14 Candidates    : 15 321
15 Voters        : 1000 322
16 Attributes    : ['name', 'seed', 'candidates', 323
17             'voters', 'RandomWeights', 324
18             'sumWeights', 'poll1', 'poll2', 325
19             'bipartisan', 'linearBallot', 326
20             'ballot'] 327
21 >>> lvp.showRandomPolls() 328
22 Random repartition of voters 329
23 Party-1 supporters : 460 (46.0%) 330
24 Party-2 supporters : 436 (43.6%) 331
25 Other voters       : 104 (10.4%) 332
26 *----- random polls -----* 333
27 Party-1(46.0%) | Party-2(43.6%) | expected 334
28 -----
29   c06 : 19.91% |   c11 : 22.94% |   c06 : 15.00% 335
30   c07 : 14.27% |   c08 : 15.65% |   c11 : 13.08% 336
31   c03 : 10.02% |   c04 : 15.07% |   c08 : 09.01% 337
32   c13 : 08.39% |   c06 : 13.40% |   c07 : 08.79% 338
33   c15 : 08.39% |   c03 : 06.49% |   c03 : 07.44% 339
34   c11 : 06.70% |   c09 : 05.63% |   c04 : 07.11% 340
35   c01 : 06.17% |   c07 : 05.10% |   c01 : 05.06% 341
36   c12 : 04.81% |   c01 : 05.09% |   c13 : 05.04% 342
37   c08 : 04.75% |   c12 : 03.43% |   c15 : 04.23% 343
38   c10 : 04.66% |   c13 : 02.71% |   c12 : 03.71% 344
39   c14 : 04.42% |   c14 : 02.70% |   c14 : 03.21% 345
40   c05 : 04.01% |   c15 : 00.86% |   c09 : 03.10% 346
41   c09 : 01.40% |   c10 : 00.44% |   c10 : 02.34% 347
42   c04 : 01.18% |   c05 : 00.29% |   c05 : 01.97% 348
43   c02 : 00.90% |   c02 : 00.21% |   c02 : 00.51% 349

```

In this example (see above Lines 21 and following), we obtain 460 Party-1 supporters (46%), 436 Party-2 supporters (43.6%), and 104 other voters (10.4%). Favourite candidates of Party-1 supporters, with more than 10%, appear to be $c06$ (19.91%), $c07$ (14.27%), and $c03$ (10.02%). Whereas for Party-2 supporters, favourite candidates appear to be $c11$ (22.94%), followed by $c08$ (15.65%), $c04$ (15.07%) and $c06$ (13.4%). Being *first* choice for Party-1 supporters and *fourth* choice for Party-2 supporters, this candidate $c06$ is a natural candidate for clearly winning this election game (see Listing 7.11).

Listing 7.11 The uninominal and BORDA election winner

```

1 >>> lvp.computeSimpleMajorityWinner()                                361
2 ['c06']                                                               362
3 >>> lvp.computeInstantRunoffWinner()                            363
4 ['c06']                                                               364
5 >>> lvp.computeBordaWinners()                                365
6 ['c06']                                                               366

```

Is candidate $c06$ also a CONDORCET winner? To verify, we start by creating the corresponding *majority margins* digraph $mmdg$ with the help of the `MajorityMarginsDigraph` class. The created digraph instance contains 15 *actions*—the candidates—and 104 *oriented arcs*—the *positive* majority margins—(see Listing 7.12 Lines 7–8).

Listing 7.12 A majority margins digraph constructed from a linear voting profile

```

1 >>> from votingProfiles import MajorityMarginsDigraph          372
2 >>> mmdg = MajorityMarginsDigraph(lvp)                         373
3 >>> mmdg                                         374
4 *----- Digraph instance description -----*                  375
5 Instance class      : MajorityMarginsDigraph                  376
6 Instance name       : rel_randLinearProfile                377
7 Digraph Order       : 15                                     378
8 Digraph Size        : 104                                    379
9 Valuation domain   : [-1000.00;1000.00]                  380
10 Determinateness (%) : 67.08                                381
11 Attributes         : ['name', 'actions', 'voters',          382
12                      'ballot', 'valuationdomain',          383
13                      'relation', 'order',          384
14                      'gamma', 'notGamma']          385

```

The `showHTMLRelationTable()` method visualises the resulting pairwise majority margins by showing the HTML formatted version of the $mmdg$ relation table in a browser view.

```

1 >>> mmdg.showHTMLRelationTable(\                                389
2 ...          tableTitle='Pairwise majority margins',          390
3 ...          relationName='M(x,y)')                         391

```

A complete light green *row* in Fig. 7.4 on the following page reveals a CONDORCET winner, whereas a complete light green *column* reveals a CONDORCET

Fig. 7.4 Browsing the majority margins. Light green cells contain the positive majority margins, whereas light red cells contain the negative majority margins

Pairwise majority margins

M(x,y)	c01	c02	c03	c04	c05	c06	c07	c08	c09	c10	c11	c12	c13	c14	c15
c01	—	768	-138	108	478	-436	-198	-140	238	440	-268	148	50	202	218
c02	-768	—	-796	-484	-368	-858	-828	-772	-546	-496	-800	-722	-768	-696	-658
c03	138	796	—	160	590	-286	-80	-8	372	522	-158	280	210	360	338
c04	-108	484	-160	—	184	-370	-180	-288	160	136	-420	16	-62	56	30
c05	-478	368	-590	-184	—	-730	-640	-472	-234	-116	-550	-442	-522	-376	-386
c06	436	858	286	370	730	—	248	234	574	692	102	556	482	566	520
c07	198	828	80	180	640	-248	—	0	358	602	-94	304	266	384	420
c08	140	772	8	288	472	-234	0	—	436	396	-176	276	134	298	244
c09	-238	546	-372	-160	234	-574	-358	-436	—	116	-594	-126	-194	-90	-14
c10	-440	496	-522	-136	116	-692	-602	-396	-116	—	-510	-310	-442	-304	-266
c11	268	800	158	420	550	-102	94	176	594	510	—	388	268	474	292
c12	-148	722	-280	-16	442	-556	-304	-276	126	310	-388	—	-92	100	148
c13	-50	768	-210	62	522	-482	-266	-134	194	442	-268	92	—	158	186
c14	-202	696	-360	-56	376	-566	-384	-298	90	304	-474	-100	-158	—	68
c15	-218	658	-338	-30	386	-520	-420	-244	14	266	-292	-148	-186	-68	—

Valuation domain: [-1000; +1000]

loser. We recover again candidate c06 as CONDORCET winner,⁴ whereas the obvious CONDORCET loser is here candidate c02, the candidate with the lowest support in both parties (see Listing 7.10 on page 92 Line 43).

With a same bipolar *first-ranked* and *last ranked* selection procedure, we may weakly rank the candidates (with possible ties) by iterating these *first-ranked* and *last ranked* choices among the remaining candidates (Bisdorff 1999).

Before showing the *ranking-by-choosing* result, we have to, first, compute the iterated bipolar selection procedure (see Listing 7.13).

Listing 7.13 Ranking by iterating choosing the *first* and *last* remaining candidates

```

1  >>> cdg.showRankingByChoosing()
2  Ranking by Choosing and Rejecting
3  1st best ranked ['c06']
4  2nd best ranked ['c11']
5  3rd best ranked ['c07', 'c08']
6  4th best ranked ['c03']
7  5th best ranked ['c01']
8  6th best ranked ['c13']
9  7th first ranked ['c04']
10 7th last ranked ['c12']
11 6th last ranked ['c14']
12 5th last ranked ['c15']
13 4th last ranked ['c09']
14 3rd last ranked ['c10']
15 2nd last ranked ['c05']
16 1st last ranked ['c02']

```

⁴ The concept of CONDORCET winner—a generalisation of absolute majority winners—proposed by CONDORCET in 1784, is an early historical example of initial digraph kernel (see Chap. 17).

The first selection concerns c06 (first) and c02 (last), followed by c11 (first) 418
opposed to c05 (last), and so on, until there remains at iteration step 7 a last pair of 419
candidates, namely [c04, c12] (see Lines 9–10). 420

Notice furthermore the 3rd-best ranked candidates at iteration step 3 (see Line 5), 421
namely the pair (c07, c08). Both candidates represent indeed conjointly the 3rd 422
best ranked choice. We obtain hence a *weak ranking*, i.e. a ranking with a tie. 423

Let us mention that the *instant runoff* procedure, we used before when operated 424
with a `Comments=True` parameter setting, will deliver a more or less similar 425
reversed linear ordering-by-rejecting result, namely [c02, c10, c14, c05, 426
c09, c13, c12, c15, c04, c01, c08, c03, c07, c11, c06], 427
ordered from the *last* to the *first* choice. 428

Remarkable about both these *ranking-by-choosing* or *ordering-by-rejecting* 429
results is the fact that the random voting behaviour, simulated here with the 430
help of two discrete random variables,⁵ 431 Defined, respectively, by the two random 431
party polls, is rendering a ranking that is more or less in accordance with the 432
simulated balance of the polls: Party-1 supporters : 460; Party-2 supporters: 436 433
(see Listing 7.10 on page 92 Lines 29–43 third column). Despite a random voting 434
behaviour per voter, the given polls apparently show a *very strong incidence* on the 435
eventual election result. In order to avoid any manipulation of the election outcome, 436
public media are therefore in some countries not allowed to publish polls during the 437
last weeks before a general election. 438

Mind that the specific *ranking-by-choosing* procedure, we use here on the 439
majority margins digraph, operates the selection procedure by extracting at each 440
step *initial* and *terminal* prekernels, i.e. difficult operational problems (see Chap. 17 441
and Bisendorff 1999); A technique that does not allow in general to tackle voting 442
profiles with much more than 30 candidates. 443

Next Chap. 8 on multiple incommensurable criteria ranking methods presents 444
more methods and tools for ranking from pairwise majority margins when a larger 445
number of potential candidates or decision alternatives is given. 446

References

447

- Bisendorff R (1999) Bipolar ranking from pairwise fuzzy outrankings. *Belg J Oper Res Stat Comput Sci* 37(4):379–387. <http://hdl.handle.net/10993/38738> 448
449
- Bisendorff R (2020a) Lecture 2: Who wins the election? In: Lectures of the algorithmic decision 450
theory course, University of Luxembourg. <http://hdl.handle.net/10993/37933> 451
- Bisendorff R (2020b) Lecture 4: Discrete random variables. In: Lectures of the computational 452
statistics course, University of Luxembourg. <http://hdl.handle.net/10993/37870> 453

⁵ Discrete random variables with a given empirical probability law (here the polls) are provided in the `randomNumbers` module by the `DiscreteRandomVariable` class (Bisendorff 2020b).

Bisdorff R, Meyer P, Roubens M (2008) Rubis: a bipolar-valued outranking method for the best choice decision problem. <i>Q J Oper Res</i> 6(2):143–165. http://hdl.handle.net/10993/23716	454
de Borda J (1781) Mémoire sur les élections au scrutin. <i>Mémoires de l'Académie royale des sciences</i> , Paris	456
de Caritat, Marquis de Condorcet J (1784) <i>Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix</i> . Imprimerie royale, Paris	458
	459

Uncorrected Proof

Chapter 8

Ranking with Multiple Incommensurable Criteria

1
2
3

... whether we are deciding between buying different commodity baskets, or making choices about what to do on a holiday, or deciding for whom to vote for in an election, we are inescapably involved in evaluating alternatives with non-commensurable aspects.

—Amartya Sen, *Idea of Justice* (Sen 2009)

9

Contents

10

8.1	The Ranking Problem	97	11
8.2	The COPELAND Ranking	100	12
8.3	The NETFLOWS Ranking	102	13
8.4	KEMENY Rankings	104	14
8.5	SLATER Rankings	107	15
8.6	The KOHLER Ranking-by-Choosing Rule	109	16
8.7	The RANKEDPAIRS Ranking Rule	112	17

Abstract The DIGRAPH3 python resources provide several algorithms for solving the multiple incommensurable criteria ranking problem via bipolar-valued outranking digraphs. The COPELAND, NETFLOWS, KEMENY, SLATER, KOHLER, and the RANKEDPAIRS ranking rules are introduced and illustrated with a random outranking digraph.

18
19
20
21
22
23
24
25
26
27

8.1 The Ranking Problem

23

We need to rank without ties a set X of items (usually decision alternatives) that are evaluated on multiple incommensurable performance criteria; yet, for which we know their pairwise bipolar-valued *strict outranking* characteristics, i.e. $r(x \succsim y)$ for all $x, y \in X$ (see Sect. 3.5 and Bisdorff 2013).

24
25
26
27

Let us consider in Listing 8.1 a didactic outranking digraph g generated from a random *Cost-Benefit* performance tableau (see Sect. 6.3) concerning 9 decision alternatives evaluated on 13 performance criteria. The `BipolarOutrankingDigraph` computes the corresponding bipolar-valued *outranking digraph* g and a codual transform gives us the requested strict outranking digraph gcd with the pairwise $r(x \succsim y)$ characteristic values (see Lines 11–19 below).

Listing 8.1 Random bipolar-valued strict outranking relation characteristics

```

1 >>> from randomPerfTabs import RandomCBPerformanceTableau 35
2 >>> t = RandomCBPerformanceTableau(numberOfActions=9, \
3 ...           numberOfCriteria=13, seed=200) 36
4 >>> from outrankingDigraphs import BipolarOutrankingDigraph 37
5 >>> g = BipolarOutrankingDigraph(t) 38
6 >>> gcd = ~(-g) # strict (codual) outranking digraph 39
7 >>> gcd.showRelationTable(ReflexiveTerms=False) 40
8 * ---- Relation Table ---- 41
9 r(>) | 'a1' 'a2' 'a3' 'a4' 'a5' 'a6' 'a7' 'a8' 'a9' 42
10 ----- 43
11 'a1' | - 0.00 +0.10 -1.00 -0.13 -0.57 -0.23 +0.10 +0.00 44
12 'a2' | -1.00 - 0.00 +0.00 -0.37 -0.42 -0.28 -0.32 -0.12 45
13 'a3' | -0.10 0.00 - -0.17 -0.35 -0.30 -0.17 -0.17 +0.00 46
14 'a4' | 0.00 0.00 -0.42 - -0.40 -0.20 -0.60 -0.27 -0.30 47
15 'a5' | +0.13 +0.22 +0.10 +0.40 - +0.03 +0.40 -0.03 -0.07 48
16 'a6' | -0.07 -0.22 +0.20 +0.20 -0.37 - +0.10 -0.03 -0.07 49
17 'a7' | -0.20 +0.28 -0.03 -0.07 -0.40 -0.10 - - +0.27 +1.00 50
18 'a8' | -0.10 -0.02 -0.23 -0.13 -0.37 +0.03 -0.27 - +0.03 51
19 'a9' | 0.00 +0.12 -1.00 -0.13 -0.03 -0.03 -1.00 -0.03 - - 52

```

Some ranking rules will work on the associated CONDORCET digraph, i.e. the corresponding *median cut* polarised strict outranking digraph. 55

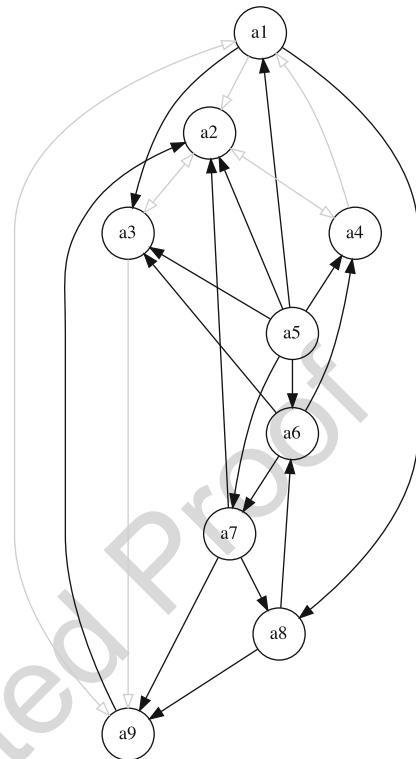
Listing 8.2 Median cut polarised strict outranking relation characteristics

```

1 >>> ccd = PolarisedOutrankingDigraph(gcd,\                                56
2 ...                                     level=g.valuationdomain['med'],\      57
3 ...                                     KeepValues=False,StrictCut=True)      58
4 >>> ccd.showRelationTable(ReflexiveTerms=False,\                         59
5 ...                                     IntegerValues=True)                  60
6 ---- Relation Table ----                                         61
7 r(>)_med | 'a1' 'a2' 'a3' 'a4' 'a5' 'a6' 'a7' 'a8' 'a9' 62
8 -----|----- 63
9 'a1' | - 0 +1 -1 -1 -1 -1 +1 0 64
10 'a2' | -1 - +0 0 -1 -1 -1 -1 -1 65
11 'a3' | -1 0 - -1 -1 -1 -1 -1 0 66
12 'a4' | 0 0 -1 - -1 -1 -1 -1 -1 67
13 'a5' | +1 +1 +1 +1 - +1 +1 -1 -1 68
14 'a6' | -1 -1 +1 +1 -1 - - +1 -1 -1 69
15 'a7' | -1 +1 -1 -1 -1 -1 - - +1 +1 70
16 'a8' | -1 -1 -1 -1 -1 +1 -1 - - +1 71
17 'a9' | 0 +1 -1 -1 -1 -1 -1 -1 - - 72

```

Fig. 8.1 The strict outranking relation \succ . The relation shown here is, for instance, *not transitive*: alternative a_8 outranks alternative a_6 and alternative a_6 outranks a_4 , however, a_8 does not outrank a_4 . Furthermore, alternatives a_6 , a_7 , and a_8 show a cyclic outranking relation



Digraph3 (graphviz), R. Bisdorff, 2020

Unfortunately, such crisp median cut CONDORCET digraphs, associated with a 73 given strict outranking digraph, only exceptionally model a linear ordering. Usually, 74 pairwise majority comparisons do not render a *complete* or, at least, a *transitive* 75 partial order. There may even frequently appear *cyclic* outranking situations (see 76 Sect. 7.4). 77

To discover how *difficult* this ranking problem can get, let us have a look in 78 Fig. 8.1 at the corresponding strict outranking digraph *graphviz* drawing.¹ 79

```

1  >>> gcd.exportGraphViz(fileName='rankingTutorial')          80
2  *---- exporting a dot file for GraphViz tools -----* 81
3  Exporting to rankingTutorial.dot 82
4  dot -Grankdir=BT -Tpng rankingTutorial.dot\ 83
      -o rankingTutorial.png 84

```

The `computeTransitivityDegree()` method computes the *transitivity* 85 degree of the outranking digraph `gcd` shown in Fig. 8.1, i.e. the ratio of the number 86

¹ The `exportGraphViz()` method is depending on drawing tools from *graphviz* software (<https://graphviz.org/>).

of closed transitive triples— $x \succsim y$, $y \succsim z$ and $x \succsim z$ —over the number of triples 87
 $x \succsim y$, $y \succsim z$ (see below Lines 3–4). 88

```

1 >>> gcd.computeTransitivityDegree(Comments=True) 89
2 Transitivity degree of graph <codual_rel_randomCBperftab> 90
3 triples x>y>z: 78, closed: 38, open: 40 91
4 closed/triples = 0.487 92

```

With only 49% of the required transitive arcs, the strict outranking digraph gcd 93
is hence very far from being transitive; a serious problem when a linear ordering of 94
the decision alternatives is looked for. 95

The `computeChordlessCircuits()` method, followed by the `showChordlessCircuits()` method, can check furthermore whether there appear 96
any cyclic outranking situations.² 97

```

1 >>> gcd.computeChordlessCircuits () 99
2 >>> gcd.showChordlessCircuits () 100
3 1 circuit(s). 101
4 ----- Chordless circuits -----* 102
5 1: ['a6', 'a7', 'a8'] , credibility : 0.033 103

```

There is one chordless circuit detected in the given strict outranking digraph 104
gcd, namely alternative a6 outranks alternative a7, the latter outranks a8, and 105
a8 outranks again alternative a6 (see Fig. 8.1 on the previous page). Any potential 106
linear ordering of these three alternatives will, in fact, always contradict somehow 107
the given outranking relation. 108

Now, several heuristic ranking rules have been proposed for constructing a linear 109
ordering which is closest in some specific sense to a given outranking relation. The 110
DIGRAPH3 resources provide some of the most common of these ranking rules, like 111
the COPELAND, KEMENY, SLATER, KOHLER, and the RANKEDPAIRS ranking 112
rules. 113

8.2 The COPELAND Ranking

114

Algorithm 8.1 COPELAND ranking rule (Copeland 1951)

in : median cut strict outranking digraph $G(X, \succsim_0)$,
out : linear ranking of the set X of decision actions.

1. For each alternative $x \in X$, the COPELAND ranking rule computes a score resulting from the sum over all non-reflexive pairs (x, y) of the differences between the crisp *strict outranking* characteristics $r(x \succsim_0 y)$ and the crisp *strict outranked* characteristics $r(y \succsim_0 x)$.
 2. The alternatives are ranked in decreasing order of these COPELAND scores; ties, the case given, being resolved with a lexicographical rule applied to the identifiers of the decision actions.
-

² The `computeChordlessCircuits()` and `showChordlessCircuits()` methods are separate because there are various methods available for enumerating the chordless circuits in a digraph (Bisdorff 2010).

In Listing 8.3 we show the ranking of the given strict outranking digraph gcd 115 obtained with the `CopelandRanking` class from the `linearOrders` module. 116

Listing 8.3 Computing a COPELAND ranking

```

1 >>> from linearOrders import CopelandRanking 117
2 >>> cop = CopelandRanking(gcd, Comments=True) 118
3 Copeland decreasing scores 119
4     a5 : +12 120
5     a1 : +2 121
6     a6 : +2 122
7     a7 : +2 123
8     a8 : 0 124
9     a4 : -3 125
10    a9 : -3 126
11    a3 : -5 127
12    a2 : -7 128
13 Copeland Ranking: 129
14 ['a5','a1','a6','a7','a8','a4','a9','a3','a2'] 130

```

Alternative a_5 obtains here the best COPELAND score (+12), followed by 131 alternatives a_1 , a_6 , and a_7 with same score (+2); following the lexicographic rule, 132 a_1 is hence ranked before a_6 and a_6 before a_7 . Same situation is observed for a_4 133 and a_9 with a score of -3 (see Listing 8.3 Lines 4–12). The COPELAND ranking 134 rule is in fact *invariant* under the *codual* transform (see Sect. 2.6) and renders a same 135 linear order indifferently from digraphs g or gcd . 136

The COPELAND rule works well, furthermore on any strict outranking digraph 137 that models a linear (partial) order on the *median cut* strict outranking digraph ccd 138 (Dias and Lamboray 2010). In Listing 8.4, the `computeRankingCorrelation()` 139 method, coupled with the `showCorrelation()` method, indicates the 140 ordinal correlation of the COPELAND ranking result, shown in Listing 8.3 Line 14, 141 with the given outranking digraph g (see Chap. 16 and Bisдорff 2012). 142

Listing 8.4 Checking the ordinal quality of the COPELAND ranking

```

1 >>> corr = g.computeRankingCorrelation(cop.copelandRanking) 143
2 >>> g.showCorrelation(corr) 144
3 Correlation indexes: 145
4     Crisp ordinal correlation : +0.463 146
5     Valued equivalence : +0.107 147
6     Epistemic determination : 0.230 148

```

With an epistemic determination level of 0.230 (Line 6), the crisp ordinal 149 correlation—KENDALL τ —index of +0.463 is in fact supported by 150 61.5%(100.0x(1.0 + 0.23)/2) of the strict outranking situations. Furthermore, 151 the bipolar-valued *relational equivalence* between the strict outranking relation 152 and the COPELAND ranking equals +0.107, i.e. a *majority* of 55.35% of the criteria 153 significance supports the relational equivalence between the given strict outranking 154 relation and the COPELAND ranking (see Chap. 16). 155

The COPELAND scores, shown in Listing 8.3 on the previous page 156 deliver actually only a *weak ranking*, i.e. a ranking with ties. This weak 157

ranking may be constructed with the `WeakCopelandOrder` class from the `transitiveDigraphs` module. 158

159

Listing 8.5 Computing a weak COPELAND ranking

```

1 >>> from transitiveDigraphs import WeakCopelandOrder 160
2 >>> wcop = WeakCopelandOrder(g) 161
3 >>> wcop.showRankingByChoosing() 162
4 Ranking by Choosing and Rejecting 163
5   1st ranked ['a5'] 164
6     2nd ranked ['a1', 'a6', 'a7'] 165
7       3rd ranked ['a8'] 166
8         3rd last ranked ['a4', 'a9'] 167
9           2nd last ranked ['a3'] 168
10          1st last ranked ['a2'] 169

```

In Listing 8.5, the `WeakCopelandOrder` class models the weak ranking 170
 actually delivered by the COPELAND scores (see Listing 8.3 on the previous page). 171
 We may draw its corresponding skeleton.³ 172

```

1 >>> wcop.exportGraphViz(fileName='weakCopelandRanking') 173
2 *---- exporting a dot file for GraphViz tools -----* 174
3 Exporting to weakCopelandRanking.dot 175
4 dot -Grankdir=TB -Tpng weakCopelandRanking.dot\ 176
5           -o weakCopelandRanking.png 177

```

Let us now consider a similar ranking rule, but working directly on the criteria 178
significance majority margins, i.e. the *bipolar-valued* outranking characteristic 179
 valuation. 180

8.3 The NETFLOWS Ranking

181

The bipolar-valued version of the COPELAND ranking rule, we call NETFLOWS,⁴ 182
 computes for each alternative $x \in X$ a *net flow* score. 183

Algorithm 8.2 NETFLOWS ranking rule

in : bipolar-valued strict outranking digraph $G(X, \succsim)$,
out : linear ranking of the set X of decision actions.

1. The NETFLOWS rule computes for each alternative $x \in X$ the sum over all non-reflexive $y \in X$ of the differences between the *strict outranking* characteristics $r(x \succsim y)$ and the *strict outranked* characteristics $r(y \succsim x)$.
 2. The alternatives are ranked in decreasing order of these NETFLOWS scores; ties, the case given, being resolved with a lexicographical rule applied to the identifiers of the decision actions.
-

³ The skeleton of a transitive relation drops the transitivity induced arcs.

⁴ This ranking rule is also known under the name PROMETHEE ranking rule (Brans and Vincke 1985).

The `NetFlowsRanking` class from the `linearOrders` module computes 184
this linear ranking from the given strict outranking digraph `gcd`. 185

Listing 8.6 Computing a NETFLOWS ranking

```

1 >>> from linearOrders import NetFlowsRanking 186
2 >>> nf = NetFlowsRanking(gcd, Comments=True) 187
3   Net flow scores : 188
4     a5 : +3.600 189
5     a7 : +2.800 190
6     a6 : +1.300 191
7     a3 : +0.033 192
8     a1 : -0.400 193
9     a8 : -0.567 194
10    a4 : -1.283 195
11    a9 : -2.600 196
12    a2 : -2.883 197
13 NetFlows Ranking: 198
14   ['a5','a7','a6','a3','a1','a8','a4','a9','a2'] 199
15 >>> cop.copelandRanking # comparing both 200
16   ['a5','a1','a6','a7','a8','a4','a9','a3','a2'] 201

```

In Listing 8.6 here, the NETFLOWS scores actually deliver directly a linear 202 ranking *without ties* which is rather different from the one delivered by the 203 COPELAND rule (compare Lines 14 and 16). It may happen, however, that we 204 obtain, as with the COPELAND scores above, only a ranking with ties, which 205 may then be resolved similarly by following a lexicographic rule applied to the 206 identifiers of the decision alternatives. In such cases, it is possible to construct again 207 a *weak ranking* with the corresponding `WeakNetFlowsOrder` class from the 208 `transitiveDigraphs` module. 209

It is worthwhile noticing again, that similar to the COPELAND ranking rule seen 210 before, the NETFLOWS ranking rule is also *invariant* under the codual transform 211 (see Sect. 2.6) and delivers the same ranking result indifferently from digraph `g` or 212 `gcd`. 213

The NETFLOWS ranking result appears to be better correlated (+0.638 vs. 214 +0.463) with the given strict outranking relation than its crisp cousin, the 215 COPELAND ranking (see Line 4 in Listing 8.7). 216

Listing 8.7 Checking the quality of the NETFLOWS ranking

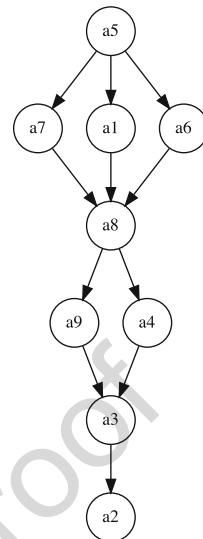
```

1 >>> corr = gcd.computeOrdinalCorrelation(nf) 217
2 >>> gcd.showCorrelation(corr) 218
3   Correlation indexes: 219
4     Extended Kendall tau      : +0.638 220
5     Epistemic determination   :  0.230 221
6     Bipolar-valued equivalence : +0.147 222

```

Indeed, the ordinal correlation index of +0.638 leads to a bipolar-valued 223 *relational equivalence* characteristics of +0.147, i.e. a majority of 57.35% of 224 the criteria significance supports the relational equivalence between the given 225 outranking digraphs `g` or `gcd` and the corresponding NETFLOWS ranking (see 226

Fig. 8.2 Drawing of the weak COPELAND ranking.
The drawing shows the skeleton of the weak ranking produced by the corresponding ties of the COPELAND scores



Digraph3 (graphviz)
R. Bisdorff, 2020

Chap. 16). The weaker ordinal ranking quality of the COPELAND rule (+0.463) 227 stems in this example here essentially from the *weakness* of the actual COPELAND 228 ranking result and our subsequent *arbitrary* lexicographic resolution of the many 229 ties given by the COPELAND scores (see Fig. 8.2). 230

To appreciate now the ordinal correlations of both the COPELAND and the 231 NETFLOWS rankings with the underlying strict outranking relation, it is useful to 232 consider the ‘optimal’ KEMENY and SLATER ranking rules. 233

8.4 KEMENY Rankings

A KEMENY ranking k is a linear ranking without ties of the set of n decision 235 alternatives X which is *closest*, in the sense of the ordinal KENDALL distance,⁵ 236 to the given valued outranking digraph g (Kemeny 1959). 237

Formally: 238

$$k = \operatorname{argmax}_{p \in \mathcal{P}(X)} \sum_{i \neq j} (r(p[i] \succ p[j]) - r(p[j] \succ p[i])), \quad (8.1)$$

where $\mathcal{P}(X)$ denotes the set of all permutations of X and $i, j = 0, \dots, n$. 239

⁵ See Chap. 16 and Bisdorff (2012).

The KemenyRanking class from the `linearOrders` module computes such a ranking which is highest possible correlated with the underlying strict outranking relation. The KEMENY rule is also *invariant* under the codual transform.

Mind that the KemenyRanking class constructor, in order to find a KEMENY ranking, has to compute NETFLOWS scores for every permutation of the list of decision alternatives (see Eq. 8.1 on the preceding page). Therefore the class is limited, by default, to digraphs of order up to 7 (Bisdorff 2021). In Listing 8.8 Line 2, the `orderLimit` parameter allows to raise this limit.

Listing 8.8 Computing a KEMENY ranking

```

1 >>> from linearOrders import KemenyRanking          248
2 >>> ke = KemenyRanking(gcd,orderLimit=9)           249
3 >>> # default orderLimit is 7                      250
4 >>> ke.showRanking()                                251
5  ['a5','a6','a7','a3','a9','a4','a1','a8','a2']    252
6 >>> corr = gcd.computeOrdinalCorrelation(ke)        253
7 >>> gcd.showCorrelation(corr)                      254
8 Correlation indexes:                                255
9   Extended Kendall tau      : +0.779               256
10  Epistemic determination   :  0.230               257
11  Bipolar-valued equivalence: +0.179               258

```

So, $+0.779$ represents the *highest possible* ordinal correlation index—*fitness*—any potential linear ranking can achieve with the given pairwise outranking digraph (see Listing 8.8 Line 9).

A KEMENY ranking may not be unique. In our example here, we obtain in fact two such KEMENY rankings with a same *maximal* KEMENY index of 12.92.

Listing 8.9 Optimal KEMENY rankings

```

1 >>> ke.maximalRankings          264
2  [['a5','a6','a7','a3','a8','a9','a4','a1','a2'], 265
3  ['a5','a6','a7','a3','a9','a4','a1','a8','a2']] 266
4 >>> ke.maxKemenyIndex        267
5 Decimal('12.9166667')          268

```

Figure 8.3 on the following page shows the partial order defined by the epistemic disjunction of both optimal KEMENY rankings obtained with the `RankingsFusionDigraph` class (see Sect. 2.5).

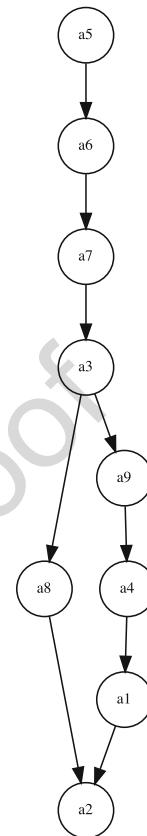
Listing 8.10 Computing the epistemic disjunction of all optimal KEMENY rankings

```

1 >>> from transitiveDigraphs import RankingsFusionDigraph 272
2 >>> wke = RankingsFusionDigraph(ke,ke.maximalRankings) 273
3 >>> wke.exportGraphViz(fileName='tutorialKemeny')        274
4 *---- exporting a dot file for GraphViz tools -----* 275
5   Exporting to tutorialKemeny.dot                         276
6   dot -Grankdir=TB -Tpng tutorialKemeny.dot\           277
7       -o tutorialKemeny.png                                278

```

Fig. 8.3 *Epistemic disjunction of optimal KEMENY rankings.* It is interesting to notice that both KEMENY rankings only differ in their respective positioning of alternative a_8 ; either before or after alternatives a_9 , a_4 , and a_1



Digraph3 (graphviz)
R. Bisdorff, 2020

To retain now a specific representative among all the potential rankings with maximal KEMENY index, we will choose, with the help of the `showRankingConsensusQuality()` method, the one proposing the best criteria consensus. 279
280
281

Listing 8.11 Computing the consensus quality of the first KEMENY ranking

```

1 >>> g.showRankingConsensusQuality(ke.maximalRankings[0]) 282
2 Consensus quality of ranking: 283
3 ['a5','a6','a7','a3','a8','a9','a4','a1','a2'] 284
4 crit. (weight): tau | crit. (weight): tau 285
5 -----|----- 286
6 b09 (0.050) : +0.361 | b04 (0.050) : +0.333 287
7 b08 (0.050) : +0.292 | b01 (0.050) : +0.264 288
8 c01 (0.167) : +0.250 | b03 (0.050) : +0.222 289
9 b07 (0.050) : +0.194 | b05 (0.050) : +0.167 290
10 c02 (0.167) : +0.000 | b10 (0.050) : +0.000 291
11 b02 (0.050) : -0.042 | b06 (0.050) : -0.097 292
12 c03 (0.167) : -0.167 293
13 Summary: 294
14     Weighted mean marginal correlation (a): +0.099 295
15     Standard deviation (b) : +0.177 296
16     Ranking fairness (a) - (b) : -0.079 297
  
```

In Listing 8.11 on the preceding page are shown the ordinal correlation of the first 298 Kemeny ranking with all the marginal strict outranking relations per performance 299 criterion (see Lines 6–12). The Benefit criterion b_{09} is highest correlated (+0.361, 300 Line 6), whereas the Costs criterion c_{03} is lowest correlated (−167, Line 12) with 301 this KEMENY ranking. 302

The ranking consensus quality of the second KEMENY ranking is shown in 303 Listing 8.12 below. 304

Listing 8.12 Computing the consensus quality of the second KEMENY ranking

```

1 >>> g.showRankingConsensusQuality(ke.maximalRankings[1]) 305
2 Consensus quality of ranking: 306
3 ['a5','a6','a7','a3','a9','a4','a1','a8','a2'] 307
4 crit. (weight): tau | crit. (weight): tau 308
5 ----- 309
6 b09 (0.050) : +0.306 | b08 (0.050) : +0.236 310
7 c01 (0.167) : +0.194 | b07 (0.050) : +0.194 311
8 c02 (0.167) : +0.167 | b04 (0.050) : +0.167 312
9 b03 (0.050) : +0.167 | b01 (0.050) : +0.153 313
10 b05 (0.050) : +0.056 | b02 (0.050) : +0.014 314
11 b06 (0.050) : -0.042 | c03 (0.167) : -0.111 315
12 b10 (0.050) : -0.111 316
13 Summary: 317
14     Weighted mean marginal correlation (a): +0.099 318
15     Standard deviation (b) : +0.132 319
16     Ranking fairness (a)-(b) : -0.033 320

```

The Benefit criterion b_{09} is again highest correlated (+0.306, Line 6) with this 321 KEMENY ranking, but the weakest correlated criterion is now Benefit criterion b_{10} 322 (−0.111, Line 12). 323

Both KEMENY rankings show the same *weighted mean marginal correlation* 324 (+0.099) with all 13 performance criteria. However, the second ranking shows a 325 slightly lower *standard deviation*: +0.132 versus +0.177, resulting in a slightly 326 *fairer* ranking result: −0.033 versus −0.079 (see Listing 8.11 on the facing page 327 Lines 14–16 and Listing 8.12 Lines 14–16). 328

When several rankings with maximal KEMENY index are given, the Kemeny- 329 Ranking class constructor instantiates the ranking with *highest* mean marginal 330 correlation and, in case of ties, with *lowest* weighted standard deviation. Here we 331 obtain ranking: [a5, a6, a7, a3, a9, a4, a1, a8, a2] (see Line 5 in Listing 8.8 on 332 page 105). 333

8.5 SLATER Rankings

The SLATER ranking rule is identical to the KEMENY rule, but it is working- 335 instead, on the CONDORCET—*median cut polarised*—digraph ccd (Slater 336 1961). The SLATER rule is again *invariant* under the codual transform and the 337 `SlaterRanking` class from the `linearOrders` module delivers hence indif- 338 ferently on `g` or `gcd` the following results: 339

Fig. 8.4 *Epistemic disjunction of optimal SLATER rankings. What precise SLATER ranking result should we hence adopt?*

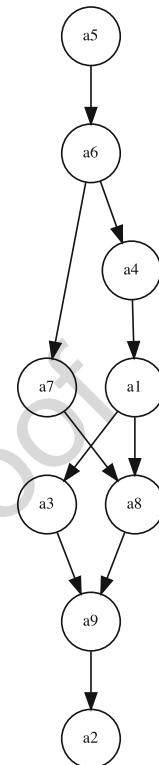


Diagram3 (graphviz)
R. Bisdorff, 2020

Listing 8.13 Computing a SLATER ranking

```

1  >>> from linearOrders import SlaterRanking 340
2  >>> sl = SlaterRanking(gcd,orderLimit=9) 341
3  >>> sl.slaterRanking 342
4  ['a5','a6','a4','a1','a3','a7','a8','a9','a2'] 343
5  >>> corr = gcd.computeRankingCorrelation(sl.slaterRanking) 344
6  >>> sl.showCorrelation(corr) 345
7  Correlation indexes: 346
8      Extended Kendall tau      : +0.676 347
9      Epistemic determination   :  0.230 348
10     Bipolar-valued equivalence : +0.156 349
11  >>> len(sl.maximalRankings) 350
12  7 351
  
```

We notice in Listing 8.13 above that the SLATER ranking shown in Line 4 represents a rather good fit (+0.676), slightly better apparently than the NETFLOWS ranking result (+0.638). However, there are in fact 7 such optimal SLATER rankings (see Line 12). The corresponding epistemic disjunction, again with the RankingsFusionDigraph class from the transitiveDigraphs module, gives the partial ordering shown in Fig. 8.4:

Listing 8.14 Computing the epistemic disjunction of optimal SLATER rankings

```

1 >>> from transitiveDigraphs import RankingsFusionDigraph 358
2 >>> slw = RankingsFusionDigraph(sl, sl.maximalRankings) 359
3 >>> slw.exportGraphViz(fileName='tutorialSlater') 360
4 ----- exporting a dot file for GraphViz tools -----
5   Exporting to tutorialSlater.dot 361
6   dot -Grankdir=TB -Tpng tutorialSlater.dot\ 362
      -o tutorialSlater.png 363
7

```

The KEMENY and SLATER ranking rules are furthermore computationally *difficult* problems and effective ranking results are only computable for tiny outranking digraphs (<20 objects) (see Eq. 8.1 on page 104). 365
366
367

More computationally efficient ranking heuristics, like the COPELAND and NETFLOWS rules, are therefore needed in practice. Let us finally, after these 368
369
370 *ranking-by-scoring* strategies, also present two popular *ranking-by-choosing* rules. 370

8.6 The KOHLER Ranking-by-Choosing Rule

371

Algorithm 8.3 The KOHLER *ranking-by-choosing* rule (Kohler 1978)

in : bipolar-valued digraph $G(X, \precsim)$,
out : linear ranking of the set X of decision actions.

At step i (i goes from 1 to n) do the following:

1. Compute for each row of the bipolar-valued *strict* outranking relation table (see Listing 8.1 on page 98) the smallest value;
2. Select the row where this minimum is maximal. Ties are resolved in lexicographic order;
3. Put the selected decision alternative at rank i ;
4. Delete the corresponding row and column from the relation table and restart until the table is empty.

Listing 8.15 Computing a KOHLER ranking

```

1 >>> from linearOrders import KohlerRanking 372
2 >>> kocd = KohlerRanking(gcd) 373
3 >>> kocd.showRanking() 374
4   ['a5', 'a7', 'a6', 'a3', 'a9', 'a8', 'a4', 'a1', 'a2'] 375
5 >>> corr = gcd.computeOrdinalCorrelation(kocd) 376
6 >>> gcd.showCorrelation(corr) 377
7   Correlation indexes: 378
8     Extended Kendall tau : +0.747 379
9     Epistemic determination : 0.230 380
10    Bipolar-valued equivalence : +0.172 381

```

With this *min-max* lexicographic ranking-by-choosing strategy, we obtain a 382
correlation result (+0.747) that is until now clearly the nearest to an optimal 383

KEMENY ranking (see Listing 8.8 on page 105). Only two adjacent pairs: (a6, 384
 a7) and (a8, a9) are actually inverted here. Notice that the KOHLER ranking 385
 rule, contrary to the previously mentioned rules, is **not** invariant under the codual 386
 transform and requires to work on the strict outranking digraph g_{cd} for a better 387
 correlation result. 388

```

1 >>> ko = KohlerRanking(g) 389
2 >>> corr = g.computeOrdinalCorrelation(ko) 390
3 >>> g.showCorrelation(corr) 391
4   Correlation indexes: 392
5     Crisp ordinal correlation : +0.483 393
6     Epistemic determination   : 0.230 394
7     Bipolar-valued equivalence : +0.111 395

```

But the KOHLER rule has a *dual* version, the prudent *Arrow-Raynaud* ordering- 396
 by-choosing rule, where a corresponding *max-min* strategy, when used on the *non-* 397
strict dual outranking digraph $-g$, for ordering from *last* to *first* produces a similar 398
 ranking result (Arrow and Raynaud 1986). 399

Noticing that the NETFLOWS score of an alternative x represents in fact a 400
 bipolar-valued characteristic of the assertion “*alternative x is ranked first*”, we may 401
 enhance the KOHLER rule by replacing the simple *min-max* strategy with an *iterated* 402
 maximal NETFLOWS score selection. 403

Algorithm 8.4 The iterated NETFLOWS ranking-by-choosing rule

in : bipolar-valued digraph $G(X, \succsim)$,
out : linear ranking of the set X of n decision actions.

At step i (i goes from 1 to n) do the following:

1. Compute for each row of the bipolar-valued outranking relation table (see Listing 8.1 on page 98) the corresponding NETFLOWS score;
 2. Select the row where this score is *maximal*, ties being resolved by lexicographic order;
 3. Put the corresponding decision alternative at rank i ;
 4. Delete the corresponding row and column from the relation table and restart until the table is empty.
-

The `IteratedNetFlowsRanking` class from the `linearOrders` module 404
 computes this ranking result. 405

Listing 8.16 Ranking-by-choosing with iterated maximal NETFLOWS scores

```

1 >>> from linearOrders import IteratedNetFlowsRanking 406
2 >>> inf = IteratedNetFlowsRanking(g) 407
3 >>> inf 408
4   ----- Digraph instance description -----* 409
5     Instance class      : IteratedNetFlowsRanking 410
6     Instance name       : rel_randomCBperfTab_ranked 411
7     Digraph Order       : 9 412
8     Digraph Size        : 36 413

```

```

9  Valuation domain      : [-1.00;1.00]          414
10 Determinateness (%) : 100.00                  415
11 Attributes          : ['valuedRanks', 'valuedOrdering', 416
12                  'iteratedNetFlowsRanking', 417
13                  'iteratedNetFlowsOrdering', 418
14                  'name', 'actions', 'order', 419
15                  'valuationdomain', 'relation', 420
16                  'gamma', 'notGamma'] 421
17 >>> inf.iteratedNetFlowsRanking 422
18 ['a5','a7','a6','a3','a4','a1','a8','a9','a2'] 423
19 >>> corr = g.computeRankingCorrelation(\ 424
20 ...          inf.iteratedNetFlowsRanking) 425
21 >>> g.showCorrelation(corr) 426
22 Correlation indexes: 427
23   Crisp ordinal correlation : +0.743 428
24   Epistemic determination   : 0.230 429
25   Bipolar-valued equivalence : +0.171 430

```

Like the KOHLER rule, the iterated NETFLOWS rule has also a dual *ordering-by-choosing* version, where instead of selecting at each step i the row with maximal NETFLOWS score, we choose the row with the *minimal* NETFLOWS score. Both the ranking and ordering result are computed by the IteratedNetFlowsRanking class (see Lines 12–13 in Listing 8.16 on the preceding page).

```

1 >>> inf.iteratedNetFlowsOrdering 436
2 ['a2','a9','a1','a4','a3','a8','a7','a6','a5'] 437
3 >>> corr = g.computeOrderCorrelation(\ 438
4 ...          inf.iteratedNetFlowsOrdering) 439
5 >>> g.showCorrelation(corr) 440
6 Correlation indexes: 441
7   Crisp ordinal correlation : +0.751 442
8   Epistemic determination   : 0.230 443
9   Bipolar-valued equivalence : +0.173 444

```

The iterated NETFLOWS ranking and its dual, the iterated NETFLOWS ordering, do not usually deliver both the same result. In our example outranking digraph g , for instance, it is the *ordering-by-choosing* result that obtains a slightly better correlation with the given outranking digraph (+0.751), a result that is also slightly better than the original KOHLER ranking result (+0.747, see Listing 8.15 on page 109 Line 8 on page 109).

With different *ranking-by-choosing* and *ordering-by-choosing* results, it may be useful to *fuse* now, similar to what we have done before with KEMENY's and SLATER's optimal rankings, both, the iterated NETFLOWS ranking and ordering into a partial ranking. But we are hence back to the practical problem of what linear ranking should we eventually retain?

Let us finally mention another interesting *ranking-by-choosing* approach.

8.7 The RANKEDPAIRS Ranking Rule

457

N. Tideman's ranking-by-choosing heuristic, the RANKEDPAIRS rule, working best 458
 this time on the non-strict outranking digraph g , is based on a *prudent incremental* 459
 construction of linear orders that avoids on the fly any cycling outranking situations 460
 (Tideman 1987). 461

Algorithm 8.5 The RANKEDPAIRS ranking rule

in : bipolar-valued digraph $G(X, \succsim)$,
out : linear ranking of the set X of decision actions.

1. Rank the ordered pairs (x, y) of alternatives in decreasing order of $r(x \succsim y) + r(y \succsim x)$;
 2. Consider the pairs in that order (ties are resolved by a lexicographic rule):
 - if the next pair does not create a *circuit* with the pairs already blocked, block this pair;
 - if the next pair creates a *circuit* with the already blocked pairs, skip it.
-

With our didactic outranking digraph g , we get the following result. 462

Listing 8.17 Computing a RANKEDPAIRS ranking

```

1 >>> from linearOrders import RankedPairsRanking
2 >>> rp = RankedPairsRanking(g)
3 >>> rp.showRanking()
4 ['a5', 'a6', 'a7', 'a3', 'a8', 'a9', 'a4', 'a1', 'a2']
```

The RANKEDPAIRS rule renders in our example here luckily one of the two 467
 optimal KEMENY ranking, as we may verify below. 468

```

1 >>> ke.maximalRankings
2 [[ 'a5', 'a6', 'a7', 'a3', 'a8', 'a9', 'a4', 'a1', 'a2'],
3  [ 'a5', 'a6', 'a7', 'a3', 'a9', 'a4', 'a1', 'a8', 'a2']]
4 >>> corr = g.computeOrdinalCorrelation(rp)
5 >>> g.showCorrelation(corr)
6 Correlation indexes:
7   Extended Kendall tau      : +0.779
8   Epistemic determination   : 0.230
9   Bipolar-valued equivalence : +0.179
```

Similar to KOHLER's rule, the RANKEDPAIRS rule has also a prudent dual 478
 version, the *Dias-Lamboray ordering-by-choosing* rule, which produces, when 479
 working this time on the dual outranking digraph $-g$ a similar ranking result (see 480
 Dias and Lamboray 2010; Lamboray 2009). 481

Besides of not providing a unique linear ranking, the *ranking-by-choosing* rules, 482
 as well as their duals, the *ordering-by-choosing* rules, are unfortunately not scalable 483
 to larger outranking digraphs (of orders > 100). For such bigger outranking digraphs, 484
 with several hundred or thousands of alternatives, only the COPELAND and the 485
 NETFLOWS *ranking-by-scoring* rules, with a polynomial complexity of $O(n^2)$, 486

where n is the order of the outranking digraph, remain in fact tractable. Furthermore, 487 as computing the COPELAND and NETFLOWS scores may be done separately per 488 alternative, the latter ranking rules can right away be processed in parallel when 489 multiprocessing resources are available. 490

The physical necessity to write down a list of items in a linear sequence renders 491 the ranking decision problem very important in practice. However, a relative rating 492 of such items into performance quantiles classes would be, from the very preference 493 modelling perspective, more expressive and faithful. This is the subject of the 494 following Chap. 9. 495

References

496

- Arrow KJ, Raynaud H (1986) Social choice and multicriterion decision-making. The MIT Press, 497 Cambridge 498
- Bisdorff R (2010) Enumerating chordless circuits in directed graphs. In: ORBEL24-2010, 24th 499 annual conference of the Belgian Operational Research Society (ORBEL aka Sogesci- 500 B.V.W.B.), January 28–29, Liège (BE), Université de Liège (BE), pp 1–12. <http://hdl.handle.net/10993/23926> 501
- Bisdorff R (2012) On measuring and testing the ordinal correlation between bipolar outranking 502 relations. In: Mousseau V, Pirlot M (eds) DAP'2012 from multiple criteria decision aid to 503 preference learning, University of Mons (Belgium), pp 91–100. <http://hdl.handle.net/10993/23909> 504
- Bisdorff R (2013) On polarizing outranking relations with large performance differences. J Multi- 505 Criteria Decis Anal Wiley 20:3–12. <http://hdl.handle.net/10993/245> 506
- Bisdorff R (2021) Technical documentation of the Digraph3 collection of Python modules. <https://digraph3.readthedocs.io/en/latest/techDoc.html> 509
- Brans JP, Vincke P (1985) A preference ranking organisation method: the PROMETHEE method 511 for MCDM. Manag Sci 31(6):647–656 512
- Copeland A (1951) A reasonable social welfare function. Seminar on mathematics in social 513 sciences, University of Michigan 514
- Dias L, Lamboray C (2010) Extensions of the prudence principle to exploit a valued outranking 515 relation. Eur J Oper Res 201(3):828–837 516
- Kemeny J (1959) Mathematics without numbers. Daedalus 88:577–591 517
- Kohler G (1978) Choix multicritère et analyse algébrique de données ordinaires. PhD thesis, 518 Université scientifique et médicale de Grenoble 519
- Lamboray C (2009) A prudent characterization of the Ranked-Pairs rule. Soc Choice Welf 32:129– 520 155 521
- Sen A (2009) Idea of justice. Allen Lane, London 522
- Slater P (1961) Inconsistencies in a schedule of paired comparisons. Biometrika 48:303–312 523
- Tideman N (1987) Independence of clones as a criterion for voting rules. Soc Choice Welf 524 4(3):185–206 525

Chapter 9

Rating by Sorting into Relative Performance Quantiles

1
2
3

Contents	4
9.1 Quantile Sorting on a Single Performance Criterion	115 5
9.2 Sorting into Quantiles with Multiple Performance Criteria	116 6
9.3 The Sparse Pre-ranked Outranking Digraph Model	120 7
9.4 Ranking Pre-ranked Sparse Outranking Digraphs	123 8

Abstract In this chapter, we apply order statistics for sorting a set X of n potential decision alternatives, evaluated on m incommensurable performance criteria, into q quantile equivalence classes. The sorting algorithm is based on pairwise outranking characteristics involving the quantile class limits observed on each criterion. Thus we may implement a weak ordering algorithm of complexity $O(nmq)$. 9 10 11 12 13

9.1 Quantile Sorting on a Single Performance Criterion 14

A single criterion sorting category K is a (usually) lower-closed interval $[m_k; M_k]$ on a real-valued performance measurement scale, with $m_k \leq M_k$. If x is a measured performance on this scale, we may distinguish three sorting situations: 15 16 17

1. $x < m_k$ and $x < M_k$: The performance x is lower than category K . 18
2. $x \geq m_k$ and $x < M_k$: The performance x belongs to category K . 19
3. $x > m_k$ and $x \geq M_k$: The performance x is higher than category K . 20

As the relation $<$ is the dual of \geq (\geq), it will be sufficient to check that $x \geq m_k$ as well as $x \geq M_k$ are true for x to be considered a member of category K . 21 22

Upper-closed categories (in a more mathematical integration style) may as well be considered. In this case it is sufficient to check that $m_k \geq x$ as well as $M_k \geq x$ are true for x to be considered a member of category K . It is worthwhile noticing that a category K such that $m_k = M_k$ is hence always empty by definition. In order to be able to properly sort over the complete range of values to be sorted, we will need to use a special, two-sided closed last, respectively first, category. 23 24 25 26 27 28

Let $K = K_1, \dots, K_q$ be a non-trivial partition of the criterion's performance measurement scale into $q \geq 2$ ordered categories K_k —i.e. lower-closed intervals $[m_k; M_k[$ —such that $m_k < M_k$, $M_k = m_{k+1}$ for $k = 0, \dots, q-1$ and $M_q = \infty$. And, let $A = \{a_1, a_2, a_3, \dots\}$ be a finite set of not all equal performance measures observed on the scale in question.

Property For all performance measure $x \in A$ there exists now a unique k such that $x \in K_k$. If we assimilate, like in descriptive statistics, all the measures gathered in a category K_k to the central value of the category—i.e. $(m_k + M_k)/2$ —the sorting result will hence define a weak order (complete preorder) on A .

Let $Q = \{Q_0, Q_1, \dots, Q_q\}$ denote the set of $q+1$ increasing order-statistical quantiles—like quartiles or deciles—we may compute from the ordered set A of performance measures observed on a performance scale. If $Q_0 = \min(X)$, the following intervals: $[Q_0; Q_1[$, $[Q_1; Q_2[$, ..., $[Q_{q-1}; \infty[$ define a set of q lower-closed sorting categories. And, in the case of upper-closed categories, if $Q_q = \max(X)$, we obtain the intervals $]-\infty; Q_1]$, $]Q_1; Q_2]$, ..., $]Q_{q-1}; Q_q]$. The corresponding sorting of A will result, in both cases, in a repartition of all measures x into the q quantile categories K_k for $k = 1, \dots, q$.

Example Let $A = \{a_7 = 7.03, a_{15} = 9.45, a_{11} = 20.35, a_{16} = 25.94, a_{10} = 31.44, a_9 = 34.48, a_{12} = 34.50, a_{13} = 35.61, a_{14} = 36.54, a_{19} = 42.83, a_5 = 50.04, a_2 = 59.85, a_{17} = 61.35, a_{18} = 61.61, a_3 = 76.91, a_6 = 91.39, a_1 = 91.79, a_4 = 96.52, a_8 = 96.56, a_{20} = 98.42\}$ be a set of 20 increasing performance measures observed on a given criterion. The lower-closed category limits we obtain with quartiles ($q = 4$) are: $Q_0 = 7.03 = a_7$, $Q_1 = 34.485$, $Q_2 = 54.945$ (median performance), and $Q_3 = 91.69$. And the sorting into these four categories defines on A a complete preorder with the following four equivalence classes: $K_1 = \{a_7, a_{10}, a_{11}, a_{10}, a_{15}, a_{16}\}$, $K_2 = \{a_5, a_9, a_{13}, a_{14}, a_{19}\}$, $K_3 = \{a_2, a_3, a_6, a_{17}, a_{18}\}$, and $K_4 = \{a_1, a_4, a_8, a_{20}\}$.

9.2 Sorting into Quantiles with Multiple Performance Criteria

Let us now suppose that we are given a performance tableau with a set X of n decision alternatives evaluated on a coherent family of m performance criteria associated with the corresponding outranking relation \succsim defined on X . We denote x_j the performance of alternative x observed on criterion j .

Suppose furthermore that we want to sort the decision alternatives into q upper-closed quantile equivalence classes. We therefore consider a series: $k = k/q$ for $k = 0, \dots, q$ of $q+1$ equally spaced quantiles, like quartiles: 0, 0.25, 0.5, 0.75, 1; quintiles: 0, 0.2, 0.4, 0.6, 0.8, 1; or deciles: 0, 0.1, 0.2, ..., 0.9, 1, for instance.

The upper-closed \mathbf{q}^k class corresponds to the m quantile intervals $]q_j(p_{k-1}); q_j(p_k)]$ observed on each criterion j , where $k = 2, \dots, q$, $q_j(p_q) = \max_X(x_j)$, and the first class gathers all performances below or equal to $Q_j(p_1)$.

The lower-closed \mathbf{q}_k class corresponds to the m quantile intervals $[q_j(p_{k-1}); q_j(p_k)]$ observed on each criterion j , where $k = 1, \dots, q-1$, $q_j(p_0) = \min_X(x_j)$, and the last class gathers all performances above or equal to $Q_j(p_{q-1})$.

We call **q-tiles** a complete series of $k = 1, \dots, q$ upper-closed \mathbf{q}^k , respectively, lower-closed \mathbf{q}_k , multiple-criteria quantile classes.

Property With the help of the bipolar-valued characteristic of the outranking relation $r(x \succsim y)$ we may compute as follows the bipolar-valued characteristic of the assertion: x belongs to upper-closed q -tiles class \mathbf{q}^k class, respectively, lower-closed class \mathbf{q}_k (Bisdorff 2020):

$$r(x \in \mathbf{q}^k) = \min [-r(\mathbf{q}(p_{q-1}) \succsim x), r(\mathbf{q}(p_q) \succsim x)] \quad (9.1)$$

$$r(x \in \mathbf{q}_k) = \min [r(x \succsim \mathbf{q}(p_{q-1})), -r(x \succsim \mathbf{q}(p_q))] \quad (9.2)$$

The \min operator implements the logical conjunction and, the outranking relation \succsim verifying the coduality principle, $-r(\mathbf{q}(p_{q-1}) \succsim x) = r(\mathbf{q}(p_{q-1}) \prec x)$, respectively, $-r(x \succsim \mathbf{q}(p_q)) = r(x \prec \mathbf{q}(p_q))$.

The `QuantilesSortingDigraph` class from the `sortingDigraphs` module can compute, for instance, such a quintiling of a given random performance tableau (Bisdorff 2021).

Listing 9.1 Computing a quintiles sorting result

```

1  >>> from randomPerfTabs import RandomPerformanceTableau
2  >>> t = RandomPerformanceTableau(numberOfActions=50, seed=5)
3  >>> from sortingDigraphs import QuantilesSortingDigraph
4  >>> qs = QuantilesSortingDigraph(t, limitingQuantiles=5)
5  >>> qs
6  ----- Object instance description -----
7  Instance class   : QuantilesSortingDigraph
8  Instance name    : sorting_with_5-tile_limits
9  Actions          : 50
10 Criteria         : 7
11 Categories        : 5
12 Lowerclosed       : False
13 Size              : 841
14 Valuation domain  : [-1.00;1.00]
15 Determinateness (%) : 81.39
16 Attributes        : ['actions', 'actionsOrig',
17   'criteria', 'evaluation', 'runTimes', 'name',
18   'limitingQuantiles', 'LowerClosed',
19   'categories', 'criteriaCategoryLimits',
20   'profiles', 'profileLimits', 'hasNoVeto',
21   'valuationdomain', 'nbrThreads', 'relation',
22   'categoryContent', 'order', 'gamma', 'notGamma']
23 ----- Constructor run times (in sec.) -----
24 # Threads          : 1
25 Total time         : 0.03120
26 Data input         : 0.00300
27 Compute profiles  : 0.00075
28 Compute relation  : 0.02581
29 Weak Ordering      : 0.00052

```

5-Tiling the 50 decision alternatives results in 5 ordered upper-closed categories 113
 of decision alternatives—the quintiles—supported by a 81.39% majority of criteria 114
 significance (see Line 15 above). 115

The `showCriteriaQuantileLimits()` method shows in Listing 9.2 the 116
 quintile limits separating on each criterion the respective evaluations of the 50 117
 decision alternatives into the 5 quintiles. Highest evaluations are observed on 118
 criterion `g1` (see Line 6 below). 119

Listing 9.2 Inspecting the quantile limits

```

1 >>> qs.showCriteriaQuantileLimits() 120
2     Quantile Class Limits (q = 5) 121
3     Upper-closed classes 122
4     crit.      0.20      0.40      0.60      0.80      1.00 123
5     *----- 124
6     g1      31.35      41.09      58.53      71.91      98.08 125
7     g2      27.81      39.19      49.87      61.66      96.18 126
8     g3      25.10      34.78      49.45      63.97      92.59 127
9     g4      24.61      37.91      53.91      71.02      89.84 128
10    g5      26.94      36.43      52.16      72.52      96.25 129
11    g6      23.94      44.06      54.92      67.34      95.97 130
12    g7      30.94      47.40      55.46      69.04      97.10 131

```

And with the `showSorting()` method we can inspect the actual quintiles 132
 sorting result in Listing 9.3 below. 133

Listing 9.3 Computing a quintiles sorting result

```

1 >>> qs.showSorting() 134
2     *--- Sorting results in descending order ---* 135
3     ]0.80 - 1.00]: ['a22'] 136
4     ]0.60 - 0.80]: ['a03', 'a07', 'a08', 'a11', 'a14', 'a17', 137
5           'a19', 'a20', 'a29', 'a32', 'a33', 'a37', 138
6           'a39', 'a41', 'a42', 'a49'] 139
7     ]0.40 - 0.60]: ['a01', 'a02', 'a04', 'a05', 'a06', 'a08', 140
8           'a09', 'a16', 'a17', 'a18', 'a19', 'a21', 141
9           'a24', 'a27', 'a28', 'a30', 'a31', 'a35', 142
10          'a36', 'a40', 'a43', 'a46', 'a47', 'a48', 143
11          'a49', 'a50'] 144
12     ]0.20 - 0.40]: ['a04', 'a10', 'a12', 'a13', 'a15', 'a23', 145
13           'a25', 'a26', 'a34', 'a38', 'a43', 'a44', 146
14           'a45', 'a49'] 147
15     ] < - 0.20]: ['a44'] 148

```

Most of the decision alternatives (26) are gathered in the median quintile $]0.40 - 149$
 $0.60]$ class, whereas the highest quintile $]0.80 - 1.00]$ and the lowest quintile $] < 150$
 $-0.20]$ classes gather each one a unique decision alternative (`a22`, respectively, 151
`a44`). 152

Inspecting the details of the corresponding sorting characteristics may be done 153
 with the `showSortingCharacteristics()` method. 154

Listing 9.4 Bipolar-valued sorting characteristics (extract)

```

1 >>> qs.showSortingCharacteristics() 155
2   x in q^k           r(q^k-1 < x)  r(q^k >= x)  r(x in q^k) 156
3   a22 in ]< - 0.20]  1.00          -0.86          -0.86 157
4   a22 in ]0.20 - 0.40] 0.86          -0.71          -0.71 158
5   a22 in ]0.40 - 0.60] 0.71          -0.71          -0.71 159
6   a22 in ]0.60 - 0.80] 0.71          -0.14          -0.14 160
7   a22 in ]0.80 - 1.00] 0.14          1.00          0.14 161
8   ...
9   ...
10  a44 in ]< - 0.20]  1.00          0.00          0.00 164
11  a44 in ]0.20 - 0.40] 0.00          0.57          0.00 165
12  a44 in ]0.40 - 0.60] -0.57          0.86          -0.57 166
13  a44 in ]0.60 - 0.80] -0.86          0.86          -0.86 167
14  a44 in ]0.80 - 1.00] -0.86          0.86          -0.86 168
15  ...
16  ...
17  a49 in ]< - 0.20]  1.00          -0.43          -0.43 171
18  a49 in ]0.20 - 0.40] 0.43          0.00          0.00 172
19  a49 in ]0.40 - 0.60] 0.00          0.00          0.00 173
20  a49 in ]0.60 - 0.80] 0.00          0.57          0.00 174
21  a49 in ]0.80 - 1.00] -0.57          0.86          -0.57 175

```

In Listing 9.4 Line 7, alternative a22 verifies indeed positively both sorting conditions only for the highest quintile [0.80–1.00] class. Whereas alternatives a44 and a49, for instance, weakly verify both sorting conditions for two, respectively, three, adjacent quintile classes (Lines 10–11 and 18–20).

Quantiles sorting results, indeed, verify always the following properties (Bisdorff 2020):

Property 9.1 (Formal Properties of a Quantiles Sorting Result)

1. *Coherence*: Each object is sorted into a non-empty subset of *adjacent q-tiles* classes. An alternative that would *miss* evaluations on all the criteria will be sorted conjointly in all *q-tiled* classes.
2. *Uniqueness*: When $r(x \in q^k) \neq 0.0$ for $k = 1, \dots, q$, the performance x is sorted into *exactly one single q-tiled* class.
3. *Separability*: Computing the sorting result for performance x is *independent* from the computing of the other performances' sorting results. This property gives access to efficient parallel processing of class membership characteristics.

The *q-tiles* sorting result leaves us hence with more or less *overlapping* ordered quantile equivalence classes. For constructing now a linearly ranked *q-tiles* partition of X , we may apply three strategies:

1. *Average* (default): In decreasing lexicographic order of the average of the lower and upper quantile limits and the upper quantile class limit;
2. *Optimistic*: In decreasing lexicographic order of the upper and lower quantile class limits;
3. *Pessimistic*: In decreasing lexicographic order of the lower and upper quantile class limits;

Listing 9.5 Weakly ranking the quintiles sorting result

```

1 >>> qs.showQuantileOrdering(strategy='average') 200
2   ] 0.80-1.00] : ['a22'] 201
3   ] 0.60-0.80] : ['a03','a07','a11','a14','a20','a29', 202
4   'a32','a33','a37','a39','a41','a42'] 203
5   ] 0.40-0.80] : ['a08','a17','a19'] 204
6   ] 0.20-0.80] : ['a49'] 205
7   ] 0.40-0.60] : ['a01','a02','a05','a06','a09','a16', 206
8   'a18','a21','a24','a27','a28','a30', 207
9   'a31','a35','a36','a40','a46','a47', 208
10  'a48','a50'] 209
11  ] 0.20-0.60] : ['a04','a43'] 210
12  ] 0.20-0.40] : ['a10','a12','a13','a15','a23','a25', 211
13  'a26','a34','a38','a45'] 212
14  ] < -0.40] : ['a44'] 213

```

Following, for instance, the *average* ranking strategy, we find confirmed in the weak ranking shown in Listing 9.5 that alternative $a49$ is indeed sorted into three adjacent quintiles classes, namely $[0.20-0.80]$ and precedes the $[0.40-0.60]$ class, of same average of lower and upper limits (see Line 6).

The `QuantilesSortingDigraph` class constructor models hence a linearly ordered decomposition of the corresponding bipolar-valued outranking digraph. This decomposition leads us to a new *sparse pre-ranked* outranking digraph model.

9.3 The Sparse Pre-ranked Outranking Digraph Model

Following the methodological requirements of the outranking approach, a given outranking digraph is, necessarily associated with a corresponding performance tableau (see Chap. 3). And, we can use this associated performance tableau for linearly decomposing the set of potential decision alternatives into ordered quantiles equivalence classes by using the quantiles sorting algorithm seen in the previous Section.

In the coding example shown in Listing 9.6 below, we generate, first a simple performance tableau of 75 decision alternatives and, secondly, construct with the `PreRankedOutrankingDigraph` class from `sparseOutrankingDigraphs` module the corresponding sparse outranking digraph called `prg`. Notice by the way in Line 3 the `BigData` flag used for generating a parsimoniously commented performance tableau (Bisdorff 2021).

Listing 9.6 Computing a *pre-ranked* sparse outranking digraph

```

1 >>> from randomPerfTabs import RandomPerformanceTableau 234
2 >>> tp = RandomPerformanceTableau(numberOfActions=75,\ 235
3 ...                                BigData=True,seed=100) 236
4 >>> from sparseOutrankingDigraphs import\ 237
5 ...                                PreRankedOutrankingDigraph 238
6 >>> prg = PreRankedOutrankingDigraph(Tp,quantiles=5) 239
7 >>> prg 240

```

```

8   ----- Object instance description -----
9   Instance class      : PreRankedOutrankingDigraph      241
10  Instance name       : randomperftab_pr                242
11  Actions             : 75                                243
12  Criteria            : 7                                 244
13  Sorting by          : 5-Tiling                         245
14  Ordering strategy   : average                          246
15  Components          : 9                                 247
16  Minimal order       : 1                                 248
17  Maximal order       : 25                               249
18  Average order       : 8.3                             250
19  fill rate           : 20.432%                         251
20  Attributes          :                                252
21      ['actions','criteria','evaluation','NA','name', 253
22      'order','runTimes','dimension','sortingParameters', 254
23      'valuationdomain','profiles','categories','sorting', 255
24      'decomposition','nbrComponents','components', 256
25      'fillRate','minimalComponentSize','maximalComponentSize', 257
26      ... ]                                              258
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259

```

The ordering of the 5-tiling result is following the *average* lower and upper 260
quintile limits strategy (see Listing 9.6 on the facing page Line 14). We obtain 261
here 9 ordered components of minimal order 1 and maximal order 25. With the 262
showDecomposition() method, the corresponding *pre-ranked decomposition* 263
may be inspected as follows: 264

Listing 9.7 The quantiles decomposition of a pre-ranked outranking digraph

```

1 >>> prg.showDecomposition()                                265
2     *--- quantiles decomposition in decreasing order---* 266
3     c1. ]0.80-1.00] : [5, 42, 43, 47]                  267
4     c2. ]0.60-1.00] : [73]                                268
5     c3. ]0.60-0.80] : [1, 4, 13, 14, 22, 32, 34, 35, 40, 269
6                     41, 45, 61, 62, 65, 68, 70, 75]        270
7     c4. ]0.40-0.80] : [2, 54]                            271
8     c5. ]0.40-0.60] : [3, 6, 7, 10, 15, 18, 19, 21, 23, 24, 272
9                     27, 30, 36, 37, 48, 51, 52, 56, 58, 273
10                    63, 67, 69, 71, 72, 74]                274
11     c6. ]0.20-0.60] : [8, 11, 25, 28, 64, 66]          275
12     c7. ]0.20-0.40] : [12, 16, 17, 20, 26, 31, 33, 38, 39, 276
13                     44, 46, 49, 50, 53, 55]                277
14     c8. ] <-0.40] : [9, 29, 60]                         278
15     c9. ] <-0.20] : [57, 59]                           279

```

The highest quintile class (]80% – 100%]) contains decision alternatives 5, 42, 280
43 and 47. Lowest quintile class (] – 20%]) gathers alternatives 57 and 59 (see 281
Listing 9.7 Lines 3 and 15). 282

The resulting sparse outranking relation is shown with the showHTMLRelationMap() 283
method in a heatmap browser view. 284

```

1   >>> prg.showHTMLRelationMap()                           285

```

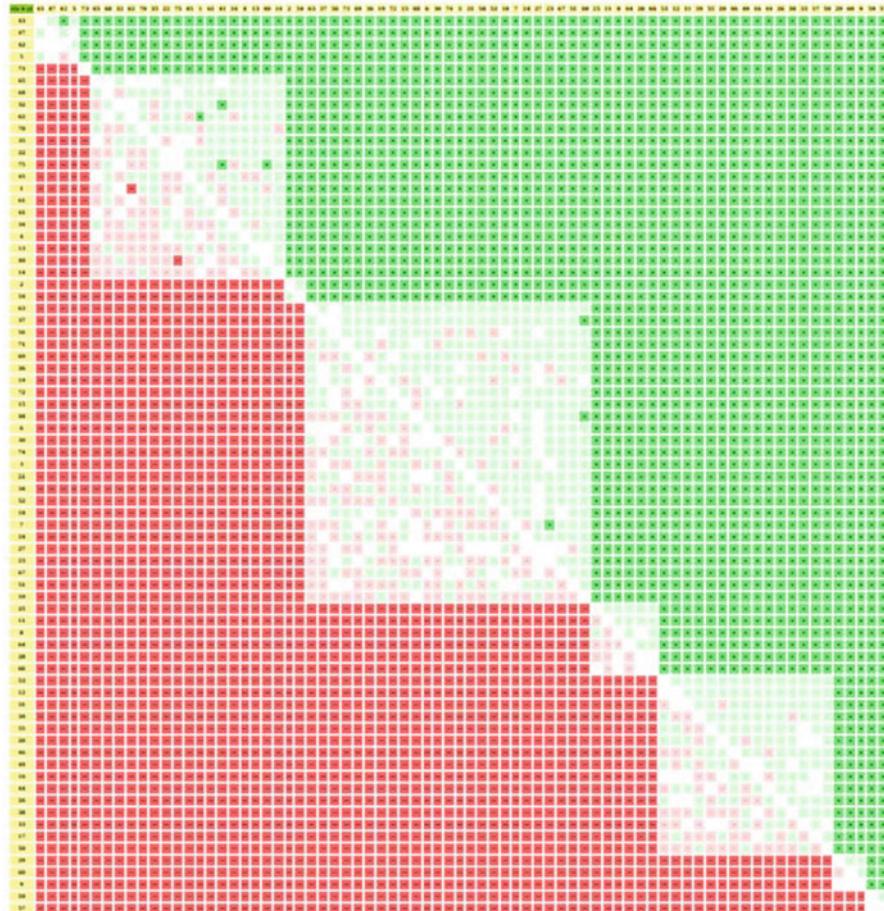


Fig. 9.1 The relation map of a sparse outranking digraph

In Fig. 9.1 we easily recognise the 9 linearly ordered quantile equivalence 286 classes. *Green* and *light-green* pixels show positive *outranking* situations, whereas 287 positive *outranked*—negative *outranking*—situations are shown in *red* and *light-red*. 288 Indeterminate situations appear in white. In each one of the 9 quantile equivalence 289 classes we recover the corresponding bipolar-valued outranking *sub-relation*, which 290 leads to an actual *fill-rate* of 20.4% (see Listing 9.6 on page 120 Line 19). 291

Computing the ordinal correlation between the sparse and the standard outrank- 292 ing digraph allows to check how faithful the sparse model represents the complete 293 outranking relation. 294

```
1 >>> g = BipolarOutrankingDigraph(tp) 295
2 >>> corr = prg.computeOrdinalCorrelation(g) 296
3 >>> g.showCorrelation(corr) 297
4     Correlation indexes: 298
5     Crisp ordinal correlation : +0.863 299
6     Epistemic determination : 0.315 300
7     Bipolar-valued equivalence : +0.272 301
```

The ordinal correlation index between the standard and the sparse outranking digraphs is quite high (+0.863) and their bipolar-valued equivalence is supported by a mean criteria significance majority of $(1.0 + 0.272)/2 = 64\%$. 304

It is worthwhile noticing in Listing 9.6 on page 120 (Lines 20 and following) that sparse pre-ranked outranking digraphs do not contain a `relation` attribute. The access to pairwise outranking characteristic values is here provided via a corresponding `relation()` function.

Listing 9.8 Functional binary relation characteristics

```

1 def relation(self,x,y):
2     """
3         Dynamic construction of the global
4         outranking characteristic function  $r(x,y)$ .
5     """
6     Min = self.valuationdomain['min']
7     Med = self.valuationdomain['med']
8     Max = self.valuationdomain['max']
9     if x == y:
10         return Med
11     cx = self.actions[x]['component']
12     cy = self.actions[y]['component']
13     if cx == cy:
14         return self.components[cx]['subGraph'].relation[x][y]
15     elif self.components[cx]['rank'] >
16         self.components[cy]['rank']:
17         return Min
18     else:
19         return Max

```

In Listing 9.8 Lines 9–10, all reflexive situations are set to the *indeterminate* value. When two decision alternatives belong to a same component—quantile equivalence class—we access the relation attribute of the corresponding outranking sub-digraph (Lines 13–14). Otherwise we just check the respective ranks of the components.

9.4 Ranking Pre-ranked Sparse Outranking Digraphs

Each one of these nine ordered components is locally ranked by using a suitable ranking rule. Best operational results, both in run times and quality, are more or less equally obtained with the COPELAND and the NETFLOWS rules (see Sects. 8.2 and 8.3). The eventually obtained linear ordering (from the worst to best) is stored

in a `prg.boostedOrder` attribute. A reversed linear order (from the best to the worst) is stored in a `prg.boostedRanking` attribute. 338
339

```
1 >>> prg.boostedRanking 340
2 [43, 47, 42, 5, 73, 65, 68, 32, 62, 70, 35, 22, 75, 45, 1, 341
3 61, 41, 34, 4, 13, 40, 14, 2, 54, 63, 37, 56, 71, 69, 36, 342
4 19, 72, 15, 48, 6, 30, 74, 3, 21, 58, 52, 18, 7, 24, 27, 343
5 23, 67, 51, 10, 25, 11, 8, 64, 28, 66, 53, 12, 31, 39, 55, 344
6 20, 46, 49, 16, 44, 26, 38, 33, 17, 50, 29, 60, 9, 59, 57] 345
```

Alternative 43 appears *first-ranked*, whereas alternative 57 is *last ranked* (see 346
above). 347

The quality of this ranking result may be assessed by computing with the 348
`computeRankingCorrelation()` method its ordinal correlation index with 349
the standard outranking digraph `g`. 350

```
1 >>> corr = g.computeRankingCorrelation(prg.boostedRanking) 351
2 >>> g.showCorrelation(corr) 352
3 Correlation indexes: 353
4   Crisp ordinal correlation : +0.807 354
5   Epistemic determination : 0.315 355
6   Bipolar-valued equivalence : +0.254 356
```

One may also verify below that the COPELAND ranking obtained, for instance, 357
from the standard outranking digraph `g` is as well highly correlated (+0.822) with 358
the one obtained from the sparse outranking digraph `prg`. 359

```
1 >>> from linearOrders import CopelandOrder 360
2 >>> cop = CopelandOrder(g) 361
3 >>> print(cop.computeRankingCorrelation(prg.boostedRanking)) 362
4   {'correlation': 0.822, 'determination': 1.0} 363
```

Noticing the computational efficiency of the quantiles sorting construction, coupled 364
with the separability property of the quantile class membership characteristics 365
computation, we will make usage of the `PreRankedOutrankingDigraph` 366
constructor in Chap. 11 for HPC ranking big and even huge performance tableaux 367
(Bisdorff 2016). The next Chap. 10 addresses now the problem of absolute rating 368
into learned quantile performance norms. 369

References

- Bisdorff R (2016) Computing linear rankings from trillions of pairwise outranking situations. 371
In: Busa-Fekete R, Hüllermeier E, Mousseau V, Pfannschmidt K (eds) From multiple criteria 372
decision aid to preference learning, DAP'2016, University of Paderborn (Germany), pp 1–16. 373
<http://hdl.handle.net/10993/28613> 374
- Bisdorff R (2020) Lecture 10: On quantiles rating with multiple incommensurable criteria. In: 375
Lectures of the algorithmic decision theory course, University of Luxembourg. <http://hdl.handle.net/10993/37933> 376
377
- Bisdorff R (2021) Technical documentation of the Digraph3 collection of Python modules. <https://digraph3.readthedocs.io/en/latest/techDoc.html> 378
379

Chapter 10

Rating-by-Ranking with Learned Performance Quantile Norms

Contents	4
10.1 The Absolute Rating Problem	125 5
10.2 Incremental Learning of Historical Performance Quantiles	126 6
10.3 Rating-by-Ranking New Performances with Quantile Norms	129 7

Abstract We address in this chapter the problem of rating multiple-criteria performances of a set of potential decision alternatives with respect to performance quantiles learned from historical performance data gathered from similar decision alternatives observed in the past. We show how to learn performance quantiles from such historical performance tableaux. New performance records may now be rated with respect to these quantile norms. 8
9
10
11
12
13

10.1 The Absolute Rating Problem

14

To illustrate the *absolute rating* problem we face, consider for a moment that, in a 15 given decision making problem we observe, for instance, in Table 10.1 below, the 16 multi-criteria performance evaluations of two potential decision alternatives, named 17 $a1001$ and $a1010$, evaluated on 7 *incommensurable* performance criteria: 2 *Costs* 18 criteria $c1$ and $c2$ (to *minimise*) and 5 *Benefits* criteria $b1$ to $b5$ (to *maximise*). 19

The performance on *Benefits* criteria $b1$, $b4$, and $b5$ is measured on a cardinal 20 scale from 0.0 (worst) to 100.0 (best) whereas, the performance on the *Benefits* 21 criteria $b2$ and $b3$ and on the *Costs* criterion $c1$ is measured on an ordinal scale 22 from 0 (worst) to 10 (best), respectively, -10 (worst) to 0 (best). The performance 23

Table 10.1 Multi-criteria performances of two potential decision alternatives

Criterion (weight)	$b1$ (2)	$b2$ (2)	$b3$ (2)	$b4$ (2)	$b5$ (2)	$c1$ (5)	$c2$ (5)	t3.1
$a1001$	37.0	2	2	61.0	31.0	-4	-40.0	t3.2
$a1010$	32.0	9	6	55.0	51.0	-4	-35.0	t3.3

on the *Costs* criterion c2 is eventually measured on a cardinal *negative* scale from 24
 –100.00 (worst) to 0.0 (best). The two *Costs* criteria are equi-significant (weight 25
 5). Similarly, the five Benefits criteria are also equi-significant (weight 2). The 26
importance (sum of significance weights: $2 \times 5 = 10$) of the *Costs* criteria is hence 27
equivalent to the importance (sum of significance weights: $5 \times 2 = 10$) of the 28
Benefits criteria. 29

The non-trivial decision problem we now face, is to decide, how the previous two 30
 multiple-criteria performance records of alternatives a1001, respectively, a1010, 31
 may be rated: *excellent?* *good?*, or *fair?*; perhaps even, *weak?* Or *very weak?* When 32
 compared with similar multi-criteria performance records one has already rated into 33
 quantiles in the past. 34

To solve this *absolute* rating problem, first, we need to estimate multi-criteria 35
performance quantiles from historical records (Bisdorff 2020). 36

10.2 Incremental Learning of Historical Performance 37 Quantiles 38

Suppose that we see flying in random multiple-criteria performances from a given 39
 model of random performance tableau (see Chap. 5). The question we address here 40
 is to estimate empirical performance quantiles on the basis of so far observed 41
 performance vectors. For this task, we are inspired by Chambers et al. (2006), 42
 who present an efficient algorithm for incrementally updating a quantile-binned 43
 cumulative distribution function (CDF) with newly observed CDFs.¹ 44

The `PerformanceQuantiles` class, using the `IncrementalQuantilesEstimator` class from the `randomNumbers` module, implements such 45
 a performance quantiles estimation based on a given performance tableau (Bisdorff 46
 2021). 47

Its main attributes are: 49

- Ordered `objectives` and a `criteria` dictionaries from a valid performance 50
 tableau instance;
- A list `quantileFrequencies` of quantile frequencies like: 51
 - `quartiles` [0.0, 0.25, 0.5, 0.75, 1.0], 53
 - `quintiles` [0.0, 0.2, 0.4, 0.6, 0.8, 1.0], or 54
 - `deciles` [0.0, 0.1, 0.2, ...1.0] for instance; 55
- An ordered dictionary `limitingQuantiles` of so far estimated *lower* 56
 (default) or *upper* quantile class limits for each frequency per criterion; 57

¹ We have adapted in Python a C++ implementation published by Press et al. (2007, Chap. 5).

- An ordered dictionary `historySizes` for keeping track of the number of evaluations seen so far per criterion. Missing data, the case given, make these sizes vary from criterion to criterion. 58
59
60

Below, we show an example Python session concerning 900 decision alternatives 61 randomly generated with a *Cost-Benefit* performance tableau model (see Sect. 5.3) 62 from which are also drawn the performances of alternatives `a1001` and `a1010` 63 shown in Table 10.1 on page 125 above. 64

Listing 10.1 Computing performance quantiles from a given performance tableau

```

1  >>> from performanceQuantiles import PerformanceQuantiles      65
2  >>> from randomPerfTabs import RandomCBPerformanceTableau      66
3  >>> nbrActions=900                                         67
4  >>> nbrCrit = 7                                         68
5  >>> seed = 100                                         69
6  >>> pt = RandomCBPerformanceTableau(numberOfActions=nbrActions, \ 70
7  ...                                         numberOfCriteria=nbrCrit, seed=seed) 71
8  >>> pq = PerformanceQuantiles(pt,\ 72
9  ...                                         numberOfBins = 'quartiles',\ 73
10 ...                                         LowerClosed=True) 74
11 >>> pq 75
12 *----- PerformanceQuantiles instance description -----* 76
13 Instance class      : PerformanceQuantiles 77
14 Instance name       : 4-tiled_performances 78
15 Objectives          : 2 79
16 Criteria            : 7 80
17 Quantiles           : 4 81
18 History sizes       : {'c1': 887, 'b1': 888, 'b2': 891, 'b3': 895, 82
19 ...                                         'b4': 892, 'c2': 893, 'b5': 887} 83
20 Attributes          : ['perfTabType', 'valueDigits', 84
21 ...                                         'actionsTypeStatistics', 85
22 ...                                         'objectives', 'BigData', 86
23 ...                                         'missingDataProbability', 87
24 ...                                         'criteria', 'LowerClosed', 'name', 88
25 ...                                         'quantilesFrequencies', 'historySizes', 89
26 ...                                         'limitingQuantiles', 'cdf'] 90

```

In Line 10 above, the `PerformanceQuantiles` class parameter `numberOfBins` we use for setting the wished number of quantile frequencies, may be either *quartiles* (4 bins), *quintiles* (5 bins), *deciles* (10 bins), *dodeciles* (20 bins) or any other integer number of quantile bins. The quantile bins may be either *lower-closed* (default) or *upper-closed*. 91
92
93
94
95

Inspecting the estimated quartile limits may be operated with the `showLimitingQuantiles()` method. 96
97

Listing 10.2 Printing out the estimated quartile limits

```

1  >>> pq.showLimitingQuantiles(ByObjectives=True) 98
2  ---- Historical performance quantiles -----* 99
3  Costs 100
4  criteria | weights | '0.00' '0.25' '0.50' '0.75' '1.00' 101

```

5	---	---	---	---	---	---	---	102
6	'c1'	5	-10	-7	-5	-3	0	103
7	'c2'	5	-96.37	-70.65	-50.10	-30.00	-1.43	104
8	Benefits							105
9	criteria	weights	'0.00'	'0.25'	'0.50'	'0.75'	'1.00'	106
10	---	---	---	---	---	---	---	107
11	'b1'	2	1.99	29.82	49.44	70.73	99.83	108
12	'b2'	2	0	3	5	7	10	109
13	'b3'	2	0	3	5	7	10	110
14	'b4'	2	3.27	30.10	50.82	70.89	98.05	111
15	'b5'	2	0.85	29.08	48.55	69.98	97.56	112

Both objectives are *equally important*; the sum of weights (10) of the *Costs* criteria 113 balance the sum of weights (10) of the *Benefits* criteria (see Listing 10.2 on the 114 preceding page column 2). The preference direction of the *Costs* criteria c1 and c2 115 is *negative*; the lesser the costs, the better it is, whereas all the *Benefits* criteria b1 116 to b5 show *positive* preference directions, i.e. the higher the benefits, the better it is. 117 The columns entitled 0.00, respectively, 1.00 show the *quartile* Q0, respectively, 118 Q4, i.e. the *worst*, respectively, *best* performance observed so far on each criterion. 119 Column 0.50 shows the *median* (Q2) performance observed on the criteria. 120

New decision alternatives with random multiple-criteria performance 121 vectors from the same random performance tableau model as pt (see 122 Listing 10.1 on the previous page) may now be generated with a generic 123 RandomPerformanceGenerator class from the randomPerfTabs module 124 (Bisdorff 2021).² 125

Listing 10.3 Generating 100 new random decision alternatives of the same model

```
1 >>> from randomPerfTabs import RandomPerformanceGenerator 126
2 >>> rpg = RandomPerformanceGenerator(pt, seed=seed) 127
3 >>> newTab = rpg.randomPerformanceTableau(100) 128
```

The so far estimated historical quantile limits must, first, be updated with this 129 newly arriving 100 data records: 130

```
1 >>> # Updating the quartile norms shown above 131
2 >>> pq.updateQuantiles(newTab, historySize=None) 132
```

Parameter historySize of the updateQuantiles() method (Line 2 above) 133 allows to *balance* the new evaluations against the *historical* ones. 134

With historySize = None (the default setting), the balance in the example 135 above is 900/1000 (90%, the weight of historical data) against 100/1000 136 (10%, the weight of the new incoming observations). Setting historySize 137 = 0, for instance, will ignore all historical data (0/100 against 100/100) and 138 restart building the quantile estimation with solely the new incoming data. The 139

² The RandomPerformanceGenerator class works for the *standard* performance tableau model (see Sect. 5.2), the *Cost-Benefit* model (see Sect. 5.3), and the 3-objectives model (see Sect. 5.4).

Performance quantiles

Sampling sizes between 986 and 995.

criterion	0.00	0.25	0.50	0.75	1.00
b1	1.99	28.77	49.63	75.27	99.83
b2	0.00	2.94	4.92	6.72	10.00
b3	0.00	2.90	4.86	8.01	10.00
b4	3.27	35.91	58.58	72.00	98.05
b5	0.85	32.84	48.09	69.75	99.00
c1	-10.00	-7.35	-5.39	-3.38	0.00
c2	-96.37	-72.22	-52.27	-33.99	-1.43

Fig. 10.1 Showing updated quartiles limits per criterion. The 0.25 column shows the first quartile (Q1) limits, the 0.50 column shows the second quartile (Q2) limits and the 0.75 column shows the third quartile (Q3) limits. Column 0.00 (respectively, 1.00) shows the minimum (respectively, maximum) performance on each criterion

showHTMLLimitingQuantiles() method shows the updated quantile limits in a browser view (see Fig. 10.1). 140
141

```
1 >>> # showing the updated quantile limits in a browser view 142
2 >>> pq.showHTMLLimitingQuantiles(Transposed=True) 143
```

10.3 Rating-by-Ranking New Performances with Quantile Norms

144
145

For *rating* a newly given set of decision alternatives with the help of empirical performance quantiles estimated from historical data, the `sortingDigraphs` module provides the `LearnedQuantilesRatingDigraph` class, a specialisation of the `QuantilesSortingDigraph` class (see Chap. 9). The absolute rating result is computed by *ranking* the new performance records together with the learned historical quantile limits. 146
147
148
149
150
151

The class constructor requires a valid `PerformanceQuantiles` instance and, by default, uses the COPELAND or the NETFLOWS ranking rule, whichever fits best in an ordinal correlation sense with the underlying outranking digraph. 152
153
154

It is important to notice that the `LearnedQuantilesRatingDigraph` class, contrary to the generic `OutrankingDigraph` class, does not inherit from the generic `PerformanceTableau` class, but instead from the `PerformanceQuantiles` class (Bisdorff 2021). 155
156
157
158

The `actions` attribute in such a `LearnedQuantilesRatingDigraph` class instance contains not only the newly given decision alternatives, but also the historical quantile limits (attribute `profiles`) obtained from a given `PerformanceQuantiles` class instance, i.e. estimated quantile bins' performance limits from historical performance data. 159
160
161
162
163

We reconsider now the `PerformanceQuantiles` object instance `pq` as 164
 computed in the previous section. Let `newActions` be a list of 10 new decision 165
 alternatives generated with the same random performance tableau generator `rpg` 166
 and including, for our didactic purpose, the two decision alternatives `a1001` and 167
`a1010` mentioned at the beginning. 168

Listing 10.4 Computing the absolute rating of 10 new decision alternatives

```

1 >>> from sortingDigraphs import LearnedQuantilesRatingDigraph 169
2 >>> newActions = rpg.randomActions(10) 170
3 >>> lqr = LearnedQuantilesRatingDigraph(pq,newActions,\ 171
4 ...                                         rankingRule='best') 172
5 >>> lqr 173
6     *---- Object instance description 174
7     Instance class      : LearnedQuantilesRatingDigraph 175
8     Instance name       : normedRatingDigraph 176
9     Criteria           : 7 177
10    Quantile profiles  : 4 178
11    Lower-closed bins  : True 179
12    New actions        : 10 180
13    Size                : 93 181
14    Determinateness (%) : 76.1 182
15    Ranking rule        : Copeland 183
16    Ordinal correlation : +0.95 184
17    Attributes: ['runTimes','objectives','criteria', 185
18      'LowerClosed','quantilesFrequencies', 186
19      'limitingQuantiles','historySizes','cdf','name', 187
20      'newActions','evaluation','categories', 188
21      'criteriaCategoryLimits','profiles','profileLimits', 189
22      'hasNoVeto','actions','completeRelation','relation', 190
23      'concordanceRelation','valuationdomain','order', 191
24      'gamma','notGamma','rankingRule','rankingCorrelation', 192
25      'rankingScores','actionsRanking','ratingCategories', 193
26      'ratingRelation','relationOrig'] 194

```

Data input to the `LearnedQuantilesRatingDigraph` class constructor 195
 is a valid `PerformanceQuantiles` object `pq` and a `newActions` list of 196
 newly generated decision alternatives with the same random generator `rpg` (see 197
 Listing 10.4 Lines 3–4). 198

The `actionsSubset` parameter of the `showPerformanceTableau()` 199
 method allows a look at the digraph's nodes, here called `newActions`. 200

Listing 10.5 Performance tableau of the new incoming decision alternatives

```

1 >>> lqr.showPerformanceTableau(actionsSubset=lqr.newActions) 201
2     *---- performance tableau -----* 202
3     criteria | a1001 a1002 a1003 a1004 a1005 a1006 a1007 a1008 a1009 a1010 203
4     -----|----- 204
5     'b1' | 37.0 27.0 24.0 16.0 42.0 33.0 39.0 64.0 42.0 32.0 205
6     'b2' | 2.0 5.0 8.0 3.0 3.0 3.0 6.0 5.0 4.0 9.0 206
7     'b3' | 2.0 4.0 2.0 1.0 6.0 3.0 2.0 6.0 6.0 6.0 207
8     'b4' | 61.0 54.0 74.0 25.0 28.0 20.0 20.0 49.0 44.0 55.0 208
9     'b5' | 31.0 63.0 61.0 48.0 30.0 39.0 16.0 96.0 57.0 51.0 209
10    'c1' | -4.0 -6.0 -8.0 -5.0 -1.0 -5.0 -1.0 -6.0 -6.0 -4.0 210
11    'c2' | -40.0 -23.0 -37.0 -37.0 -24.0 -27.0 -73.0 -43.0 -94.0 -35.0 211

```

Among the 10 new incoming decision alternatives, we recognise alternatives 212
 a1001 (see column 2) and a1010 (see last column) we have shown in Table 10.1 213
 on page 125. 214

The actions dictionary of a LearnedQuantilesRatingDigraph class 215
 instance includes, besides the 10 performance evaluations of the ten new alterna- 216
 tives, also the closed lower limits of the four quartile classes: m1 = [0.0 – [, m2 = 217
 [0.25 – [, m3 = [0.5 – [, m4 = [0.75 – [. We find these limits in the profiles 218
 attribute (see Listing 10.6 below). 219

Listing 10.6 Showing the limiting profiles of the rating quantiles

```

1 >>> lqr.showPerformanceTableau(actionsSubset=lqr.profiles) 220
2     *---- Quartiles limit profiles ----*
3     criteria | 'm1'   'm2'   'm3'   'm4'
4     -----|-----
5     'b1'   | 2.0    28.8   49.6   75.3 224
6     'b2'   | 0.0    2.9    4.9    6.7   225
7     'b3'   | 0.0    2.9    4.9    8.0   226
8     'b4'   | 3.3    35.9   58.6   72.0 227
9     'b5'   | 0.8    32.8   48.1   69.7 228
10    'c1'  | -10.0  -7.4   -5.4   -3.4 229
11    'c2'  | -96.4 -72.2  -52.3  -34.0 230

```

The main run time is spent by the LearnedQuantilesRatingDigraph 231
 class constructor in computing a bipolar-valued outranking relation on the extended 232
 actions set including both the new alternatives as well as the quartile class limits. In 233
 case of large volumes, i.e. many new decision alternatives and centile classes, for 234
 instance, a multi-threading version may be used when multiple processing cores are 235
 available (Bisdorff 2021). 236

The actual rating procedure will rely on a complete ranking of the new decision 237
 alternatives as well as the quantile class limits obtained from the corresponding 238
 bipolar-valued outranking digraph. Two efficient and scalable ranking rules, the 239
 COPELAND and its valued version, the NETFLOWS rule may be used for this 240
 purpose. The rankingRule parameter allows to choose one of both. With 241
 rankingRule='best' (see Listing 10.4 on the facing page Line 4) the 242
 LearnedQuantilesRatingDigraph constructor will choose the ranking rule 243
 that results in the highest ordinal correlation with the given outranking relation (see 244
 Chap. 16 and Bisdorff 2012). 245

In this rating example, the COPELAND rule appears to be the more appropriate 246
 ranking rule. 247

Listing 10.7 COPELAND ranking of new alternatives and historical quartile limits

```

1 >>> lqr.rankingRule 248
2   'Copeland' 249
3 >>> lqr.actionsRanking 250
4   ['m4', 'a1005', 'a1010', 'a1002', 'a1008', 'a1006', 'a1001', 251
5   'a1003', 'm3', 'a1007', 'a1004', 'a1009', 'm2', 'm1'] 252
6 >>> lqr.showCorrelation(lqr.rankingCorrelation) 253
7   Correlation indexes: 254
8     Crisp ordinal correlation : +0.945 255
9     Epistemic determination   : 0.522 256
10    Bipolar-valued equivalence : +0.493 257

```

We achieve here in Listing 10.7 on the previous page a linear ranking without ties (from best to worst) of the digraph's actions set, i.e. including the new decision alternatives as well as the quartile limits $m1$ to $m4$, which is very close in an ordinal sense ($\tau = 0.945$) to the underlying strict outranking relation. 258
259
260
261

The eventual rating procedure is based in this example on the *lower* quartile 262 limits, such that the quartile contents are filtered out in increasing order of the 263 *quartiles*. 264

```

1 >>> lqr.ratingCategories
2 OrderedDict([
3     ('m2', ['a1007', 'a1004', 'a1009']),
4     ('m3', ['a1005', 'a1010', 'a1002', 'a1008',
5          'a1006', 'a1001', 'a1003'])
6 ])

```

We notice above that no new decision alternative is actually Rated into the lowest 271 [0.0 – 0.25[, respectively, highest [0.75 – [quartile. Indeed, the absolute rating result 272 is shown with the `showQuantilesRating()` method: 273

Listing 10.8 Absolute quartiles rating result

```

1 >>> lqr.showQuantilesRating()
2 ----- Quartiles rating result -----
3 [0.50 - 0.75[ ['a1005', 'a1010', 'a1002', 'a1008',
4          'a1006', 'a1001', 'a1003']]
5 [0.25 - 0.50[ ['a1007', 'a1004', 'a1009']]

```

The same result may also be shown in a browser view with the `showHTMLRatingHeatmap()` method using a specialised rating heatmap format (see Fig. 10.2 on the facing page): 279
280
281

```

1 >>> lqr.showHTMLRatingHeatmap(
2 ...             pageTitle='Heatmap of Quartiles Rating', \
3 ...             Correlations=True, colorLevels=5)

```

Using furthermore a specialised version of the `exportGraphViz()` method 285 allows drawing in Fig. 10.3 on page 134 the same rating result in a HASSE diagram 286 format.³ 287

```

1 >>> lqr.exportRatingGraphViz('quartileRatingDigraph')
2 ----- exporting a dot file for GraphViz tools -----
3 Exporting to quartileRatingDigraph.dot
4 dot -Grankdir=TB -Tpng quartileRatingDigraph.dot -o
      quartileRatingDigraph.png

```

We have actually solved now the *absolute rating* problem stated at the beginning. 293 Decision alternatives $a1001$ and $a1010$ (see Table 10.1 on page 125) are both 294 rated into the same third quartile Q3 class (see Fig. 10.3 on page 134), even if the 295 COPELAND ranking, obtained from the underlying strict outranking digraph (see 296

³ Note that the graphviz dot file was post-edited in order to mark in blue alternatives $a1001$ and $a1010$.

Fig. 10.2 Heatmap of absolute quartiles ranking.

The quartile equivalence classes appear lower-closed. No alternative is rated into the Q1 class ([0.00 – 0.25[) and no alternative is rated into the Q4 class ([0.75 – 1.00])

Heatmap of Quartiles Rating

Ranking rule: Copeland; Ranking correlation: 0.938

criteria	c2	b3	c1	b4	b1	b2	b5
weights	5	2	5	2	2	2	2
tau(*)	+0.64	+0.54	+0.43	+0.37	+0.37	+0.35	+0.34
[0.75 -	30.00	7.00	-3.00	70.89	70.73	7.00	69.98
a1005c	-24.00	6.00	-1.00	28.00	42.00	3.00	30.00
a1010n	-35.00	6.00	-4.00	55.00	32.00	9.00	51.00
a1002c	-23.00	4.00	-6.00	54.00	27.00	5.00	63.00
a1008n	-43.00	6.00	-6.00	49.00	64.00	5.00	96.00
a1006c	-27.00	3.00	-5.00	20.00	33.00	3.00	39.00
a1001c	-40.00	2.00	-4.00	61.00	37.00	2.00	31.00
a1003a	-37.00	2.00	-8.00	74.00	24.00	8.00	61.00
[0.50 -	-50.10	5.00	-5.00	50.82	49.44	5.00	48.55
a1007c	-73.00	2.00	-1.00	20.00	39.00	6.00	16.00
a1004c	-37.00	1.00	-5.00	25.00	16.00	3.00	48.00
a1009n	-94.00	6.00	-6.00	44.00	42.00	4.00	57.00
[0.25 -	-70.65	3.00	-7.00	30.10	29.82	3.00	29.08
[0.00 -	-96.37	0.00	-10.00	3.27	1.99	0.00	0.85

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%
----------	--------	--------	--------	--------	---------

(*) tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation.

Fig. 10.2), suggests that alternative a1010 is effectively better performing than alternative a1001.

A preciser rating result may indeed be achieved when using deciles instead of quartiles for estimating the historical marginal cumulative distribution functions.

Listing 10.9 Absolute deciles rating result

```

1  >>> pq1 = PerformanceQuantiles(pt, numberOfBins='deciles', \
2 ...                               LowerClosed=True)
3  >>> pq1.updateQuantiles(newTab, historySize=None)
4  >>> lqr1 = LearnedQuantilesRatingDigraph(pq1,newActions, \
5 ...                                         rankingRule='best')
6  >>> lqr1.showQuantilesRating()
7  *----- Deciles rating result -----*
8  [0.60 - 0.70[ ['a1005', 'a1010', 'a1008', 'a1002']
9  [0.50 - 0.60[ ['a1006', 'a1001', 'a1003']
10 [0.40 - 0.50[ ['a1007', 'a1004']
11 [0.30 - 0.40[ ['a1009']

```

Compared with the quartiles rating result, we notice in Listing 10.9 that the seven alternatives (a1001, a1002, a1003, a1005, a1006, a1008, and a1010), rated before into the third quartile class [0.00 – 0.25[, are now divided up: alternatives a1002, a1005, a1008, and a1010 attain now the 7th decile class [0.60 – 0.70[, whereas alternatives a1001, a1003 and a1006 attain only the 6th decile class [0.50 – 0.60[. Of the three Q2 [0.25 – 0.50[rated alternatives (a1004, a1007 and a1009) attain the 8th decile class [0.40 – 0.50[. The three Q3 [0.50 – 0.70[rated alternatives (a1001, a1002 and a1003) attain the 9th decile class [0.30 – 0.40[.

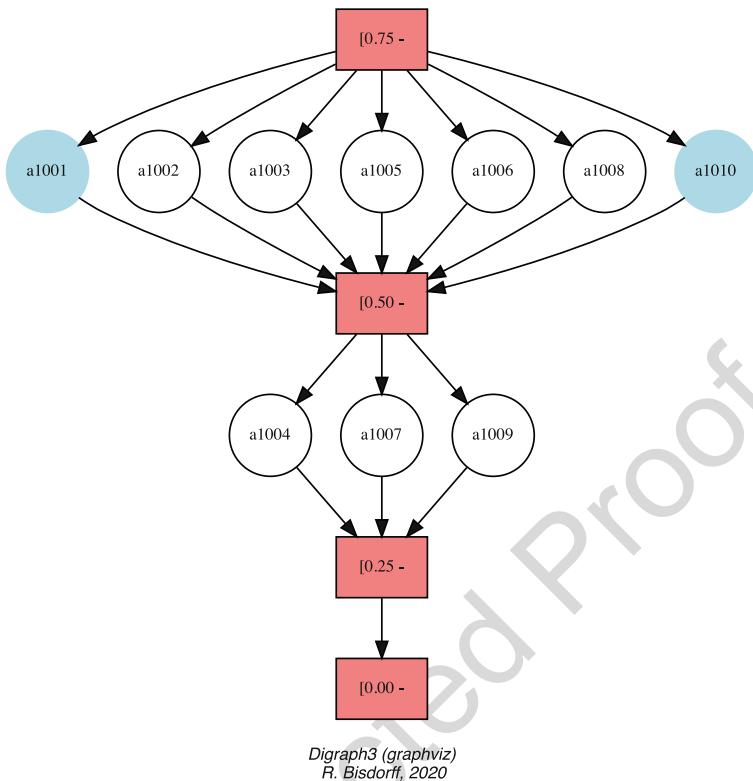


Fig. 10.3 Absolute quartiles rating digraph

and a1009), alternatives a1004 and a1007 are now rated into the 5th decile class 318
 [0.40 – 0.50[and a1009 is lowest rated into the 4th decile class [0.30 – 0.40[. 319

A browser heatmap view in Fig. 10.4 on the facing page more conveniently 320
 illustrates this refined rating result. 321

```

1 >>> lqr1.showHTMLRatingHeatmap (\ 322
2 ...     pageTitle='Heatmap of Deciles rating', \ 323
3 ...     colorLevels=5, Correlations=True) 324

```

To avoid having to recompute performance deciles from historical data when 325
 wishing to refine a rating result, it is useful, depending on the actual size of the 326
 historical data, to initially compute performance quantiles with a relatively high 327
 number of bins, for instance, *dodeciles* or even *centiles*. It is then possible to 328
 interpolate on the fly *quartiles* or *deciles*, for instance, when constructing the rating 329
 digraph. 330

Fig. 10.4 *Absolute deciles rating.* Decision alternatives $a1001$ and $a1010$ are now, as expected, rated in the 6th decile (D6), respectively, in the 7th decile (D7)

Heatmap of Deciles rating

Ranking rule: NetFlows; Ranking correlation: **0.960**

criteria	c2	b3	c1	b1	b5	b2	b4
weights	5	2	5	2	2	2	2
tau(*)	0.67	0.65	0.58	0.57	0.53	0.53	0.48
[0.90 -	20.32	7.73	-2.53	86.83	82.16	7.66	82.04
[0.80 -	29.70	7.26	-3.35	79.30	75.15	6.64	74.66
[0.70 -	37.97	6.67	-4.14	70.95	60.20	5.88	69.76
a1005c	-24.00	6.00	-1.00	42.00	30.00	3.00	28.00
a1010n	-35.00	6.00	-4.00	32.00	51.00	9.00	55.00
a1008n	-43.00	6.00	-6.00	64.00	96.00	5.00	49.00
a1002c	-23.00	4.00	-6.00	27.00	63.00	5.00	54.00
[0.60 -	44.23	5.92	-5.04	60.56	56.01	5.37	62.23
a1006c	-27.00	3.00	-5.00	33.00	39.00	3.00	20.00
a1001c	-40.00	2.00	-4.00	37.00	31.00	2.00	61.00
a1003a	-37.00	2.00	-8.00	24.00	61.00	8.00	74.00
[0.50 -	52.22	4.64	-6.02	49.56	48.07	4.83	58.45
a1007c	-73.00	2.00	-1.00	39.00	16.00	6.00	20.00
a1004c	-37.00	1.00	-5.00	16.00	48.00	3.00	25.00
[0.40 -	60.50	3.84	-6.69	39.61	40.16	4.25	49.82
a1009n	-94.00	6.00	-6.00	42.00	57.00	4.00	44.00
[0.30 -	67.14	3.12	-7.32	30.85	34.33	3.30	40.89
[0.20 -	77.07	2.55	-7.94	23.84	29.57	2.27	30.45
[0.10 -	83.04	1.99	-8.48	16.64	16.91	1.58	24.78
[0.00 -	96.37	0.00	-10.00	1.99	0.85	0.00	3.27

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%
----------	--------	--------	--------	--------	---------

(*) tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation.

Listing 10.10 From deciles interpolated quartiles rating result

```

1 >>> lqr2 = LearnedQuantilesRatingDigraph(pq1,newActions,
2 ...                               quantiles='quartiles')
3 >>> lqr2.showQuantilesRating()
4 ----- Deciles rating result -----
5 [0.50 - 0.75[ ['a1005', 'a1010', 'a1002', 'a1008',
6   'a1006', 'a1001', 'a1003']
7 [0.25 - 0.50[ ['a1004', 'a1007', 'a1009']

```

331
332
333
334
335
336
337

With the quantiles parameter (see Listing 10.10 Line 2), we recover by 338
interpolation the same quartiles rating as obtained directly with historical perfor- 339
mance quartiles (see Listing 10.8 on page 132). Mind that a correct interpolation 340
of quantiles from a given cumulative distribution function requires more or less 341
uniform distributions of observations in each bin. 342

More generally, in the case of industrial production monitoring problems, for 343
instance, where large volumes of historical performance data may be available, 344
it can be of interest to estimate even more precisely the marginal cumulative 345
distribution functions, especially when *tail* rating results, i.e. distinguishing *very* 346

best, or *very weak* multiple-criteria performances, become a critical issue. Similarly, the `historySize` parameter may be used for monitoring on the fly *unstable* random multiple-criteria performance data (Chambers et al. 2006).

The next Chap. 11 is tackling the challenging problem of ranking big performance tableaux with thousands and millions of multiple-criteria records on HPC clusters.

References

353

- Bisdorff R (2012) On measuring and testing the ordinal correlation between bipolar outranking relations. In: Mousseau V, Pirlot M (eds) DAP'2012 from multiple criteria decision aid to preference learning, University of Mons (Belgium), pp 91–100. <http://hdl.handle.net/10993/23909> 354
355
356
357
- Bisdorff R (2020) Lecture 5: Simulating from arbitrary empirical random distributions. In: Lectures of the computational statistics course, University of Luxembourg. <http://hdl.handle.net/10993/37870> 358
359
360
- Bisdorff R (2021) Technical documentation of the Digraph3 collection of Python modules. <https://digraph3.readthedocs.io/en/latest/techDoc.html> 361
362
- Chambers J, James D, Lambert D, Vander Wiel S (2006) Monitoring networked applications with incremental quantile estimation. *Stat Sci* 21(4):463–475 363
364
- Press W, Teukolsky S, Vetterling W, Flannery B (2007) Single-pass estimation of arbitrary quantiles. In: Numerical recipes: the art of scientific computing, Sect 5.8.2, 3rd edn. Cambridge University Press, Cambridge, pp 435–438 365
366
367

Chapter 11	1
HPC Ranking of Big Performance	2
Tableaux	3

Contents	4
11.1 C-compiled Python Modules	137 5
11.2 Big Data Performance Tableaux	138 6
11.3 C-implemented Integer-Valued Outranking Digraphs	139 7
11.4 The Sparse Implementation of Big Outranking Digraphs	141 8
11.5 Quantiles Ranking of Big Performance Tableaux	144 9
11.6 HPC Quantiles Ranking Records	147 10

Abstract The sparse outranking digraph, introduced in Chap. 9, is suitable for 11
 tackling the ranking of big multiple-criteria performance tableaux with thousands 12
 or millions of records. To effectively compute rankings from performance tableaux 13
 of these sizes, we propose in this chapter a collection of C-compiled and optimised 14
 DIGRAPH3 modules that may be run on HPC equipment as available, for instance, 15
 at the University of Luxembourg. 16

11.1 C-compiled Python Modules 17

The DIGRAPH3 collection provides cythonized,¹ i.e. C-compiled and optimised ver- 18
 sions of the main python modules for tackling multiple-criteria decision problems 19
 facing very large sets of decision alternatives ($>10,000$). Such problems appear 20
 usually with a combinatorial organisation of the potential decision alternatives, as 21
 is frequently the case in bioinformatics, for instance. If HPC facilities with nodes 22
 supporting numerous cores (>20) and big RAM (>50 GB) are available, ranking up 23
 to several millions of alternatives becomes effectively tractable (Bisdorff 2016). 24

Four cythonized DIGRAPH3 modules, prefixed with the letter `c` and taking a 25
`.pyx` extension, are provided with their corresponding setup tools in the `cython` 26
 directory of the DIGRAPH3 resources, namely: 27

¹ CYTHON C-extension for Python, <https://cython.org/>.

cRandPerfTabs.pyx,	28
cIntegerOutrankingDigraphs.pyx,	29
cIntegerSortingDigraphs.pyx, and	30
cSparseIntegerOutrankingDigraphs.pyx.	31

Their automatic compilation and installation, alongside the standard DIGRAPH3 python3 modules, requires the *cython* compiler:

```
...$ python3 -m pip install cython
and a C compiler:
```

```
...$ sudo apt install gcc on Ubuntu, for instance.
```

These cythonized modules, specifically designed for being run on HPC clusters, require the Unix *forking* start method of subprocesses and therefore, due to forking problems on Mac OS platforms, may only operate safely on Linux platforms (see start methods of the [Python multiprocessing library](#)).

11.2 Big Data Performance Tableaux

In order to efficiently type the C variables, the cRandPerfTabs module provides the root cPerformanceTableau class, but, with *integer* action keys, *float* performance evaluations, *integer* criteria weights and *float* discrimination thresholds. And, to limit as much as possible memory occupation of class instances, all the usual verbose comments are dropped from the description of the actions and criteria dictionaries. A random instance may be generated with the cRandomPerformanceTableau class from the cRandPerfTabs module:

Listing 11.1 Big data performance tableau format

```
1 >>> from cRandPerfTabs import cRandomPerformanceTableau
2 >>> t = cRandomPerformanceTableau(numberOfActions=4,\ 49
3 ...                               numberOfCriteria=2) 50
4 >>> t 51
5 ----- PerformanceTableau instance description ----- 52
6     Instance class      : cRandomPerformanceTableau 53
7     Seed                : None 54
8     Instance name       : cRandomperftab 55
9     Actions              : 4 56
10    Criteria             : 2 57
11    Attributes           : ['randomSeed', 'name', 'actions', 58
12                  'criteria', 'evaluation', 'weightPreorder'] 59
13 >>> t.actions 60
14     OrderedDict([(1, {'name': '#1'}), (2, {'name': '#2'}), 61
15                  (3, {'name': '#3'}), (4, {'name': '#4'})]) 62
16 >>> t.criteria 63
17     OrderedDict([ 64
18         ('g1', {'name': 'RandomPerformanceTableau() instance', 65
19                  'thresholds': {'ind': (10.0, 0.0), 66
20                                'pref': (20.0, 0.0), 67
21                                'veto': (80.0, 0.0)}, 68
22                  'scale': (0.0, 100.0), 69
23                  'weight': 1, 70
24                  'preferenceDirection': 'max'}), 71
25         ('g2', {'name': 'RandomPerformanceTableau() instance', 72
26                  'thresholds': {'ind': (10.0, 0.0), 73
27                                'pref': (20.0, 0.0), 74
28                                'veto': (80.0, 0.0)}}, 75
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
```

```

27                     'pref': (20.0, 0.0),
28                     'veto': (80.0, 0.0)},
29             'scale': (0.0, 100.0),
30             'weight': 1,
31             'preferenceDirection': 'max'})])
32 >>> t.evaluation
33 {'g1': {1: 35.17, 2: 56.4, 3: 1.94, 4: 5.51},
34   'g2': {1: 95.12, 2: 90.54, 3: 51.84, 4: 15.42}}
35 >>> t.showPerformanceTableau()
36    Criteria | 'g1'   'g2'
37    Actions  | 1      1
38    -----|-----
39    '#1'    | 91.18  90.42
40    '#2'    | 66.82  41.31
41    '#3'    | 35.76  28.86
42    '#4'    | 7.78   37.64

```

Several methods for converting from the Big Data model to the standard model and vice versa are provided by the `cPerformanceTableau` class.

```

1 >>> t1 = t.convert2Standard()          93
2 >>> t1.convertWeight2Decimal()        94
3 >>> t1.convertEvaluation2Decimal()    95
4 >>> t1
5 *----- PerformanceTableau instance description -----* 97
6   Instance class   : PerformanceTableau          98
7   Seed             : None                      99
8   Instance name   : std_cRandomperftab        100
9   Actions          : 4                         101
10  Criteria         : 2                         102
11  Attributes       : ['name', 'actions', 'criteria', 103
12    'weightPreorder', 'evaluation',           104
13    'randomSeed']                         105

```

11.3 C-implemented Integer-Valued Outranking Digraphs

106

The C-compiled version of the `BipolarOutrankingDigraph` class takes integer r -characteristic values.

107

108

Listing 11.2 Constructing big bipolar-valued outranking digraphs

```

1 >>> from cRandPerfTabs import cRandomPerformanceTableau 109
2 >>> t = cRandomPerformanceTableau(numberOfActions=1000, \ 110
3 ...                           numberOfCriteria=2, seed=100) 111
4 >>> from cIntegerOutrankingDigraphs import\ 112
5 ...                           IntegerBipolarOutrankingDigraph 113
6 >>> g = IntegerBipolarOutrankingDigraph(t,\ 114
7 ...                           Threading=True, nbrCores=4) 115
8 >>> g 116
9 *----- Object instance description -----* 117
10  Instance class   : IntegerBipolarOutrankingDigraph 118
11  Instance name   : rel_cRandomperftab 119
12  Actions          : 1000 120

```

```

13 Criteria : 2 121
14 Size : 460795 122
15 Determinateness : 55.975 123
16 Valuation domain : {'min': -2, 'med': 0, 'max': 2, 124
17 'hasIntegerValuation': True} 125
18 Attributes : ['name', 'actions', 'criteria', 126
19 'totalWeight', 'valuationdomain', 127
20 'methodData', 'evaluation', 128
21 'order', 'runTimes', 'nbrThreads', 129
22 'relation', 'gamma', 'notGamma'] 130
23 ---- Constructor run times (in sec.) ---- 131
24 Total time : 2.29166 132
25 Data input : 0.01170 133
26 Compute relation : 1.73179 134
27 Gamma sets : 0.54816 135
28 Threads : 4 136

```

On a common PC with four single threaded cores, the IntegerBipolarOutrankingDigraph class constructor takes about 2.3 seconds for computing a million pairwise outranking characteristic values (see Listing 11.2 on the previous page). In a similar setting, the standard BipolarOutrankingDigraph constructor operates more than three times slower.

```

1 >>> from outrankingDigraphs import BipolarOutrankingDigraph 142
2 >>> t1 = t.convert2Standard() 143
3 >>> g1 = BipolarOutrankingDigraph(t1,\ 144
4 ... Threading=True,nbrCores=4) 145
5 >>> g1 146
6 *----- Object instance description -----* 147
7 Instance class : BipolarOutrankingDigraph 148
8 Instance name : rel_std_cRandomperfTab 149
9 Actions : 1000 150
10 Criteria : 2 151
11 Size : 460795 152
12 Determinateness (%) : 77.99 153
13 Valuation domain : [-1.00;1.00] 154
14 Attributes : ['name','actions','valuationdomain', 155
15 'criteria','methodData','evaluation', 156
16 'NA','order','runTimes','nbrThreads', 157
17 'relation','gamma','notGamma'] 158
18 ---- Constructor run times (in sec.) ---- 159
19 Total time : 8.35748 160
20 Data input : 0.01407 161
21 Compute relation : 7.30510 162
22 Gamma sets : 1.03831 163
23 Threads : 4 164

```

By far, most of the run time is in each case needed for computing the individual pairwise outranking characteristic values: 1.7 second versus 7.3 second. Notice also below the memory occupations—about 108 MB for g and 212 MB for g1—of both outranking digraph instances (see Line 3 and 5 below).

```

1 >>> from digraphsTools import total_size 169
2 >>> total_size(g) 170

```

3	107503580	171
4	>>> total_size(g1)	172
5	211519995	173
6	>>> total_size(g.relation)/total_size(g)	174
7	0.34	175
8	>>> total_size(g.gamma)/total_size(g)	176
9	0.46	177

The standard Decimal valued BipolarOutrankingDigraph instance `g1` thus nearly doubles the memory occupation of the corresponding IntegerBipolarOutrankingDigraph `g` instance. About 80% of this memory occupation is due to the `g.relation` (34%) and the `g.gamma` (46%) dictionaries. And this ratio grows quadratically with the outranking digraph order. To limit the object sizes for really big outranking digraphs, we need to furthermore abandon the implementation of complete adjacency tables and gamma functions.

11.4 The Sparse Implementation of Big Outranking Digraphs

The idea is to first decompose the complete outranking relation into an ordered collection of equivalent quantile performance classes. Let us consider for this illustration a random performance tableau with 100 decision alternatives evaluated on 7 criteria.

```
1 >>> from cRandPerfTabs import cRandomPerformanceTableau
2 >>> t = cRandomPerformanceTableau(numberOfActions=100, \
3 ...                                     numberOfCriteria=7, seed=100)
```

The `cSparseIntegerOutrankingDigraphs` module provides the `SparseIntegerOutrankingDigraph` class for sorting the 100 decision alternatives into overlapping quartile classes and rank all 100 with respect to the average quantile limits.

Listing 11.3 Constructing the sparse integer outranking digraph

```
1 >>> from cSparseIntegerOutrankingDigraphs import \
...                  SparseIntegerOutrankingDigraph
2 >>> sg = SparseIntegerOutrankingDigraph(t, quantiles=4)
3 >>> sg
4
5 *----- Object instance description -----*
6 Instance class      : SparseIntegerOutrankingDigraph
7 Instance name       : cRandomperftab_mp
8 Actions             : 100
9 Criteria            : 7
10 Sorting by         : 4-Tiling
11 Ordering strategy : average
12 Ranking rule       : Copeland
13 Components          : 6
14 Minimal order      : 1
```

```

15 Maximal order      : 35
16 Average order     : 16.7
17 fill rate          : 24.970%
18 *---- Constructor run times (in sec.) ----
19 Nbr of threads     : 1
20 Total time         : 0.08212
21 QuantilesSorting   : 0.01481
22 Preordering         : 0.00022
23 Decomposing        : 0.06707
24 Ordering            : 0.00000
25 Attributes          : ['runTimes', 'name', 'actions',
26           'criteria', 'evaluation', 'order', 'dimension',
27           'sortingParameters', 'nbrOfCPUs',
28           'valuationdomain', 'profiles', 'categories',
29           'sorting', 'minimalComponentSize',
30           'decomposition', 'nbrComponents', 'nd',
31           'components', 'fillRate',
32           'maximalComponentSize', 'componentRankingRule',
33           'boostedRanking']
```

We obtain in this example here a decomposition into 6 linearly ordered components with a maximal component size of 35 for component c3.

Listing 11.4 The 6 components of a sparse outranking digraph

```

1 >>> sg.showDecomposition()
2 *--- quantiles decomposition in decreasing order---*
3 c1. ]0.75-1.00] : [3,22,24,34,41,44,50,53,56,62,93]
4 c2. ]0.50-1.00] : [7,29,43,58,63,81,96]
5 c3. ]0.50-0.75] : [1,2,5,8,10,11,20,21,25,28,30,33,
6           35,36,45,48,57,59,61,65,66,68,70,
7           71,73,76,82,85,89,90,91,92,94,95,97]
8 c4. ]0.25-0.75] : [17,19,26,27,40,46,55,64,69,87,98,100]
9 c5. ]0.25-0.50] : [4,6,9,12,13,14,15,16,18,23,31,32,
10          37,38,39,42,47,49,51,52,54,60,67,72,
11          74,75,77,78,80,86,88,99]
12 c6. ]<-0.25]    : [79,83,84]
```

A restricted outranking relation is stored for each component with more than one alternative. The `showRelationMap()` prints out below the relation map of the sparse digraph `sg` for the 75 first-ranked alternatives (see Fig. 11.1 on the facing page):

```
1 >>> sg.showRelationMap(toIndex=75)
```

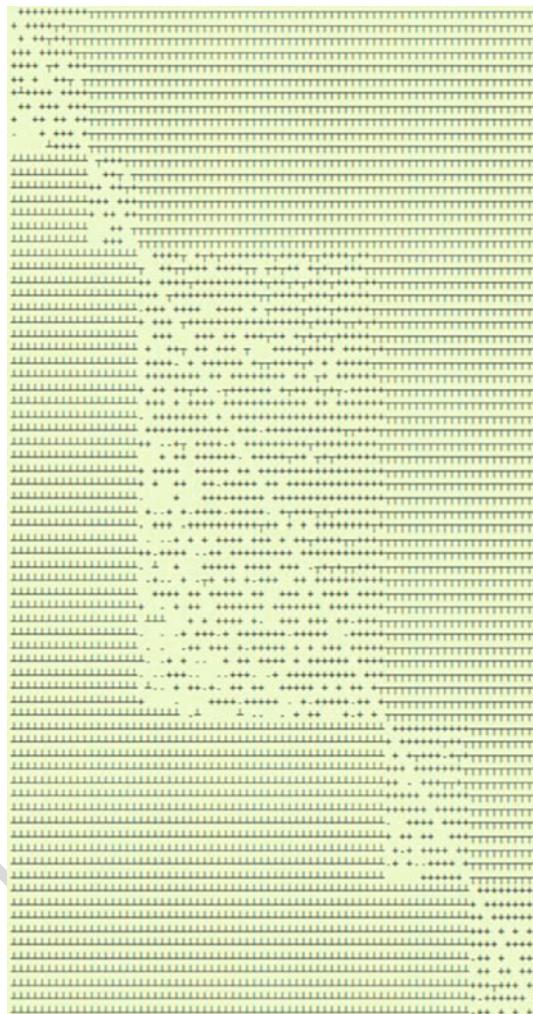
With a fill rate of 25%, the memory occupation of this sparse outranking digraph `sg` instance takes now only 769 kB, compared to the 1.7 MB required by a corresponding standard `BipolarOutrankingDigraph` instance (see Listing 11.3 on the previous page).

```

1 >>> print('%.0fkB' % (total_size(sg)/1024) )
2 769kB
```

For sparse outranking digraphs, the adjacency table is implemented as a dynamic `relation()` function instead of a double dictionary.

Fig. 11.1 Sparse quartiles-sorting decomposed outranking relation (extract).
 Legend: outranking for certain (\top); outranked for certain (\perp); more or less outranking (+); more or less outranked (-); indeterminate (\circ)



Listing 11.5 The relation() function of a sparse outranking digraph

```

1  def relation(self, int x, int y):          258
2      """                                     259
3          *Parameters*:                     260
4          * x (int action key),           261
5          * y (int action key).          262
6          Dynamic construction of the global outranking      263
7          characteristic function  $\star(x \ S \ y)\star$ .  264
8          """                                     265
9          cdef int Min, Med, Max, rx, ry      266
10         Min = self.valuationdomain['min']    267
11         Med = self.valuationdomain['med']    268
12         Max = self.valuationdomain['max']    269
13         if x == y:                         270
14             return Med                   271
15         cx = self.actions[x] ['component'] 272

```

```

16     cy = self.actions[y]['component']          273
17     #print(self.components)                   274
18     rx = self.components[cx]['rank']          275
19     ry = self.components[cy]['rank']          276
20     if rx == ry:                            277
21         try:                                278
22             rxpg = self.components[cx]['subGraph'].relation 279
23             return rxpg[x][y]                  280
24         except AttributeError:               281
25             componentRanking = self.components[cx]['componentRanking'] 282
26             if componentRanking.index(x) < componentRanking.index(y): 283
27                 return Max                  284
28             else:                           285
29                 return Min                  286
30         elif rx > ry:                      287
31             return Min                  288
32         else:                           289
33             return Max                  290

```

11.5 Quantiles Ranking of Big Performance Tableaux

291

The `boostedRanking` attribute of the sparse outranking digraph `sg` contains the 292 global linear ranking of the 100 decision alternatives. This ranking is computed 293 by locally ranking the individual 6 components with the COPELAND rule (see 294 Listing 11.4 on page 142). 295

```

1 >>> sg.boostedRanking          296
2 [22, 53, 3, 34, 56, 62, 24, 44, 50, 93, 41, 63, 29, 58, 297
3 96, 7, 43, 81, 91, 35, 25, 76, 66, 65, 8, 10, 1, 11, 61, 298
4 30, 48, 45, 68, 5, 89, 57, 59, 85, 82, 73, 33, 94, 70, 299
5 97, 20, 92, 71, 90, 95, 21, 28, 2, 36, 87, 40, 98, 46, 55, 300
6 100, 64, 17, 26, 27, 19, 69, 6, 38, 4, 37, 60, 31, 77, 78, 301
7 47, 99, 18, 12, 80, 54, 88, 39, 9, 72, 86, 42, 13, 23, 67, 302
8 52, 15, 32, 49, 51, 74, 16, 14, 75, 79, 83, 84] 303

```

When actually computing linear rankings of the complete set of alternatives, the 304 local outranking relations per component are of no practical usage, and we may 305 furthermore reduce the memory occupation of the resulting digraph by: 306

1. refining the ordering of the quantile classes by taking into account how well an 307 alternative is outranking the lower limit of its quantile class, respectively, the 308 upper limit of its quantile class is *not* outranking the alternative; 309
2. dropping the local outranking digraphs and keeping for each component only a 310 locally ranked list of alternatives. 311

We provide therefore the `cQuantilesRankingDigraph` class from the 312 `cSparseIntegerOutrankingDigraphs` module. 313

Listing 11.6 Ranking the sparse integer outranking digraph

```

1 >>> from cSparseIntegerOutrankingDigraphs import\ 314
2 ...          cQuantilesRankingDigraph          315
3 >>> qr = cQuantilesRankingDigraph(t, 4)          316

```

```

4 >>> qr
5 *----- Object instance description -----*
6 Instance class      : cQuantilesRankingDigraph
7 Instance name       : cRandomperftab_mp
8 Actions             : 100
9 Criteria            : 7
10 Sorting by         : 4-Tiling
11 Ordering strategy  : optimal
12 Ranking rule        : Copeland
13 Components          : 47
14 Minimal order      : 1
15 Maximal order      : 10
16 Average order      : 2.1
17 fill rate          : 2.566%
18 *---- Constructor run times (in sec.) ----*
19 Nbr of threads      : 1
20 Total time          : 0.03702
21 QuantilesSorting    : 0.01785
22 Preordering          : 0.00022
23 Decomposing         : 0.01892
24 Attributes           : ['runTimes', 'name',
25 'actions', 'order', 'dimension', 'sortingParameters',
26 'nbrOfCPUs', 'valuationdomain', 'profiles', 'categories',
27 'sorting', 'minimalComponentSize', 'decomposition',
28 'nbrComponents', 'nd', 'components', 'fillRate',
29 'maximalComponentSize', 'componentRankingRule',
   'boostedRanking']

```

With this *optimised* quantile ordering strategy, we obtain now 47 performance 344 equivalence classes (see Listing 11.6 on the preceding page Line 13). 345

Listing 11.7 The ordered components of the sparse outranking digraph

```

1 >>> qr.components
2 OrderedDict([
3     ('c01', {'rank': 1,
4         'lowQtileLimit': ']0.75',
5         'highQtileLimit': '1.00'],
6         'componentRanking': [53]}),
7     ('c02', {'rank': 2,
8         'lowQtileLimit': ']0.75',
9         'highQtileLimit': '1.00'],
10        'componentRanking': [3, 23, 63, 50]}),
11     ('c03', {'rank': 3,
12         'lowQtileLimit': ']0.75',
13         'highQtileLimit': '1.00'],
14         'componentRanking': [34, 44, 56, 24, 93, 41]}),
15     ...
16     ...
17     ...
18     ('c45', {'rank': 45,
19         'lowQtileLimit': ']0.25',
20         'highQtileLimit': '0.50'],
21         'componentRanking': [49]}),

```

```

22 ('c46', {'rank': 46, 367
23     'lowQtileLimit': ']0.25', 368
24     'highQtileLimit': '0.50]', 369
25     'componentRanking': [52, 16, 86]}), 370
26 ('c47', {'rank': 47, 371
27     'lowQtileLimit': ']<', 372
28     'highQtileLimit': '0.25]', 373
29     'componentRanking': [79, 83, 84]}]), 374
30 >>> print('%.0fkB' % (total_size(qr)/1024) ) 375
31     208kB 376

```

We observe in the last Line of Listing 11.7 on the preceding page an even more considerably less voluminous memory occupation 208 kB, compared to the 769 kB of the SparseIntegerOutrankingDigraph instance.

It is opportune, however, to measure the loss of quality of the resulting COPELAND ranking when working with sparse outranking digraphs.

Listing 11.8 Measuring the loss of quality with respect to the standard outranking digraph

```

1 >>> from cIntegerOutrankingDigraphs import\ 382
2 ...     IntegerBipolarOutrankingDigraph 383
3 >>> ig = IntegerBipolarOutrankingDigraph(t) 384
4 >>> print('Complete outranking : %+.3f'\ 385
5 ...     % (ig.computeOrderCorrelation(\ 386
6 ...         ig.computeCopelandOrder() ['correlation'])) 387
7     Complete outranking : +0.747 388
8
9 >>> print('Sparse 4-tiling : %+.3f'\ 389
10 ...     % (ig.computeOrderCorrelation(\ 390
11 ...         list(reversed(sg.boostedRanking)) ['correlation'])) 391
12     Sparse 4-tiling : +0.717 392
13
14 >>> print('Optimzed sparse 4-tiling: %+.3f'\ 393
15 ...     % (ig.computeOrderCorrelation(\ 394
16 ...         list(reversed(qr.boostedRanking)) )\ 395
17 ...         ['correlation'])) 396
18     Optimzed sparse 4-tiling: +0.705 397

```

The best ranking correlation with the pairwise outranking situations (+0.747) is naturally given when we apply the COPELAND rule to the standard outranking digraph (see Listing 10.8 on page 132 Line 7). When applying the same rule to the sparse 4-tiled outranking digraph, one gets a correlation of +0.717 (Line 12), and when applying the COPELAND rule to the optimised sparse 4-tiled digraph, we still obtain a correlation of +0.705 (Line 18). These optimistic results actually depend on the number of quantiles we use as well as on the given model of random performance tableau.

In case of Random3ObjectivesPerformanceTableau instances we would get in a similar setting an even better standard outranking correlation of +0.86, a sparse 4-tiling correlation of +0.82, and an optimised sparse 4-tiling correlation of +0.81.

Fig. 11.2 *HPC-UL Ranking Performance Records (Spring 2018)*. On the big memory equipped Gaia-183 node with 64 cores we were able to rank one million, respectively, 6 million decision alternatives in about 2 minutes, respectively, 41 minutes

\approx^q outranking order	relation size	q	fill rate	nbr. cores	run time
5 000	25×10^6	4	0.005%	28	0.5"
10 000	1×10^8	4	0.001%	28	1"
100 000	1×10^{10}	5	0.002%	28	10"
1 000 000	1×10^{12}	6	0.001%	64	2'
3 000 000	9×10^{12}	15	0.004%	64	13'
6 000 000	36×10^{12}	15	0.002%	64	41'

11.6 HPC Quantiles Ranking Records

412

Following from the separability property of the q -tiles sorting of each action into a q -tiles class, the q -sorting algorithm may be safely split into as much threads as are multiple processing cores available for working in parallel (see Proposition 9.1). Furthermore, the ranking procedure being local to each diagonal component, these procedures may as well be safely processed in parallel threads on each component restricted outranking digraph.

Using the HPC platform of the University of Luxembourg—<https://hpc-docs.uni.lu/> (Varrette et al. 2014)—the run times shown in Fig. 11.2 for very big ranking problems could be achieved both on:

- *Iris* -skylake nodes with 28 cores: (see <https://hpc-docs.uni.lu/systems/iris>), and
- the 3TB -bigmem *Gaia-183* node with 64 cores (see <https://hpc.uni.lu/systems/gaia>),

by running the cythonized Python modules in an Intel compiled virtual Python 3.6.5 environment [GCC Intel(R) 17.0.1—enable-optimisations c++ gcc 6.3 mode] on Debian 8 Linux.

The next Part III presents three real case studies of building a best choice recommendation—Chap. 12, ranking incommensurable multiple-criteria performance records—Chap. 13, and rating student enrolment qualities—Chap. 14. Chapter 15 eventually list some exercises for home work and exam questions.

References

433

- Bisdorff R (2016) Computing linear rankings from trillions of pairwise outranking situations. In: Busa-Fekete R, Hüllermeier E, Mousseau V, Pfannschmidt K (eds) From multiple criteria decision aid to preference learning , DAP'2016, University of Paderborn (Germany), pp 1–16. <http://hdl.handle.net/10993/28613>
- Varrette S, Bouvry P, Cartiaux H, Georgatos F (2014) Management of an academic HPC cluster: the UL experience. In: Intl. conf. on high performance computing & simulation (HPCS 2014), Bologna (Italy). IEEE, New York, pp 959–967

Part III ¹ Evaluation and Decision Case Studies ²

The third part with four chapters proposes decision making case studies illustrating ³ different performance evaluation models and decision problems. Chapter 12 con- ⁴ cerns building a best choice recommendation for helping a young student to select a ⁵ foreign language study program. Chapter 13, inspired by the THE World University ⁶ Rankings, presents and discusses the mutiple incommensurable criteria ranking of ⁷ Academic Computer Science Departments. Inspired by the 2004 data published by ⁸ DER SPIEGEL magazine, Chap. 14 showcases a multiple-criteria rating problem ⁹ about the student enrolment quality of German universities. Chapter 15 lists finally ¹⁰ some exercises for students taking a course on Algorithmic Decision Theory. ¹¹

Chapter 12	1
Alice's Best Choice: A Selection Case	2
Study	3

Contents	4
12.1 The Decision Problem	152 5
12.2 The Performance Tableau	153 6
12.3 Building a Best Choice Recommendation	156 7
12.4 Robustness Analysis	161 8

Abstract The chapter presents a case study concerning the building of a best choice recommendation for Alice, a German student who wants some advice concerning the choice of her future University studies. We present Alice's performance tableau—potential foreign language study programs, her decision objectives, performance criteria and performance evaluations—and build a best choice recommendation for her. A thorough robustness analysis confirms a very best choice.

The chapter content is inspired by a multiple criteria decision analysis case study (Düsch 2001) (Fig. 12.1).

AQ1
Fig. 12.1 Alice D., 19 years old German student finishing her secondary studies in Köln (Germany), desires to undertake foreign languages studies



12.1 The Decision Problem

17

Alice will probably receive her “Abitur” with satisfactory and/or good marks and 18
 wants to start her further studies thereafter. She would not mind staying in Köln, yet 19
 is ready to move elsewhere if opportune. The length of the higher studies do concern 20
 her, as she wants to earn her life as soon as possible. Her parents, however, agree to 21
 financially support her study fees as well as her living costs during her studies. 22

Alice has already identified 10 potential study programs. 23

In Table 12.1 we notice that Alice considers three *Graduate Interpreter* studies 24
 (8 or 9 Semesters), respectively, in Köln, in Saarbrücken or in Heidelberg; and five 25
Qualified translator studies (8 or 9 Semesters), respectively, in Köln, in Düsseldorf, 26
 in Saarbrücken, in Heidelberg or in Munich. She also considers two short (4 27
 Semesters) study programs at the Chamber of Commerce in Köln. 28

Four **decision objectives** of more or less equal importance are guiding Alice's 29
 choice: 30

1. *maximise* the attractiveness of the study place (GEO), 31
2. *maximise* the attractiveness of her further studies (LEA), 32
3. *minimise* her financial dependency on her parents (FIN), 33
4. *maximise* her professional perspectives (PRA). 34

The decision consequences Alice wishes to take into account for evaluating the 35
 potential study programs with respect to each of the four objectives are modelled 36
 by the following *coherent family of criteria* (Roy 1991; Roy and Bouyssou 1993). 37
 Such a family of performance criteria verifies: 38

1. *Exhaustiveness*: No argument acceptable to Alice can be put forward to justify 39
 a preference in favour of study program x versus program y when x and y have 40
 the same performance level on each of the performance criteria; 41
2. *Cohesiveness*: Alice recognises that program x must be preferred to program y 42
 whenever the performance level of x is significantly better than that of y on one 43

Table 12.1 The potential study programs

ID	Diploma	Institution	City	
T-UD	Qualified translator (T)	University (UD)	Düsseldorf	t3.1 t3.2
T-FHK	Qualified translator (T)	Higher Technical School (FHK)	Köln	t3.3
T-FHM	Qualified translator (T)	Higher Technical School (FHM)	München	t3.4
I-FHK	Graduate interpreter (I)	Higher Technical School (FHK)	Köln	t3.5
T-USB	Qualified translator (T)	University (USB)	Saarbrücken	t3.6
I-USB	Graduate interpreter (I)	University (USB)	Saarbrücken	t3.7
T-UHB	Qualified translator (T)	University (UHB)	Heidelberg	t3.8
I-UHB	Graduate interpreter (I)	University (UHB)	Heidelberg	t3.9
S-HKK	Specialised secretary (S)	Chamber of Commerce (HKK)	Köln	t3.10
C-HKK	Foreign correspondent (C)	Chamber of Commerce (HKK)	Köln	t3.11

Table 12.2 Alice's family of performance criteria

ID	Name	Comment	Objective	Weight	
DH	Proximity	Distance in km to her home (min)	GEO	3	t6.1
BC	Big City	Number of inhabitants (max)	GEO	3	t6.2
AS	Studies	Attractiveness of the studies (max)	LEA	6	t6.3
SF	Fees	Annual study fees (min)	FIN	2	t6.4
LC	Living	Monthly living costs (min)	FIN	2	t6.5
SL	Length	Length of the studies (min)	FIN	2	t6.6
AP	Profession	Attractiveness of the profession (max)	PRA	2	t6.7
AI	Income	Annual income after studying (max)	PRA	2	t6.8
PR	Prestige	Occupational prestige (max)	PRA	2	t6.9
					t6.10

of the criteria of positive weight, performance levels of x and y being the same
on each of the other criteria; 44
45

3. *Non-redundancy*: One of the above requirements is violated if one of the
46 performance criteria is left out from the family. 47

Within each decision objective, the performance criteria are considered to be
48 *equi-significant*. Hence, the four decision objectives get a same *importance weight*
49 of 6 (see Table 12.2 Column 5). 50

12.2 The Performance Tableau

51

The actual evaluations of Alice's potential study programs are stored in a file named
52 AliceChoice.py of PerformanceTableau format.¹ 53

Listing 12.1 Alice's performance tableau

```

1 >>> from perfTabs import PerformanceTableau
2 >>> t = PerformanceTableau('AliceChoice')
3 >>> t.showObjectives()
4     *----- decision objectives -----*
5     GEO: Geographical aspect
6     DH Distance to parent's home 3
7     BC Number of inhabitants 3
8     Total weight: 6 (2 criteria)
9     LEA: Learning aspect
10    AS Attractiveness of the study program 6
11    Total weight: 6.00 (1 criteria)
12    FIN: Financial aspect
13    SF Annual registration fees 2

```

¹ The performance tableau AliceChoice.py may be found in the examples directory of the DIGRAPH3 resources (Bisdorff 2021).

AliceChoice: Family of Criteria

#	Identifier	Name	Comment	Weight	Scale			Thresholds (ax + b)		
					direction	min	max	indifference	preference	veto
1	AI	Annual professional income after studying	Professional aspect measured in x / 1000 Euros	2.00	max	0.00	50.00	0.00x + 0.00	0.00x + 1.00	
2	AP	Attractiveness of the profession	Professional aspect subjectively measured on a three-level scale: 0 (weak), 1 (fair), 2 (good)	2.00	max	0.00	2.00	0.00x + 0.00	0.00x + 1.00	
3	AS	Attractiveness of the study program	Learning aspect subjectively measured from 0 (weak) to 10 (excellent)	6.00	max	0.00	10.00	0.00x + 0.00	0.00x + 1.00	0.00x + 7.00
4	BC	Number of inhabitants	Geographical aspect: measured in x / 1000	3.00	max	0.00	2000.00	0.01x + 0.00	0.05x + 0.00	
5	DH	Distance to parent's home	Geographical aspect measured in km	3.00	min	0.00	1000.00	0.00x + 0.00	0.00x + 10.00	
6	LC	Monthly living costs	Financial aspect measured in Euros	2.00	min	0.00	1000.00	0.00x + 0.00	0.00x + 100.00	
7	OP	Occupational Prestige	Professional aspect measured in SIOPS points	2.00	max	0.00	100.00	0.00x + 0.00	0.00x + 10.00	
8	SF	Annual registration fees	Financial aspect measured in Euros	2.00	min	400.00	4000.00	0.00x + 0.00	0.00x + 100.00	
9	SL	study time	Financial aspect measured in number of semesters	2.00	min	0.00	10.00	0.00x + 0.00	0.00x + 0.50	

Fig. 12.2 Alice's performance criteria

14	LC Monthly living costs	2	67
15	SL Study time	2	68
16	Total weight: 6.00 (3 criteria)		69
17	PRA: Professional aspect		70
18	AP Attractiveness of the profession	2	71
19	AI Annual professional income after studying	2	72
20	OP Occupational Prestige	2	73
21	Total weight: 6.00 (3 criteria)		74

Details of the performance criteria may be consulted in the browser view below. 75

```
1 >>> t.showHTMLCriteria () 76
```

It is worthwhile noticing in Fig. 12.2 that, on her subjective attractiveness scale 77 of the study programs (criterion AS), Alice considers a performance differences of 78 points to be *considerable* and triggering, the case given, an outranking polarisation 79 (Bisdorff 2013). Notice also the proportional *indifference* (1%) and *preference* (5%) 80 discrimination thresholds shown on criterion BC (number of inhabitants). 81

Alice is subjectively evaluating the *Attractiveness* of the studies (criterion AS) 82 on an ordinal scale from *weak* (0) to *excellent* (10). Similarly, she is subjectively 83 evaluating the *Attractiveness* of the respective professions (criterion AP) on a three 84 level ordinal scale from *weak* (0), *fair* (1) to *good* (2). Considering the *Occupational* 85 *Prestige* (criterion OP), she looked up the SIOPS.² All the other evaluation data she 86 found on the internet. 87

² Standard International Occupational Prestige Scale (Ganzeboom and Treiman 1996).

Heatmap of Performance Tableau 'AliceChoice'

criteria	AS	AP	SF	OP	AI	DH	LC	BC	SL
weights	+6.00	+2.00	+2.00	+2.00	+2.00	+3.00	+2.00	+3.00	+2.00
tau(*)	+0.71	+0.64	+0.36	+0.36	+0.24	+0.03	-0.04	-0.07	-0.24
I-FHK	8	2	-400	62	35	0	0	1015	-8
I-USB	8	2	-400	62	45	-269	-1000	196	-9
T-FHK	5	1	-400	62	35	0	0	1015	-8
I-UHB	8	2	-400	62	45	-275	-1000	140	-9
T-UD	5	1	-400	62	45	-41	-1000	567	-9
T-USB	5	1	-400	62	45	-260	-1000	196	-9
T-FHM	4	1	-400	62	35	-631	-1000	1241	-8
T-UHB	5	1	-400	62	45	-275	-1000	140	-9
C-HKK	2	0	-4000	44	30	0	0	1015	-4
S-HKK	1	0	-4000	44	30	0	0	1015	-4

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%
-----------------	--------	--------	--------	--------	---------

(*) tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation

Ranking rule: **NetFlows**

Ordinal (Kendall) correlation between global ranking and global outranking relation: **+0.692**

Fig. 12.3 Heatmap of Alice's performance tableau

In the *heatmap view* shown in Fig. 12.3, we may now consult Alice's performance evaluations.

```
1 >>> t.showHTMLPerformanceHeatmap (\n2 ... colorLevels=5,Correlations=True,ndigits=0)
```

Her ten potential study programs are ordered with the **NETFLOWS** ranking rule applied to the corresponding bipolar-valued outranking digraph (see Sect. 8.3).
Graduate interpreter studies in Köln (I-FHK) or Saarbrücken (I-USB), followed by *Qualified Translator* studies in Köln (T-FHK) appear to be Alice's most preferred alternatives. The least attractive for her appear to be studies at the Chamber of Commerce of Köln (C-HKK, S-HKK). Notice by the way that evaluations on performance criteria to be *minimised*, like *Distance to Home* (criterion DH) or *Study time* (criterion SL), are registered as *negative* values, so that smaller measures are, in this case, preferred to larger ones.

It is finally interesting to observe in Fig. 12.3 (third row) that the *most significant* performance criteria, appear to be for Alice, on the one side, the *Attractiveness* of the study program (criterion AS, tau = +0.72) followed by the *Attractiveness* of the future profession (criterion AP, tau = +0.62). On the other side, *Study times* (criterion SL, tau = -0.24), *Big city* (criterion BC, tau = -0.07) as well as *Monthly living costs* (criterion LC, tau = -0.04) appear to be *not so significant* (see Chap. 16).

12.3 Building a Best Choice Recommendation

107

Let us now have a look at the resulting pairwise outranking situations.

108

Listing 12.2 Computing Alice's outranking digraph

```

1  >>> from outrankingDigraphs import BipolarOutrankingDigraph 109
2  >>> dg = BipolarOutrankingDigraph(t) 110
3  >>> dg 111
4  *----- Object instance description -----* 112
5  Instance class      : BipolarOutrankingDigraph 113
6  Instance name       : rel_AliceChoice 114
7  Actions             : 10 115
8  Criteria            : 9 116
9  Size                : 67 117
10 Determinateness (%) : 73.91 118
11 Valuation domain    : [-1.00;1.00] 119
12 >>> dg.computeSymmetryDegree(Comments=True) 120
13 Symmetry degree of graph <rel_AliceChoice> : 0.49 121

```

From Alice's performance tableau we obtain in the digraph dg 67 positively validated pairwise outranking situations, supported by a 74% majority of criteria significance (see Lines 9–10 in Listing 12.2).

Due to the poorly discriminating performance evaluations, nearly half of these outranking situations (see Line 12) are *symmetric* and reveal actually *more or less indifference* situations between the potential study programs. This is well illustrated in the *relation map* of the outranking digraph shown in Fig. 12.4 on the facing page.

```

1  >>> dg.showHTMLRelationMap (\ 129
2  ...          tableTitle='Outranking relation map', \ 130
3  ...          rankingRule='Copeland') 131

```

We have mentioned that Alice considers a performance difference of 7 points on the *Attractiveness of studies* criterion AS to be considerable which triggers, the case given, a potential polarisation of the outranking characteristics. In Fig. 12.4 on the next page, these polarisations appear in the last column and last row. The `showPolarisations()` method is useful for inspecting the occurrence of outranking polarisations.

Listing 12.3 Inspecting polarised outranking situations

```

1  >>> dg.showPolarisations () 138
2  *----- Negative polarisations -----* 139
3  number of negative polarisations : 3 140
4  1: r(S-HKK >= I-FHK) = -0.17 141
5  criterion: AS 142
6  Considerable performance difference : -7.00 143
7  Veto discrimination threshold       : -7.00 144
8  Polarisation: r(S-HKK >= I-FHK) = -0.17 ==> -1.00 145
9  2: r(S-HKK >= I-USB) = -0.17 146
10 criterion: AS 147
11 Considerable performance difference : -7.00 148

```

Outranking relation map

Ranking rule: Copeland

r(x S y)	I-FHK	I-USB	I-UHB	T-FHK	T-UD	T-USB	T-UHB	T-FHM	C-HKK	S-HKK
I-FHK		.	.	+	+
I-USB	.		+	.	.	.	+	.	.	+
I-UHB	+	.	.	+
T-FHK
T-UD	-	.	.	.		+	+	.	.	.
T-USB	-		+	.	.	.
T-UHB	-	-
T-FHM	-	-	-
C-HKK	-	-	-	-	-	-	-	-		+
S-HKK	-	-	-	-	-	-	-	-	.	.

Semantics

- green +: certainly valid
- light green .: valid
- white: indeterminate
- red -: invalid
- dark red -: certainly invalid

Fig. 12.4 COPELAND ranked outranking relation map

```

12      Veto discrimination threshold      : -7.00          149
13      Polarisation: r(S-HKK >= I-USB) = -0.17 ==> -1.00 150
14      3: r(S-HKK >= I-UHB) = -0.17          151
15      criterion: AS                      152
16      Considerable performance difference : -7.00          153
17      Veto discrimination threshold      : -7.00          154
18      Polarisation: r(S-HKK >= I-UHB) = -0.17 ==> -1.00 155
19      *---- Positive polarisations ----*          156
20      number of positive polarisations: 3          157
21      1: r(I-FHK >= S-HKK) = 0.83          158
22      criterion: AS                      159
23      Considerable performance difference : 7.00          160
24      Counter-veto threshold          : 7.00          161
25      Polarisation: r(I-FHK >= S-HKK) = 0.83 ==> +1.00 162
26      2: r(I-USB >= S-HKK) = 0.17          163
27      criterion: AS                      164
28      Considerable performance difference : 7.00          165
29      Counter-veto threshold          : 7.00          166
30      Polarisation: r(I-USB >= S-HKK) = 0.17 ==> +1.00 167
31      3: r(I-UHB >= S-HKK) = 0.17          168
32      criterion: AS                      169
33      Considerable performance difference : 7.00          170
34      Counter-veto threshold          : 7.00          171
35      Polarisation: r(I-UHB >= S-HKK) = 0.17 ==> +1.00 172

```

In Listing 12.3 on the facing page, we see that considerable performance differences concerning the *Attractiveness of the studies* (AS criterion) are indeed observed between the *Specialised Secretary* study program offered in Köln and the

Graduate Interpreter study programs offered in Köln, Saarbrücken and Heidelberg. 176
 They polarise, hence, three *more or less invalid* outranking situations to *certainly invalid* 177
invalid (Lines 8, 13, 18) and corresponding three *more or less valid* converse 178
 outranking situations to *certainly valid* ones (Lines 25, 30, 35). 179

One may finally notice in the relation map, shown in Fig. 12.4 on the previous 180
 page, that the four best-ranked study programs, I-FHK, I-USB, I-UHB, and 181
 T-FHK, are in fact CONDORCET winners (see Listing 12.4 Line 2), i.e. they are 182
 all four *indifferent* one of the other **and** positively *outranking* all other alternatives, 183
 a result confirmed in Listing 12.4 below by our RUBIS best choice recommendation 184
 (Chap. 4 and Bisdorff et al. 2008). 185

Listing 12.4 Alice's best choice recommendation

```

1  >>> dg.computeCondorcetWinners()          186
2  ['I-FHK', 'I-UHB', 'I-USB', 'T-FHK']      187
3  >>> dg.showBestChoiceRecommendation()      188
4  Best choice recommendation(s) (BCR)          189
5  (in decreasing order of determinateness)      190
6  Credibility domain: [-1.00,1.00]           191
7  === >> potential first choice(s)          192
8  choice : ['I-FHK', 'I-UHB', 'I-USB', 'T-FHK'] 193
9  independence : 0.17                         194
10 dominance : 0.08                           195
11 absorbency : -0.83                         196
12 covering (%) : 62.50                        197
13 determinateness (%) : 68.75                 198
14 most credible action(s) = {'I-FHK': 0.75, 'T-FHK': 0.17, 199
15                           'I-USB': 0.17, 'I-UHB': 0.17} 200
16 === >> potential last choice(s)           201
17 choice : ['C-HKK', 'S-HKK']                 202
18 independence : 0.50                         203
19 dominance : -0.83                         204
20 absorbency : 0.17                          205
21 covered (%) : 100.00                        206
22 determinateness (%) : 58.33                 207
23 most credible action(s) = {'S-HKK': 0.17, 'C-HKK': 0.17} 208

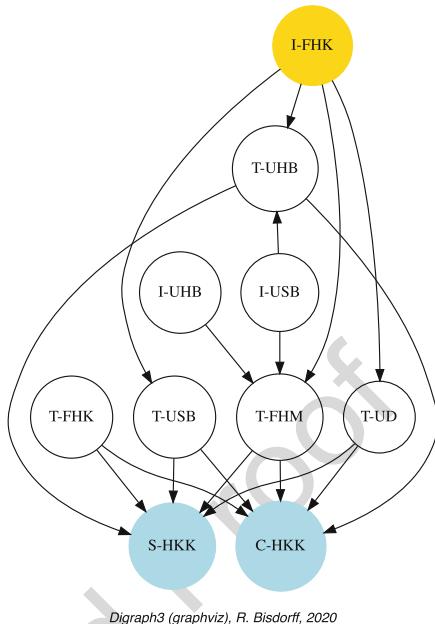
```

Most credible best choice among the four best-ranked study programs eventually 209
 becomes the *Graduate Interpreter* study program at the Technical High School in 210
 Köln (see Line 14) supported by a $(0.75 + 1)/2.0 = 87.5\%(18/24)$ majority of 211
 global criteria significance.³ 212

In the relation map, shown in Fig. 12.4 on the preceding page, we see in 213
 the left lower corner that the *asymmetric part* of the outranking relation, i.e. 214
 the corresponding *strict* outranking relation, is actually *transitive* (see Line 3 in 215
 Listing 12.5 on the next page below). Hence, a *graphviz* drawing of its *skeleton*, 216
 oriented by the previous *first*, respectively, *last* choice, may well illustrate our 217

³ See Sect. 4.5 on computing best choice recommendations and Sect. 17.6 on solving kernel equation systems.

Fig. 12.5 Alice's best choice recommendation. We notice that the *Graduate Interpreter* studies come first, followed by the *Qualified Translator* studies. Last come the *Chamber of Commerce*'s specialised studies. This confirms again the high significance that Alice attaches to the *attractiveness* of her further studies and of her future profession (see criteria AS and AP in Fig. 12.3 on page 155)



DiGraph3 (graphviz), R. Bisdorff, 2020

best choice recommendation. The `Reverse=True` flag in Line 4 eliminates the 218 transitivity induced arcs from digraph `dgcd` (Fig. 12.5). 219

Listing 12.5 Alice's strict best choice recommendation

```

1  >>> dgcd = ~(-dg)
2  >>> dgcd.isTransitive()
3      True
4  >>> dgcd.closeTransitive(Reverse=True, InSite=True)
5  >>> dgcd.exportGraphViz('aliceBestChoice', \
6      ...                     firstChoice=['I-FHK'], \
7      ...                     lastChoice=['S-HKK', 'C-HKK'])
8  *---- exporting a dot file for GraphViz tools ----*
9  Exporting to aliceBestChoice.dot
10 dot -Grankdir=BT -Tpng aliceBestChoice.dot -o
     aliceBestChoice.png

```

Let us now, for instance, check the pairwise outranking situations observed 231 between the first and second-ranked alternative, i.e. *Graduate Interpreter* studies in 232 Köln versus *Graduate Interpreter* studies in Saarbrücken (see *I-FHK* and *I-USB* 233 in Fig. 12.6 on the next page). 234

```

1  >>> dg.showHTMLPairwiseOutrankings('I-FHK', 'I-USB')

```

In a similar way, one may finally compute a *weak ranking* of all the potential 236 studies with the help of the `RankingByChoosingDigraph` class 237 from the `transitiveDigraphs` module that computes a bipolar ranking by 238 conjointly *best-choosing* and *last-rejecting* (Bisdorff 1999). 239

Fig. 12.6 Comparing the first and second best-ranked study programs. The Köln alternative is *at least as well evaluated as* the Saarbrücken alternative on all the performance criteria, except the *Annual income* (of significance 2/24). Conversely, the Saarbrücken alternative is clearly *outperformed* from the geographical (0/6) as well as from the financial perspective (2/6)

Pairwise Comparison

Comparing actions : (I-FHK,I-USB)

crit.	wght.	g(x)	g(y)	diff	ind	pref	concord	v polarisation
AI	2.00	35.00	+45.00	-10	0.00	1.00	-2.00	
AP	2.00	2.00	+2.00	0	0.00	1.00	+2.00	
AS	6.00	8.00	+8.00	0	0.00	1.00	+6.00	
BC	3.00	1015.00	+196.00	819	10.15	50.75	+3.00	
DH	3.00	0.00	-269.00	269	0.00	10.00	+3.00	
LC	2.00	0.00	-1000.00	1000	0.00	100.00	+2.00	
OP	2.00	62.00	+62.00	0	0.00	10.00	+2.00	
SF	2.00	-400.00	-400.00	0	0.00	100.00	+2.00	
SL	2.00	-8.00	-9.00	1	0.00	0.50	+2.00	

Valuation in range: -24.00 to +24.00; global concordance: +20.00

Pairwise Comparison

Comparing actions : (I-USB,I-FHK)

crit.	wght.	g(x)	g(y)	diff	ind	pref	concord	v polarisation
AI	2.00	45.00	+35.00	10	0.00	1.00	+2.00	
AP	2.00	2.00	+2.00	0	0.00	1.00	+2.00	
AS	6.00	8.00	+8.00	0	0.00	1.00	+6.00	
BC	3.00	196.00	+1015.00	-819	10.15	50.75	-3.00	
DH	3.00	-269.00	+0.00	-269	0.00	10.00	-3.00	
LC	2.00	-1000.00	+0.00	-1000	0.00	100.00	-2.00	
OP	2.00	62.00	+62.00	0	0.00	10.00	+2.00	
SF	2.00	-400.00	-400.00	0	0.00	100.00	+2.00	
SL	2.00	-9.00	-8.00	-1	0.00	0.50	-2.00	

Valuation in range: -24.00 to +24.00; global concordance: +4.00

Listing 12.6 Weakly ranking by bipolar best-choosing and last-rejecting

```

1  >>> from transitiveDigraphs import\           240
2      ...                               RankingByChoosingDigraph 241
3  >>> rbc = RankingByChoosingDigraph(dg) 242
4  >>> rbc.showRankingByChoosing() 243
5      Ranking by Choosing and Rejecting 244
6      1st ranked ['I-FHK'] 245
7      2nd ranked ['I-USB'] 246
8      3rd ranked ['I-UHB'] 247
9      4th ranked ['T-FHK'] 248
10     5th ranked ['T-UD'] 249
11     5th last ranked ['T-UD'] 250
12     4th last ranked ['T-UHB', 'T-USB'] 251
13     3rd last ranked ['T-FHM'] 252
14     2nd last ranked ['C-HKK'] 253
15     1st last ranked ['S-HKK'] 254

```

In Listing 12.6, we find confirmed that the *Interpreter* studies appear all preferred to the *Translator* studies. Furthermore, the *Interpreter* studies in Saarbrücken appear preferred to the same studies in Heidelberg. The Köln alternative is apparently the preferred one of all the *Translator* studies. And, the *Foreign Correspondent* and the *Specialised Secretary* studies appear second-last and last ranked.

Yet, how *robust* are our findings with respect to potential settings of the decision objectives' importance and the performance criteria significance weights?

12.4 Robustness Analysis

262

Alice considers her four decision objectives as being *more or less* equally important. 263
 Here we have, however, allocated *strictly equal* importance weights with *strictly* 264
equi-significant criteria per objective. How robust is the previous best choice 265
 recommendation when, now, we would consider the importance of the objectives 266
 and, hence, the significance of the respective performance criteria to be *more or less* 267
uncertain? 268

To answer this question, we consider the respective criteria significance weights 269
 w_j for $j = 1, \dots, 9$ to be *triangular random variables* in the range 0 to $2w_j$ with 270
 mode = w_j . Computing a corresponding 90%-*confident* outranking digraph may 271
 be done with the help of the `ConfidentBipolarOutrankingDigraph` class 272
 (see Chap. 18). 273

Listing 12.7 Computing the 90% confident outranking digraph

```

1 >>> from outrankingGraphs import \
2 ...           ConfidentBipolarOutrankingDigraph
3 >>> cdg = ConfidentBipolarOutrankingDigraph(t, \
4 ...           distribution='triangular', confidence=90.0)
5 >>> cdg
6 ----- Object instance description -----
7 Instance class      : ConfidentBipolarOutrankingDigraph
8 Instance name        : rel_AliceChoice_CLT
9 Actions              : 10
10 Criteria             : 9
11 Size                 : 44
12 Valuation domain    : [-1.00;1.00]
13 Uncertainty model   : triangular(a=0,b=2w)
14 Likelihood domain   : [-1.0;+1.0]
15 Confidence level     : 90.0%
16 Confident majority   : 14/24 (58.3%)
17 Determinateness (%) : 68.19

```

Of the original 67 valid outranking situations, 44 outranking situations are 291
 retained as being 90%-confident (see Listing 12.7 Line 11). The corresponding 292
 90%-confident *qualified majority* of criteria significance amounts to 14/24 = 293
 58.3% (Line 16). 294

Concerning now a 90%-confident best choice recommendation, we are lucky. 295

Listing 12.8 Computing the 90%-confident best choice recommendation

```

1 >>> cdg.computeCondorcetWinners()
2 ['I-FHK']
3 >>> cdg.showBestChoiceRecommendation()
4 ****
5 Best choice recommendation(s) (BCR)
6 (in decreasing order of determinateness)
7 Credibility domain: [-1.00,1.00]
8 === >> potential best choice(s)

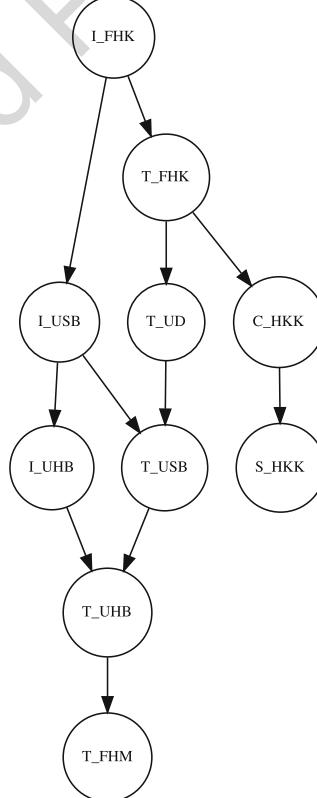
```

9	choice	:	['I-FHK', 'I-UHB', 'I-USB',	304
10			'T-FHK', 'T-FHM']	305
11	independence	:	0.00	306
12	dominance	:	0.42	307
13	absorbency	:	0.00	308
14	covering (%)	:	20.00	309
15	determinateness (%)	:	61.25	310
16	- most credible action(s) = { 'I-FHK': 0.75, }			311

The *Graduate Interpreter* studies in Köln remain indeed a 90%-confident 312
 CONDORCET winner (see Fig. 12.7 Line 2). Hence, the same study program also 313
 remains our 90%-confident most credible best choice supported by a comfortable 314
 18/24(87.5%) majority of the global criteria significance (see Lines 9–10 and 16 in 315
 Listing 12.8 on the previous page on the previous page). 316

When previously comparing the two best-ranked study programs (see Fig. 12.6 317
 on page 160), we have observed that *I-FHK* actually positively outranks *I-USB* on 318
 all four decision objectives. When admitting equi-significant criteria significances 319

Fig. 12.7 Unopposed partial ranking of the potential study programs. Again, when equally significant performance criteria are assumed per decision objective, we observe in the figure here that *I-FHK* remains the stable best choice, independently of the actual importance weights that Alice may wish to allocate to her four decision objectives



per objective, this outranking situation is hence valid independently of the importance weights Alice may allocate to each of her decision objectives. 320
321

The `UnOpposedBipolarOutrankingDigraph` class constructor from the 322
outrankingDigraphs module computes these *unopposed* outranking situa- 323
tions (see Sect. 19.5). 324

Listing 12.9 Computing the unopposed outranking situations

```

1 >>> from outrankingDigraphs import\ 325
2 ... 326
3 >>> uop = UnOpposedBipolarOutrankingDigraph(t) 327
4 >>> uop 328
5 *----- Object instance description -----* 329
6 Instance class : UnOpposedBipolarOutrankingDigraph 330
7 Instance name : AliceChoice_unopposed_outrankings 331
8 Actions : 10 332
9 Criteria : 9 333
10 Size : 28 334
11 Oppositeness (%) : 58.21 335
12 Determinateness (%) : 62.94 336
13 Valuation domain : [-1.00;1.00] 337
14 >>> uop.isTransitive() 338
15 True 339

```

28 out the 67 standard outranking situations remain valid, which leads to an 340
oppositeness degree of $(1.0 - 28/67) = 58.21\%$ (see Lines 10–11 in Listing 12.9). 341
Remarkable furthermore is that this unopposed outranking digraph `uop` is actually 342
transitive, i.e. modelling a *partial ranking* of the study programs (Lines 14–15). 343

The `exportGraphViz()` method of the `TransitiveDigraph` class draws 344
the corresponding partial ranking. 345

```

1 >>> from transitiveDigraphs import TransitiveDigraph 346
2 >>> TransitiveDigraph.exportGraphViz(uop,\ 347
3 ... 348
4 *---- exporting a dot file for GraphViz tools -----* 349
5 Exporting to choiceUnopposed.dot 350
6 dot -Grankdir=TB -Tpng choiceUnopposed.dot -o 351
choiceUnopposed.png 352

```

In view of her performance tableau shown in Fig. 12.3 on page 155, *Graduate* 353
Interpreter studies at the Technical High School Köln, thus, represent definitely 354
Alice's very best choice. 355

For further reading about the RUBIS *Best Choice* methodology, one may consult 356
in Bisdorff (2015) the study of a *real decision-aiding case* about choosing a best 357
poster in a scientific conference. 358

References

359

Bisdorff R (1999) Bipolar ranking from pairwise fuzzy outrankings. *Belg J Oper Res Stat Comp Sci* 37(4):379–387. <http://hdl.handle.net/10993/38738> 360
361

Bisdorff R (2013) On polarizing outranking relations with large performance differences. <i>J Multi-Crit Dec Anal</i> Wiley 20:3–12. http://hdl.handle.net/10993/245	362
	363
Bisdorff R (2015) The EURO 2004 best poster award: choosing the best poster in a scientific conference. In: Bisdorff R, Dias L, Meyer P, Mousseau V, Pirlot M (eds) <i>Evaluation and decision models with multiple criteria: case studies</i> . Springer, New York, pp 117–166	364
	365
	366
Bisdorff R (2021) Documentation of the Digraph3 collection of Python modules for Algorithmic Decision Theory. https://digraph3.readthedocs.io/en/latest/	367
	368
Bisdorff R, Meyer P, Roubens M (2008) Rubis: a bipolar-valued outranking method for the best choice decision problem. <i>Quart J Oper Res</i> 6(2):143–165. http://hdl.handle.net/10993/23716	369
	370
Düscher E (2001) Entscheidung für eine fremdsprachenausbildung. In: Eisenführ F, Langer T, Weber M (eds) <i>Fallstudien zu rationalem Entscheiden. Fallstudie A</i> . Springer, New York, pp 1–18	371
	372
Ganzeboom H, Treiman D (1996) Internationally comparable measures of occupational status for the 1988 international standard classification of occupations. <i>Soc Sci Res</i> 25:201–239	373
	374
Roy B (1991) The outranking approach and the foundations of electre methods. <i>Theory Decis</i> 31(1):49–73	375
	376
Roy B, Bouyssou D (1993) <i>Aide Multicritère à la Décision : Méthodes et Cas</i> . Economica, Paris	377
	377

AUTHOR QUERY

- AQ1. Missing citation for “Figs. 12.1, 12.5” was inserted here. Please check if appropriate. Otherwise, please provide citation for “Figs. 12.1, 12.5”. Note that the order of main citations of figures in the text must be sequential.

Uncorrected Proof

Chapter 13

The Best Academic Computer Science Depts: A Ranking Case Study

1
2
3

Contents

13.1 The THE Performance Tableau	165	5
13.2 Ranking with Multiple Criteria of Ordinal Significance	171	6
13.3 How to Judge the Quality of a Ranking Result?	178	7

Abstract In this case study, we are solving with our DIGRAPH3 resources a ranking decision problem based on published data from the *Times Higher Education* (THE) *World University Rankings* 2016 by *Computer Science* (CS) subject. Several hundred academic CS Departments, from all over the world, were ranked that year following an overall numerical score based on the weighted average of five performance criteria: *Teaching* (the learning environment, 30%), *Research* (volume, income and reputation, 30%), *Citations* (research influence, 27.5%), *International outlook* (staff, students, and research, 7.5%), and *Industry income* (innovation, 5%). To illustrate our DIGRAPH3 programming resources, we shall first have a look into the THE multiple-criteria ranking data with short Python scripts. In a second section, we shall relax the commensurability hypothesis of the ranking criteria and show how to similarly rank with multiple incommensurable performance criteria of solely ordinal significance. A third section is finally devoted to introduce quality measures for qualifying ranking results.

13.1 The THE Performance Tableau

22

For this decision making case study, we use an extract of the published TIMES HIGHER EDUCATION (THE) World University rankings 2016 by Computer Science (CS) subject, concerning the 75 first-ranked academic Institutions.¹ The multiple-

¹ THE World University Rankings 2016 by Computer Science subject.

criteria performance tableau data, collected from the THE web pages, is stored in a file named `the-cs-2016.py` of PerformanceTableau format.²

26
27

AQ1

Listing 13.1 Performance tableau of the

```

1  >>> from perfTabs import PerformanceTableau
2  >>> cspt = PerformanceTableau('the_cs_2016')
3  >>> cspt
4  *----- PerformanceTableau instance description -----*
5  Instance class      : PerformanceTableau
6  Instance name       : the-cs-2016
7  Actions             : 75
8  Objectives          : 5
9  Criteria            : 5
10 NaN proportion (%) : 0.0
11 Attributes          : ['name','description','actions',
12                           'objectives','criteria',
13                           'weightPreorder','NA','evaluation']

```

Potential decision alternatives, in our case here, are the 75 THE best-ranked CS Departments in 2016, all of them located at world renowned Institutions, like California Institute of Technology, Swiss Federal Institute of Technology Zurich, Technical University München, University of Oxford or the National University of Singapore (see Listing 13.2 below).

Instead of using prefigured DIGRAPH3 `show...()` methods, readily available for inspecting `PerformanceTableau` instances, Listing 13.2 illustrates how to write small Python scripts for printing out its content.

Listing 13.2 Printing the CS Departments

```

1  >>> for x in cspt.actions:
2  ...     print('%s:\t%s (%s)' %
3  ...           (x,cspt.actions[x]['name'],cspt.actions[x]['comment']) )
4  albt: University of Alberta (CA)
5  anu: Australian National University (AU)
6  ariz: Arizona State University (US)
7  bjup: Beijing University (CN)
8  bro: Brown University (US)
9  calt: California Institute of Technology (US)
10 cbu: Columbia University (US)
11 chku: Chinese University of Hong Kong (HK)
12 cihk: City University of Hong Kong (HK)
13 cir: University of California at Irvine (US)
14 cmel: Carnegie Mellon University (US)
15 cou: Cornell University (US)
16 csb: University of California at Santa Barbara (US)
17 csd: University Of California at San Diego (US)
18 dut: Delft University of Technology (NL)
19 eind: Eindhoven University of Technology (NL)
20 ens: Superior Normal School at Paris (FR)
21 epfl: Swiss Federal Institute of Technology Lausanne (CH)
22 epfr: Polytechnic school of Paris (FR)
23 ethz: Swiss Federal Institute of Technology Zurich (CH)

```

² The performance tableau `the-cs-2016.py` is available in the `examples` directory of the DIGRAPH3 resources (Bisdorff 2021).

24	frei:	University of Freiburg (DE)	72
25	git:	Georgia Institute of Technology (US)	73
26	glas:	University of Glasgow (UK)	74
27	hels:	University of Helsinki (FI)	75
28	hkpu:	Hong Kong Polytechnic University (CN)	76
29	hkst:	Hong Kong University of Science and Technology (HK)	77
30	hku:	Hong Kong University (HK)	78
31	humb:	Berlin Humboldt University (DE)	79
32	ic1:	Imperial College London (UK)	80
33	indis:	Indian Institute of Science (IN)	81
34	itmo:	ITMO University (RU)	82
35	kcl:	King's College London (UK)	83
36	kist:	Korea Adv. Institute of Science and Technology (KR)	84
37	kit:	Karlsruhe Institute of Technology (DE)	85
38	kth:	KTH Royal Institute of Technology (SE)	86
39	kuj:	Kyoto University (JP)	87
40	kul:	Catholic University Leuven (BE)	88
41	lms:	Lomonosov Moscow State University (RU)	89
42	man:	University of Manchester (UK)	90
43	mcp:	University of Maryland College Park (US)	91
44	mel:	University of Melbourne (AU)	92
45	mil:	Polytechnic University of Milan (IT)	93
46	mit:	Massachusetts Institute of Technology (US)	94
47	naji:	Nanjing University (CN)	95
48	ntu:	Nanyang Technological University of Singapore (SG)	96
49	ntw:	National Taiwan University (TW)	97
50	nyu:	New York University (US)	98
51	oxf:	University of Oxford (UK)	99
52	pud:	Purdue University (US)	100
53	qut:	Queensland University of Technology (AU)	101
54	rcu:	Rice University (US)	102
55	rwth:	RWTH Aachen University (DE)	103
56	shji:	Shanghai Jiao Tong University (CN)	104
57	sing:	National University of Singapore (SG)	105
58	sou:	University of Southampton (UK)	106
59	stut:	University of Stuttgart (DE)	107
60	tech:	Technion - Israel Institute of Technology (IL)	108
61	tlavu:	Tel Aviv University (IR)	109
62	tsu:	Tsinghua University (CN)	110
63	tub:	Technical University of Berlin (DE)	111
64	tud:	Technical University of Darmstadt (DE)	112
65	tum:	Technical University of Muenchen (DE)	113
66	ucl:	University College London (UK)	114
67	ued:	University of Edinburgh (UK)	115
68	uiu:	University of Illinois at Urbana-Champagne (US)	116
69	unlu:	University of Luxembourg (LU)	117
70	unsw:	University of New South Wales (AU)	118
71	unt:	University of Toronto (CA)	119
72	uta:	University of Texas at Austin (US)	120
73	utj:	University of Tokyo (JP)	121
74	utw:	University of Twente (NL)	122
75	uwa:	University of Waterloo (CA)	123
76	wash:	University of Washington (US)	124
77	wtu:	Vienna University of Technology (AUS)	125
78	zhej:	Zhejiang University (CN)	126

The THE authors base their 2016 ranking on five decision objectives (Times Higher Education 2016).

Listing 13.3 The THE ranking objectives

```

1 >>> for obj in cspt.objectives:
2 ...     print('%.1f%%, %s' % (obj['weight'], obj['name']))
3 ...     print('%.1f%%, %s' % (obj['comment'], obj['comment']))
4 ...     print('%.1f%%, %s' % (obj['name'], obj['comment']))
5 ...

```

```
6     )
7 Teaching: Best learning environment (30.0%) 134
8     Reputation survey; Staff-to-student ratio; 135
9     Doctorate-to-student ratio; 136
10    Doctorate-to-academic-staff ratio; 137
11    Institutional income. 138
12 Research: Highest volume and reputation (30.0%) 139
13     Reputation survey; 140
14     Research income; 141
15     Research productivity. 142
16 Citations: Highest research influence (27.5%) 143
17     Impact. 144
18 International outlook: Most international staff, 145
19     students and research (7.5%) 146
20     Proportions of international students; 147
21     Proportions of international staff; 148
22     International collaborations. 149
23 Industry income: Best knowledge transfer (5.0%) 150
24     Volume. 151
25
```

With a cumulated importance of 87% (see above), *Teaching*, *Research*, and *Citations* represent clearly the major ranking objectives. *International outlook* and *Industry income* are considered of minor importance (12.5%).

THE authors do, unfortunately, not publish the detail of their performance assessments for evaluating CS Depts with respect to each one of performance criteria per ranking objective.³ The THE 2016 ranking publication reveals solely a compound assessment on a single performance criteria per ranking objective. The five retained performance criteria may be printed out as follows.

```
1 >>> for g in cspt.criteria:
2 ...     print('"%s:\t%s, %s (%.1f%%)" %\
3 ...         (g,cspt.criteria[g]['name'],cspt.criteria[g]['comment'],\
4 ...          cspt.criteria[g]['weight']) )
5 gtch: Teaching, The learning environment (30.0%)
6 gres: Research, Volume, income and reputation (30.0%)
7 gcit: Citations, Research influence (27.5%)
8 gint: International outlook, In staff, students and research
9 (7.5%)
9 qind: Industry income, knowledge transfer (5.0%)
```

The largest part (87.5%) of criteria significance is, hence canonically, allocated to the major ranking criteria: *Teaching* (30%), *Research* (30%) and *Citations* (27.5%). The small remaining part (12.5%) goes to *International outlook* (7.5%) and *Industry income* (5%).

In order to render commensurable these performance criteria, the THE authors replace, per criterion, the actual performance evaluation obtained by each University with the corresponding *quantile* observed in the cumulative distribution of the performance evaluations obtained by all the surveyed institutions (Times Higher Education 2016). The THE ranking is eventually determined by an *overall score*

³ THE gives some insight on the subject and significance of the actual ranking criteria used for evaluating along each ranking objective on her website (Times Higher Education 2016).

per University which corresponds to the weighted average of these five criteria 181
 quantiles, as illustrated in Listing 13.4. 182

Listing 13.4 Computing the THE overall scores

```

1 >>> theScores = []
2 >>> for x in cspt.actions:
3 ...     xscore = Decimal('0')
4 ...     for g in cspt.criteria:
5 ...         xscore += cspt.evaluation[g][x] * \
6 ...             (cspt.criteria[g]['weight']/Decimal('100'))
7 ...     theScores.append((xscore,x))

```

In Listing 13.5 (Lines 15–16), we may thus notice that, in the 2016 edition of 190
 the THE World University rankings by CS subject, the Swiss Federal Institute of 191
 Technology Zürich is first-ranked with an overall score of 92.9; followed by the 192
 California Institute of Technology (overall score: 92.4).⁴ 193

Listing 13.5 Printing the ranked performance table

```

1 >>> theScores.sort(reverse = True)
2 >>> print('## Univ \gtch gres gcit gint gind overall')
3 >>> print('-----')
4 >>> i = 1
5 >>> for it in theScores:
6 ...     x = it[1]
7 ...     xScore = it[0]
8 ...     print('%2d: %s' % (i,x), end=' \t')
9 ...     for g in cspt.criteria:
10 ...         print('.1f' % (cspt.evaluation[g][x]),end=' ')
11 ...         print(' .1f' % xScore)
12 ...     i += 1
13     ## Univ gtch gres gcit gint gind overall
14     -----
15     1: ethz 89.2 97.3 97.1 93.6 64.1 92.9
16     2: calt 91.5 96.0 99.8 59.1 85.9 92.4
17     3: oxf 94.0 92.0 98.8 93.6 44.3 92.2
18     4: mit 87.3 95.4 99.4 73.9 87.5 92.1
19     5: git 87.2 99.7 91.3 63.0 79.5 89.9
20     6: cmel 88.1 92.3 99.4 58.9 71.1 89.4
21     7: icl 90.1 87.5 95.1 94.3 49.9 89.0
22     8: epfl 86.3 91.6 94.8 97.2 42.7 88.9
23     9: tum 87.6 95.1 87.9 52.9 95.1 87.7
24    10: sing 89.9 91.3 83.0 95.3 50.6 86.9
25    11: cou 81.6 94.1 99.7 55.7 45.7 86.6
26    12: ucl 85.5 90.3 87.6 94.7 42.4 86.1
27    13: wash 84.4 88.7 99.3 57.4 41.2 85.6
28    14: hkst 74.3 92.0 96.2 84.4 55.8 85.5
29    15: ntu 76.6 87.7 90.4 92.9 86.9 85.5
30    16: ued 85.7 85.3 89.7 95.0 38.8 85.0
31    17: unt 79.9 84.4 99.6 77.6 38.4 84.4
32    18: uiu 85.0 83.1 99.2 51.4 42.2 83.7
33    19: mcp 79.7 89.3 94.6 29.8 51.7 81.5
34    20: cbu 81.2 78.5 94.7 66.9 45.7 81.3
35    21: tsu 88.1 90.2 76.7 27.1 85.9 80.9
36    22: csd 75.2 81.6 99.8 39.7 59.8 80.5

```

⁴ The author's own Computer Science Dept at the University of Luxembourg was ranked on position 63 with an overall score of 58.0.

37	23: uwa	75.3	82.6	91.3	72.9	41.5	80.0	230
38	24: nyu	71.1	77.4	99.4	78.0	39.8	79.7	231
39	25: uta	72.6	85.3	99.6	31.6	49.7	79.6	232
40	26: kit	73.8	85.5	84.4	41.3	76.8	77.9	233
41	27: bju	83.0	85.3	70.1	30.7	99.4	77.0	234
42	28: csb	65.6	70.9	94.8	72.9	74.9	76.2	235
43	29: rwth	77.8	85.0	70.8	43.7	89.4	76.1	236
44	30: hku	77.0	73.0	77.0	96.8	39.5	75.4	237
45	31: pud	76.9	84.8	70.8	58.1	56.7	75.2	238
46	32: kist	79.4	88.2	64.2	31.6	92.8	74.9	239
47	33: kcl	45.5	94.6	86.3	95.1	38.3	74.8	240
48	34: chku	64.1	69.3	94.7	75.6	49.9	74.2	241
49	35: epfr	81.7	60.6	78.1	85.3	62.9	73.7	242
50	36: dut	64.1	78.3	76.3	69.8	90.1	73.4	243
51	37: tub	66.2	82.4	71.0	55.4	99.9	73.3	244
52	38: utj	92.0	91.7	48.7	25.8	49.6	72.9	245
53	39: cir	68.8	64.6	93.0	65.1	40.4	72.5	246
54	40: ntw	81.5	79.8	66.6	25.5	67.6	72.0	247
55	41: anu	47.2	73.0	92.2	90.0	48.1	70.6	248
56	42: rcu	64.1	53.8	99.4	63.7	46.1	69.8	249
57	43: mel	56.1	70.2	83.7	83.3	50.4	69.7	250
58	44: lms	81.5	68.1	61.0	31.1	87.8	68.4	251
59	45: ens	71.8	40.9	98.7	69.6	43.5	68.3	252
60	46: wtu	61.8	73.5	73.7	51.9	62.2	67.9	253
61	47: tech	54.9	71.0	85.1	51.7	40.1	67.1	254
62	48: bro	58.5	54.9	96.8	52.3	38.6	66.5	255
63	49: man	63.5	71.9	62.9	84.1	42.1	66.3	256
64	50: zhej	73.5	70.4	60.7	22.6	75.7	65.3	257
65	51: frei	54.2	51.6	89.5	49.7	99.9	65.1	258
66	52: unsw	60.2	58.2	70.5	87.0	44.3	63.6	259
67	53: kuj	75.4	72.8	49.5	28.3	51.4	62.8	260
68	54: sou	48.2	60.7	75.5	87.4	43.2	62.1	261
69	55: shJi	66.9	68.3	62.4	22.8	38.5	61.4	262
70	56: itmo	58.0	32.0	98.7	39.2	68.7	60.5	263
71	57: kul	35.2	55.8	92.0	46.0	88.3	60.5	264
72	58: glas	35.2	52.5	91.2	85.8	39.2	59.8	265
73	59: utw	38.2	52.8	87.0	69.0	60.0	59.4	266
74	60: stut	54.2	60.6	61.1	36.3	97.8	58.9	267
75	61: naji	51.4	76.9	48.8	39.7	74.4	58.6	268
76	62: tud	46.6	53.6	75.9	53.7	66.5	58.3	269
77	63: unlu	35.2	44.2	87.4	99.7	54.1	58.0	270
78	64: gut	45.5	42.6	82.8	75.2	63.0	58.0	271
79	65: hkpu	46.8	36.5	91.4	73.2	41.5	57.7	272
80	66: albt	39.2	53.3	69.9	91.9	75.4	57.6	273
81	67: mil	46.4	64.3	69.2	44.1	38.5	57.5	274
82	68: hels	48.8	49.6	80.4	50.6	39.5	57.4	275
83	69: cihk	42.4	44.9	80.1	76.2	67.9	57.3	276
84	70: tlavu	34.1	57.2	89.0	45.3	38.6	57.2	277
85	71: indis	56.9	76.1	49.3	20.1	41.5	57.0	278
86	72: ariz	28.4	61.8	84.3	59.3	42.0	56.8	279
87	73: kth	44.8	42.0	83.6	71.6	39.2	56.4	280
88	74: humb	48.4	31.3	94.7	41.5	45.5	55.3	281
89	75: eind	32.4	48.4	81.5	72.2	45.8	54.4	282

It is important to notice that a ranking by weighted average scores requires commensurable ranking criteria of precise decimal significance and on which precise decimal performance evaluations are given. It is very unlikely that the THE 2016 performance assessments verify indeed these conditions. Here we show how to relax these methodological requirements—precise commensurable criteria and decimal evaluations—by following instead a bipolar-valued epistemic logic based ranking methodology (see Chap. 8).

13.2 Ranking with Multiple Criteria of Ordinal Significance

290

Let us, first, have a critical look in Fig. 13.1 at the THE performance criteria.

291

```
| >>> cspt.showHTMLCriteria(Sorted=False)
```

292

Considering a very likely imprecision of the performance evaluation procedure, followed by some potential violation of uniform distributed quantile classes, we assume here that a performance quantile difference of up to 2.5% is *insignificant*, whereas a difference of 5% and more warrants a *clearly better*, respectively, *clearly less good* performance. With quantiles 94%, respectively, 87.3%, Oxford's CS teaching environment, for instance, is thus clearly better evaluated than that of the MIT (see Listing 13.5 on page 169 Lines 27–28). We shall furthermore assume that a *considerable* performance quantile difference of 60%, observed on the three major ranking criteria: *Teaching*, *Research*, and *Citations*, will trigger the polarisation of pairwise outranking, respectively, outranked situations (Bisdorff 2013).

293

The effect these performance discrimination thresholds induce on the outranking modelling can be inspected with the `showCriteria()` method.

303

304

Listing 13.6 Inspecting the performance discrimination thresholds

```
1 >>> cspt.showCriteria()
2     ----- criteria -----
3     gtch 'Teaching'
4         Scale = (Decimal('0.00'), Decimal('100.00'))
5         Weight = 0.300
6         Threshold ind : 2.50 + 0.00x ; percentile: 8.07
7         Threshold pref : 5.00 + 0.00x ; percentile: 15.75
8         Threshold veto : 60.00 + 0.00x ; percentile: 99.75
9
10    gres 'Research'
11        Scale = (Decimal('0.00'), Decimal('100.00'))
12        Weight = 0.300
13        Threshold ind : 2.50 + 0.00x ; percentile: 7.86
14        Threshold pref : 5.00 + 0.00x ; percentile: 16.14
15        Threshold veto : 60.00 + 0.00x ; percentile: 99.21
```

305

306

307

308

309

310

311

312

313

314

315

316

317

318

the_cs_2016: Family of Criteria

#	Identifier	Name	Comment	Weight	Scale			Thresholds (ax + b)		
					direction	min	max	Indifference	Preference	veto
1	gtch	Teaching	The learning environment	30.00	max	0.00	100.00	0.00x + 2.50	0.00x + 5.00	0.00x + 60.00
2	gres	Research	Volume, income and reputation	30.00	max	0.00	100.00	0.00x + 2.50	0.00x + 5.00	0.00x + 60.00
3	gcit	Citations	Research influence	27.50	max	0.00	100.00	0.00x + 2.50	0.00x + 5.00	0.00x + 60.00
4	gint	International outlook	In staff, students and research	7.50	max	0.00	100.00	0.00x + 2.50	0.00x + 5.00	
5	gind	Industry income	Innovation	5.00	max	0.00	100.00	0.00x + 2.50	0.00x + 5.00	

Fig. 13.1 The THE ranking criteria

```

15 gcit 'Citations' 319
16   Scale = (Decimal('0.00'), Decimal('100.00')) 320
17   Weight = 0.275 321
18   Threshold ind : 2.50 + 0.00x ; percentile: 11.82 322
19   Threshold pref : 5.00 + 0.00x ; percentile: 22.99 323
20   Threshold veto : 60.00 + 0.00x ; percentile: 100.00 324
21 gint 'International outlook' 325
22   Scale = (Decimal('0.00'), Decimal('100.00')) 326
23   Weight = 0.075 327
24   Threshold ind : 2.50 + 0.00x ; percentile: 6.45 328
25   Threshold pref : 5.00 + 0.00x ; percentile: 11.75 329
26 gind 'Industry income' 330
27   Scale = (Decimal('0.00'), Decimal('100.00')) 331
28   Weight = 0.050 332
29   Threshold ind : 2.50 + 0.00x ; percentile: 11.82 333
30   Threshold pref : 5.00 + 0.00x ; percentile: 21.51 334

```

Between 6% and 12% of the observed quantile differences are, hence, considered 335
 to be *insignificant*. Similarly, between 77% and 88% are considered to be *significant*. 336
 Less than 1% correspond to *considerable* quantile differences on both the *Teaching* 337
 and *Research* criteria; actually triggering an epistemic polarisation effect (Bisdorff 338
 2013). 339

Besides the likely imprecise performance discrimination, the *precise decimal* 340
significance weights, as allocated by the THE authors to the five ranking criteria 341
 are, as well, quite *questionable*. Criteria significance weights may carry sometimes 342
 hidden strategies for rendering the performance evaluations commensurable in view 343
 of a numerical computation of the overall ranking scores. The eventual ranking 344
 result is, the case given, as much depending on the precise values of the given 345
 criteria significance weights as, vice versa, the given precise significance weights 346
 are depending on the subjectively expected and accepted ranking results.⁵ We will 347
 therefore drop these precise decimal weights and, instead, only require a corre- 348
 sponding criteria significance preorder: $gtch = gres > gcit > gint > gind$. 349
Teaching environment and *Research volume and reputation* are equally considered 350
 most significant, followed by *Research influence*. Then comes *International outlook* 351
in staff, students and research and, least significant finally, *Industry income and* 352
innovation. 353

Both these working hypotheses:—performance discrimination thresholds and— 354
 solely ordinal criteria significance, give us way to a ranking methodology based on 355
robust pairwise outranking situations (see Chap. 19 and (Bisdorff 2004)). 356

**Definition 13.1 (Robust Outranking Situations with Ordinal Criteria Signifi- 357
 cance Weights)** 358

- We say that CS Dept x *robustly outranks* CS Dept y when x positively outranks 359
 y with **all** significance weight vectors that are compatible with the significance 360
 preorder: $gtch = gres > gcit > gint > gind$; 361

⁵ In a social choice context, this potential double bind between voting profiles and election result corresponds to voting manipulation strategies.

- We say that CS Dept x is *robustly outranked* by CS Dept y when x is positively outranked by y with **all** significance weight vectors that are compatible with the significance preorder: $gtch = gres > gcit > gint > gind$; 363
- Otherwise, CS Depts x and y are considered to be *incomparable*. 365

A corresponding digraph constructor is provided by the RobustOutranking- 366
Digraph class. 367

Listing 13.7 Computing the robust outranking digraph

```

1 >>> from outrankingDigraphs import RobustOutrankingDigraph 368
2 >>> rdg = RobustOutrankingDigraph(cspt) 369
3 >>> rdg 370
4     *----- Object instance description -----* 371
5     Instance class      : RobustOutrankingDigraph 372
6     Instance name       : robust_the_cs_2016 373
7     Actions             : 75 374
8     Criteria            : 5 375
9     Size                : 2993 376
10    Determinateness (%) : 78.16 377
11    Valuation domain    : [-1.00;1.00] 378
12 >>> rdg.computeIncomparabilityDegree(Comments=True) 379
13     Incomparability degree (%) of digraph <robust_the_cs_2016>: 380
14     links x<->y y: 2775, incomparable: 102, comparable: 2673 381
15     (incomparable/links) = 0.037 382
16 >>> rdg.computeTransitivityDegree(Comments=True) 383
17     Transitivity degree of digraph <robust_the_cs_2016>: 384
18     triples x>y>z: 405150, closed: 218489, open: 186661 385
19     (closed/triples) = 0.539 386
20 >>> rdg.computeSymmetryDegree(Comments=True) 387
21     Symmetry degree (%) of digraph <robust_the_cs_2016>: 388
22     arcs x>y: 2673, symmetric: 320, asymmetric: 2353 389
23     (symmetric/arcs) = 0.12 390

```

In the resulting digraph instance `rdg` (see Line 9 in Listing 13.7), we observe 391
2993 such robust pairwise outranking situations validated with a mean significance 392
of 78% (Line 10). Unfortunately, in the case here, they do not deliver any complete 393
linear ranking. The robust outranking digraph `rdg` contains 102 incomparability 394
situations (3.7%, Line 15); nearly half of its transitive closure is missing (46.1%, 395
Line 19) and 12% of the positive outranking situations correspond in fact to 396
symmetric indifference situations (Line 23).

Worse even, the digraph `rdg` admits a really high number of outranking circuits.⁶ 398

Listing 13.8 Inspecting outranking circuits

```

1 >>> rdg.computeChordlessCircuits() 399
2 >>> rdg.showChordlessCircuits() 400

```

⁶The `computeChordlessCircuits()` and `showChordlessCircuits()` methods are separate because there are various methods available for enumerating the chordless circuits in a digraph (Bisdorff 2010).

```

3 *---- Chordless circuits ----* 401
4 145 circuits. 402
5 1: ['albt','unlu','ariz','hels'], cred. : 0.300 403
6 2: ['albt','tlavi','hels'], cred. : 0.150 404
7 3: ['anu', 'man', 'itmo'], cred. : 0.250 405
8 4: ['anu', 'zhej', 'rcu'], cred. : 0.250 406
9 ...
10 ...
11 82: ['csb','epfr','rwth'], cred. : 0.250 409
12 83: ['csb','epfr','pud','nyu'], cred. : 0.250 410
13 84: ['csd','kcl','kist'], cred. : 0.250 411
14 ...
15 ...
16 142: ['kul','qut','mil'], cred. : 0.250 414
17 143: ['lms','rcu','tech'], cred. : 0.300 415
18 144: ['mil','stut','qut'], cred. : 0.300 416
19 145: ['mil','stut','tud'], cred. : 0.300 417

```

Among the 145 detected robust outranking circuits reported in Listing 13.8 on the preceding page, we notice, for instance, two outranking circuits of length 4 (see circuits 1 and 83).

Let us inspect below the bipolar-valued robust outranking characteristics of the first circuit.

Listing 13.9 Showing the relation table with stability denotation

```

1 >>> rdg.showRelationTable(actionsSubset=\ 423
2 ...     ['albt','unlu','ariz','hels'],\ 424
3 ...     Sorted=False) 425
4 * ---- Relation Table ---- 426
5 r/(stab) | 'albt' 'unlu' 'ariz' 'hels' 427
6 -----|----- 428
7 'albt' | +1.00 +0.30 +0.00 +0.00 429
8 | (+4) (+2) (-1) (-1) 430
9 'unlu' | +0.00 +1.00 +0.40 +0.00 431
10 | (+0) (+4) (+2) (-1) 432
11 'ariz' | +0.00 -0.12 +1.00 +0.40 433
12 | (+1) (-2) (+4) (+2) 434
13 'hels' | +0.45 +0.00 -0.03 +1.00 435
14 | (+2) (+1) (-2) (+4) 436
15 Valuation domain: [-1.0; 1.0] 437
16 Stability denotation semantics: 438
17 +4|-4 : unanimous outranking | outranked situation; 439
18 +2|-2 : outranking | outranked situation validated 440
19 with all potential significance weights that are 441
20 compatible with the given significance preorder; 442
21 +1|-1 : validated outranking | outranked situation 443
22 with the given significance weights; 444
23 0 : indeterminate relational situation. 445

```

In Listing 13.9, we notice that the robust outranking circuit [albt, unlu, ariz, hels] will reappear with all potential criteria significance weight vectors that are compatible with given preorder: $gtch = gres > gcit > gint > gind$.

Notice also the (± 1) marked outranking situations, like the one between `albt` and `ariz`. The statement that “*Arizona State University strictly outranks University of Alberta*” is in fact valid with the precise THE criteria significance weights, but not with all potential significance weights vectors that are compatible with the given significance preorder. All these (± 1) marked outranking situations become hence *doubtful* ($r(x \succsim y) = 0.00$) and the corresponding CS Depts, like University of Alberta and Arizona State University, become *incomparable* in a robust outranking sense.

Showing many incomparabilities and indifferences; not being transitive and containing many robust outranking circuits; all these relational characteristics, make that no ranking algorithm, applied to digraph `rdg`, does exist that would produce a *unique* optimal linear ranking result. Methodologically, we are only left with ranking heuristics. In Chap. 8 on ranking with multiple criteria we have now seen several potential heuristic ranking rules that can be used for ranking with an outranking digraph; yet, they may deliver potentially more or less diverging results. Considering the order of digraph `rdg` (75) and the largely unequal THE criteria significance weights, we rather opt, in this tutorial, for the NETFLOWS ranking rule.⁷ Its complexity in $O(n^2)$ is indeed quite tractable and, by avoiding potential *tyranny of short majority* effects, the NETFLOWS rule may specifically take the ranking criteria significance weights into a more fairly balanced account.

The NETFLOWS ranking result of the CS Depts can be directly computed with the `computeNetFlowsRanking()` method.

Listing 13.10 Computing a robust NETFLOWS ranking

```

1 >>> nfRanking = rdg.computeNetFlowsRanking() 471
2 >>> nfRanking 472
3 ['ethz', 'calt', 'mit', 'oxf', 'cmel', 'git', 'epfl', 473
4 'icl', 'cou', 'tum', 'wash', 'sing', 'hkst', 'ucl', 474
5 'uiu', 'unt', 'ued', 'ntu', 'mcp', 'csd', 'cbu', 475
6 'uta', 'tsu', 'nyu', 'uwa', 'csb', 'kit', 'utj', 476
7 'bju', 'kcl', 'chku', 'kist', 'rwth', 'pud', 'epfr', 477
8 'hku', 'rcu', 'cir', 'dut', 'ens', 'ntw', 'anu', 478
9 'tub', 'mel', 'lms', 'bro', 'frei', 'wtu', 'tech', 479
10 'itmo', 'zhej', 'man', 'kuj', 'kul', 'unsw', 'glas', 480
11 'utw', 'unlu', 'naji', 'sou', 'hkpu', 'qut', 'humb', 481
12 'shji', 'stut', 'tud', 'tlavu', 'cihk', 'albt', 'indis', 482
13 'ariz', 'kth', 'hels', 'eind', 'mil'] 483

```

We actually obtain in Listing 13.10 a very similar ranking result as the one obtained with the THE average scores. The same group of seven Depts: `ethz`, `calt`, `mit`, `oxf`, `cmel`, `git`, and `epfl` is top-ranked. And a same group of Depts: `tlavu`, `cihk`, `indis`, `ariz`, `kth`, `hels`, `eind`, and `mil` appears at the end of the list.

⁷ The reader might try other ranking rules, like the COPELAND or KOHLER rules. Mind that the latter ranking-by-choosing rule is more complex (see Chap. 8).

We can print out the difference between the overall scores based THE ranking 489 and the NETFLOWS ranking above with the following short Python script, where 490 we make use of an ordered Python dictionary with net flow scores, stored in the 491 `rdg.netFlowsRankingDict` attribute by the previous computation. 492

Listing 13.11 Comparing the robust NETFLOWS ranking with the THE ranking

```

1  >>> # rdg.netFlowsRankingDict: ordered dictionary with net flow      493
2  >>> # scores stored in rdg by the computeNetFlowsRanking() method      494
3  >>> # theScores = [(xScore_1,x_1), (xScore_2,x_2),... ]      495
4  >>> # is sorted in decreasing order of xscores_i      496
5  >>> print('
6  ...   NetFlows ranking  gtch  gres  gcit  gint  gind  THE ranking')      497
7  >>> for i in range(75):      498
8  ...     x = nfRanking[i]      499
9  ...     xScore = rdg.netFlowsRankingDict[x] ['netFlow']      500
10 ...    thexScore,thex = theScores[i]      501
11 ...    print('%2d: %s (%.2f) ' % (i+1,x,xScore), end=' \t')      502
12 ...    for g in rdg.criteria:      503
13 ...        print('%.1f ' % (t.evaluation[g][x]),end=' ')      504
14 ...    print('%.2f' % (thex,thexScore))      505
15 NetFlows ranking  gtch  gres  gcit  gint  gind  THE ranking      506
16 1: ethz (116.95) 89.2 97.3 97.1 93.6 64.1 ethz (92.88)      507
17 2: calt (116.15) 91.5 96.0 99.8 59.1 85.9 calt (92.42)      508
18 3: mit (112.72) 87.3 95.4 99.4 73.9 87.5 oxf (92.20)      509
19 4: oxf (112.00) 94.0 92.0 98.8 93.6 44.3 mit (92.06)      510
20 5: cmel (101.60) 88.1 92.3 99.4 58.9 71.1 git (89.88)      511
21 6: git (93.40) 87.2 99.7 91.3 63.0 79.5 cmel (89.43)      512
22 7: epfl (90.88) 86.3 91.6 94.8 97.2 42.7 icl (89.00)      513
23 8: icl (90.62) 90.1 87.5 95.1 94.3 49.9 epfl (88.86)      514
24 9: cou (84.60) 81.6 94.1 99.7 55.7 45.7 tum (87.70)      515
25 10: tum (80.42) 87.6 95.1 87.9 52.9 95.1 sing (86.86)      516
26 11: wash (76.28) 84.4 88.7 99.3 57.4 41.2 cou (86.59)      517
27 12: sing (73.05) 89.9 91.3 83.0 95.3 50.6 ucl (86.05)      518
28 13: hkst (71.05) 74.3 92.0 96.2 84.4 55.8 wash (85.60)      519
29 14: ucl (66.78) 85.5 90.3 87.6 94.7 42.4 hkst (85.47)      520
30 15: uiu (64.80) 85.0 83.1 99.2 51.4 42.2 ntu (85.46)      521
31 16: unt (62.65) 79.9 84.4 99.6 77.6 38.4 ued (85.03)      522
32 17: ued (58.67) 85.7 85.3 89.7 95.0 38.8 unt (84.42)      523
33 18: ntu (57.88) 76.6 87.7 90.4 92.9 86.9 uiu (83.67)      524
34 19: mcp (54.08) 79.7 89.3 94.6 29.8 51.7 mcp (81.53)      525
35 20: csd (46.62) 75.2 81.6 99.8 39.7 59.8 cbu (81.25)      526
36 21: cbu (44.27) 81.2 78.5 94.7 66.9 45.7 tsu (80.91)      527
37 22: uta (43.27) 72.6 85.3 99.6 31.6 49.7 csd (80.45)      528
38 23: tsu (42.42) 88.1 90.2 76.7 27.1 85.9 uwa (80.02)      529
39 24: nyu (35.30) 71.1 77.4 99.4 78.0 39.8 nyu (79.72)      530
40 25: uwa (28.88) 75.3 82.6 91.3 72.9 41.5 uta (79.61)      531
41 26: csb (18.18) 65.6 70.9 94.8 72.9 74.9 kit (77.94)      532
42 27: kit (16.32) 73.8 85.5 84.4 41.3 76.8 bju (77.04)      533
43 28: utj (15.95) 92.0 91.7 48.7 25.8 49.6 csb (76.23)      534
44 29: bju (15.45) 83.0 85.3 70.1 30.7 99.4 rwth (76.06)      535
45 30: kcl (11.95) 45.5 94.6 86.3 95.1 38.3 hku (75.41)      536
46 31: chku (9.43) 64.1 69.3 94.7 75.6 49.9 pud (75.17)      537
47 32: kist (7.30) 79.4 88.2 64.2 31.6 92.8 kist (74.94)      538
48 33: rwth (5.00) 77.8 85.0 70.8 43.7 89.4 kcl (74.81)      539
49 34: pud (2.40) 76.9 84.8 70.8 58.1 56.7 chku (74.23)      540
50 35: epfr (-1.70) 81.7 60.6 78.1 85.3 62.9 epfr (73.71)      541
51 36: hku (-3.83) 77.0 73.0 77.0 96.8 39.5 dut (73.44)      542
52 37: rcu (-6.38) 64.1 53.8 99.4 63.7 46.1 tub (73.25)      543
53 38: cir (-8.20) 68.8 64.6 93.0 65.1 40.4 utj (72.92)      544
54 39: dut (-8.85) 64.1 78.3 76.3 69.8 90.1 cir (72.50)      545
55 40: ens (-8.97) 71.8 40.9 98.7 69.6 43.5 ntw (72.00)      546
56 41: ntw (-11.15) 81.5 79.8 66.6 25.5 67.6 anu (70.57)      547
57 42: anu (-11.50) 47.2 73.0 92.2 90.0 48.1 rcu (69.79)      548
58 43: tub (-12.20) 66.2 82.4 71.0 55.4 99.9 mel (69.67)      549

```

59	44: mel	(-23.98)	56.1	70.2	83.7	83.3	50.4	lms	(68.38)	551
60	45: lms	(-25.43)	81.5	68.1	61.0	31.1	87.8	ens	(68.35)	552
61	46: bro	(-27.18)	58.5	54.9	96.8	52.3	38.6	wtu	(67.86)	553
62	47: frei	(-34.42)	54.2	51.6	89.5	49.7	99.9	tech	(67.06)	554
63	48: wtu	(-35.05)	61.8	73.5	73.7	51.9	62.2	bro	(66.49)	555
64	49: tech	(-37.95)	54.9	71.0	85.1	51.7	40.1	man	(66.33)	556
65	50: itmo	(-38.50)	58.0	32.0	98.7	39.2	68.7	zhej	(65.34)	557
66	51: zhej	(-43.70)	73.5	70.4	60.7	22.6	75.7	frei	(65.08)	558
67	52: man	(-44.83)	63.5	71.9	62.9	84.1	42.1	unsw	(63.65)	559
68	53: kuj	(-47.40)	75.4	72.8	49.5	28.3	51.4	kuj	(62.77)	560
69	54: kul	(-49.98)	35.2	55.8	92.0	46.0	88.3	sou	(62.15)	561
70	55: unsw	(-54.88)	60.2	58.2	70.5	87.0	44.3	shJi	(61.35)	562
71	56: glas	(-56.98)	35.2	52.5	91.2	85.8	39.2	itmo	(60.52)	563
72	57: wtu	(-59.27)	38.2	52.8	87.0	69.0	60.0	kul	(60.47)	564
73	58: unlu	(-60.08)	35.2	44.2	87.4	99.7	54.1	glas	(59.78)	565
74	59: naji	(-60.52)	51.4	76.9	48.8	39.7	74.4	utw	(59.40)	566
75	60: sou	(-60.83)	48.2	60.7	75.5	87.4	43.2	stut	(58.85)	567
76	61: hkpu	(-62.05)	46.8	36.5	91.4	73.2	41.5	naji	(58.61)	568
77	62: qut	(-66.17)	45.5	42.6	82.8	75.2	63.0	tud	(58.28)	569
78	63: humb	(-68.10)	48.4	31.3	94.7	41.5	45.5	unlu	(58.04)	570
79	64: shJi	(-69.72)	66.9	68.3	62.4	22.8	38.5	qut	(57.99)	571
80	65: stut	(-69.90)	54.2	60.6	61.1	36.3	97.8	hkpu	(57.69)	572
81	66: tud	(-70.83)	46.6	53.6	75.9	53.7	66.5	albt	(57.63)	573
82	67: tlavu	(-71.50)	34.1	57.2	89.0	45.3	38.6	mil	(57.47)	574
83	68: cihk	(-72.20)	42.4	44.9	80.1	76.2	67.9	hels	(57.40)	575
84	69: albt	(-72.33)	39.2	53.3	69.9	91.9	75.4	cihk	(57.33)	576
85	70: indis	(-72.53)	56.9	76.1	49.3	20.1	41.5	tlavu	(57.19)	577
86	71: ariz	(-75.10)	28.4	61.8	84.3	59.3	42.0	indis	(57.04)	578
87	72: kth	(-77.10)	44.8	42.0	83.6	71.6	39.2	ariz	(56.79)	579
88	73: hels	(-79.55)	48.8	49.6	80.4	50.6	39.5	kth	(56.36)	580
89	74: eind	(-82.85)	32.4	48.4	81.5	72.2	45.8	humb	(55.34)	581
90	75: mil	(-83.67)	46.4	64.3	69.2	44.1	38.5	eind	(54.36)	582

The first inversion we observe in Listing 13.11 on the preceding page (Lines 18–19) concerns Oxford University and the MIT, switching positions 3 and 4. Most inversions are similarly short and concern only switching very close positions in either way. There are some slightly more important inversions concerning, for instance, the Hong Kong University CS Dept, ranked into position 30 in the THE ranking and here in the position 36 (Line 51). The opposite situation may also happen; the Berlin Humboldt University CS Dept, occupying the 74th position in the THE ranking, advances in the robust NETFLOWS ranking to position 63 (Line 78).

In our bipolar-valued epistemic framework, the NETFLOWS score of any CS Dept x corresponds to the criteria significance support for the logical statement “ x is first-ranked”. Formally

$$r(x \text{ is first-ranked}) = \sum_{y \neq x} r((x \succ y) + (y \not\succ x)) = \sum_{y \neq x} (r(x \succ y) - r(y \succ x)). \quad (13.1)$$

Using the robust outranking characteristics of digraph rdg , we can thus explicitly compute, for instance, ETH Zürich’s NETFLOWS score, denoted nfx below.

```

1 >>> x = 'ethz'
2 >>> nfx = Decimal('0')
3 >>> for y in rdg.actions:
4 ...     if x != y:
5 ...         nfx += (rdg.relation[x][y] \
```

```

6 ... - rdg.relation[y][x] ) 602
7 >>> print(x, nfx) 603
8 ethz 116.950 604

```

In Listing 13.11 on page 176 (Line 16), one may now verify that ETH Zürich obtains indeed the highest NETFLOWS score 116.95 and gives hence the *most credible* first-ranked CS Dept of the 75 potential candidates. 605
606
607

Yet, how may we now convince the reader that the outranking based ranking result here appears more objective and trustworthy than the classic value theory based THE ranking by average quantile scores? 608
609
610

13.3 How to Judge the Quality of a Ranking Result?

In a multiple-criteria based ranking problem, inspecting pairwise marginal performance differences may give objectivity to global preferential statements. That a CS Dept x convincingly outranks Dept y can conveniently be checked with the showPairwiseOutrankings() method.. The ETH Zürich CS Dept is, for instance, first ranked before Caltech's Dept in both previous rankings. Lest us check the preferential reasons. 612
613
614
615
616
617

Listing 13.12 Comparing pairwise criteria performances

```

1 >>> rdg.showPairwiseOutrankings('ethz','calt') 618
2 *----- pairwise comparisons -----* 619
3 Valuation in range: -100.00 to +100.00 620
4 Comparing actions : ('ethz', 'calt') 621
5 crit. wght. g(ethz) g(calt) diff | ind pref r() | 622
6 ----- 623
7 'gcit' 27.50 97.10 99.80 -2.70 | 2.50 5.00 +0.00 | 624
8 'gind' 5.00 64.10 85.90 -21.80 | 2.50 5.00 -5.00 | 625
9 'gint' 7.50 93.60 59.10 +34.50 | 2.50 5.00 +7.50 | 626
10 'gres' 30.00 97.30 96.00 +1.30 | 2.50 5.00 +30.00 | 627
11 'gtch' 30.00 89.20 91.50 -2.30 | 2.50 5.00 +30.00 | 628
12 ----- 629
13 r(x >= y): +62.50 630
14 crit. wght. g(calt) g(ethz) diff | ind pref r() | 631
15 ----- 632
16 'gcit' 27.50 99.80 97.10 +2.70 | 2.50 5.00 +27.50 | 633
17 'gind' 5.00 85.90 64.10 +21.80 | 2.50 5.00 +5.00 | 634
18 'gint' 7.50 59.10 93.60 -34.50 | 2.50 5.00 -7.50 | 635
19 'gres' 30.00 96.00 97.30 -1.30 | 2.50 5.00 +30.00 | 636
20 'gtch' 30.00 91.50 89.20 +2.30 | 2.50 5.00 +30.00 | 637
21 ----- 638
22 r(y >= x): +85.00 639

```

A significant positive performance difference (+34.50), concerning the *International outlook* criterion (of 7, 5% significance), is observed in Listing 13.12 in favour of the ETH Zürich Dept (Line 9 above). Similarly, a significant positive performance difference (+21.80), concerning the *Industry income* criterion (of 5% significance), is observed, this time, in favour of the Caltech Dept (Line 17). The former, larger positive, performance difference, observed on a more significant criterion, gives so far a first convincing argument of 12.5% significance for putting 640
641
642
643
644
645
646

ETH Zürich first, before Caltech. Yet, the slightly positive performance difference (+2.70, Line 16) between Caltech and ETH Zürich on the *Citations* criterion (of 27.5% significance) confirms an “*at least as well evaluated as*” situation in favour of the Caltech Dept. 647 648 649 650

The inverse negative performance difference (-2.70 , Line 7), however, is neither significant (< -5.00), nor *insignificant* (> -2.50), and does hence neither confirm nor infirm a “*not at least as well evaluated as*” situation in disfavour of ETH Zürich. We observe here a convincing argument of 27.5% significance for ranking Caltech first, before ETH Zürich.

Notice finally, that, on the *Teaching* and *Research* criteria of total significance 656
 60%, both Depts do, with performance differences (< 2.50 !), perform one as well 657
 as the other. As these two major performance criteria together necessarily admit 658
 always the highest significance with the imposed significance weight preorder: 659
 $gtch = gres > gcit > gint > gind$, both outranking situations get in fact 660
 globally confirmed at stability level +2 (see Chap. 19). 661

A browser view of the corresponding robust relation map, ordering the CS Depts again with the same NETFLOWS ranking rule, well illustrates all such *stable outranking* situations. 662
663
664

```
1 >>> rdg.showHTMLRelationMap(\n2 ...             tableTitle='Robust Outranking Map',\n3 ...             rankingRule='NetFlows')
```

In Fig. 13.2 on the following page, *dark green*, respectively, *light green* marked positions show certainly, respectively, positively valid outranking situations, whereas *dark red*, respectively, *light red* marked positions show certainly, respectively, positively valid outranked situations. In the left upper corner one may verify that the five top-ranked Depts ([ethz, calt, oxf, mit, cmel]) are in fact mutually outranking each other and thus are all indifferent one to another. They give even robust CONDORCET winners by robustly outranking all other Depts.

Notice by the way that no certainly valid robust outranking (dark green) and no certainly valid robust outranked situations (dark red) appear in Fig. 13.3 on page 181 below, respectively, above the principal diagonal; none of these robust preferential situations are hence violated by the robust NETFLOWS ranking. The non-reflexive *white* positions in the relation map, like the one between the Georgia Institute of Technology and the MIT, mark outranking or outranked situations that are not robust with respect to the given significance weight preorder. They become, hence, doubtful and are set to the indeterminate characteristic value 0.0.

By measuring the ordinal correlation with the underlying pairwise global and marginal robust outranking situations, the quality of the robust NETFLOWS ranking result can be formally evaluated with the `computeRankingCorrelation()` method and the `showCorrelation()` method (see Chap. 16).

Listing 13.13 Measuring the quality of the NETFLOWS ranking result

```
1 >>> corrnf = rdg.computeRankingCorrelation (nfRanking)
2 >>> rdg.showCorrelation (corrnf)
```

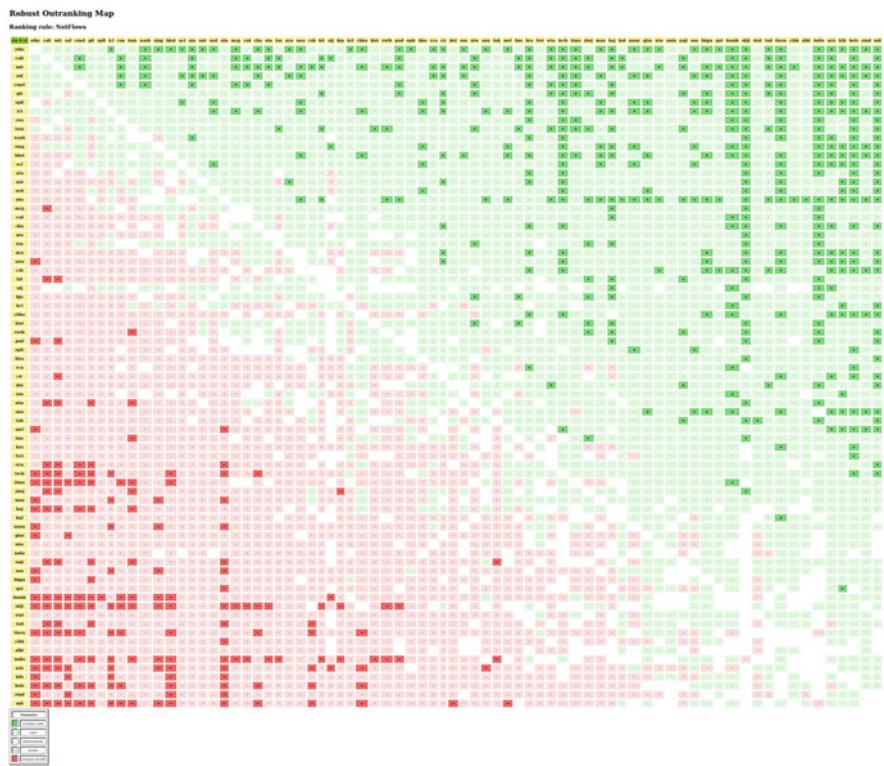


Fig. 13.2 Relation map of the robust outranking relation

```

3 Correlation indexes: 689
4 Crisp ordinal correlation : +0.901 690
5 Epistemic determination : 0.563 691
6 Bipolar-valued equivalence : +0.507 692

```

Listing 13.13 on the preceding page Line 4 indicates that the NETFLOWS ranking result is indeed highly correlated (+0.901, in KENDALL's tau index sense) with the pairwise global robust outranking relation. Their bipolar-valued *relational equivalence* index (+0.51, Line 6) indicates a more than 75% criteria significance support.

With the `showRankingConsensusQuality()` method, we can also check how the NETFLOWS ranking rule is actually balancing the five ranking objectives.

Listing 13.14 Measuring the consensus quality of the NETFLOWS ranking result

```

1 >>> rdg.showRankingConsensusQuality(nfRanking) 700
2 Criterion (weight): correlation 701
3 ----- 702
4 gtch (0.300): +0.660 703
5 gres (0.300): +0.638 704

```

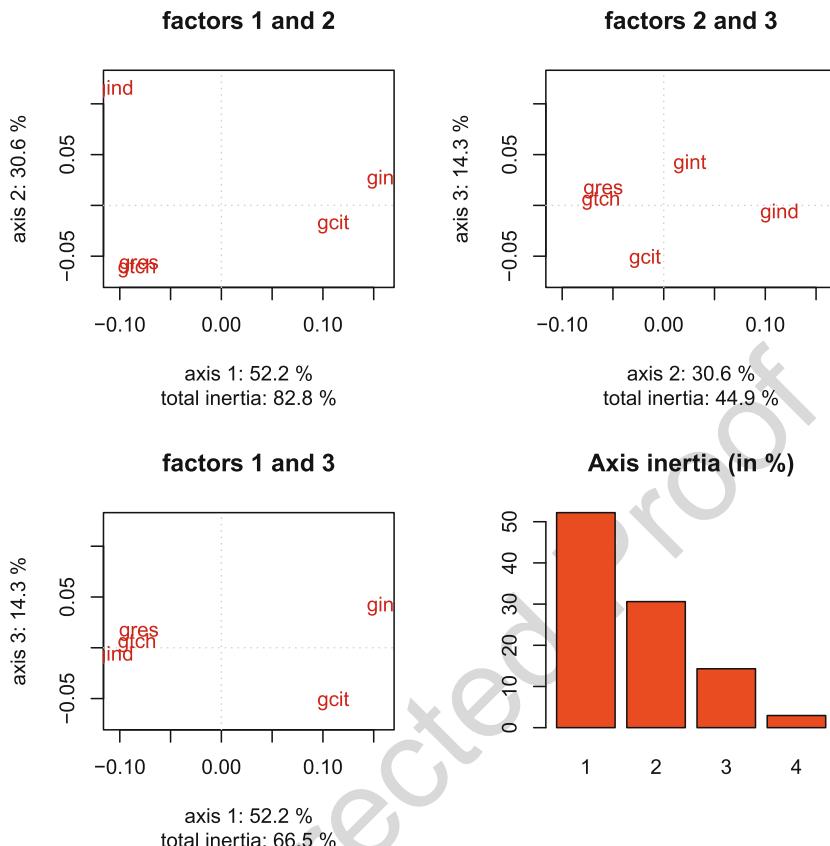


Fig. 13.3 3D PCA plot of the pairwise criteria correlation table

```

6   gct (0.275) : +0.370
7   gint (0.075) : +0.155
8   gind (0.050) : +0.101
9   Summary:
10  Weighted mean marginal correlation (a) : +0.508
11  Standard deviation (b) : +0.187
12  Ranking fairness (a) - (b) : +0.321

```

The ordinal correlation indexes with the marginal performance criterion rankings are nearly respecting the given significance weights preorder: $gtch \approx gres > gct > gint > gind$ (see Lines 4–8 above). The mean marginal ordinal correlation index is quite high (+0.51). Coupled with a low standard deviation (0.187), we obtain a quite fairly balanced ranking result (Lines 10–12).

We can furthermore inspect with the `showCriteriaCorrelationTable()` method the mutual correlation indexes observed between the individual criterion based marginal robust outranking relations.

Listing 13.15 Showing the ordinal correlation between the marginal criterion relations

```

1 >>> rdg.showCriteriaCorrelationTable() 720
2   Criteria ordinal correlation index 721
3   | gcit   gind   gint   gres   gtch 722
4   ----- 723
5   gcit | +1.00  -0.11  +0.24  +0.13  +0.17 724
6   gind |          +1.00  -0.18  +0.15  +0.15 725
7   gint |          +1.00  +0.04  -0.00 726
8   gres |          +1.00  +0.67 727
9   gtch |          +1.00 728

```

Slightly contradictory (-0.11) appear the *Citations* and *Industrial income* criteria (Line 5 Column 3 in Listing 13.15 on the previous page). Due perhaps to potential confidentiality clauses, it seems perhaps not always possible to publish industrially relevant research results in highly ranked journals. However, criteria *Citations* and *International outlook* show a slightly positive correlation ($+0.24$, Column 4), whereas the *International outlook* criterion shows no apparent correlation with both the major *Teaching* and *Research* criteria. The latter are, however, highly correlated ($+0.67$. Line 9 Column 6).

A Principal Component Analysis may well illustrate with the `export3DplotOfCriteriaCorrelation()` method the previous findings (Bisdorff 2020).

```

1 >>> rdg.export3DplotOfCriteriaCorrelation(graphType='pdf') 739

```

In Fig. 13.3 on the preceding page, one may notice first that more than 80% of the total variance of the previous correlation table is explained by the apparent opposition between the marginal outrankings of criteria: *Teaching*, *Research*, and *Industry income* on the left side, and the marginal outrankings of criteria: *Citations* and *International outlook* on the right side. Notice also in the left lower corner the nearly identical positions of the marginal outrankings of the major *Teaching* and *Research* criteria. In the factors 2 and 3 plot, about 30% of the total variance is captured by the opposition between the marginal outrankings of the *Teaching* and *Research* criteria and the marginal outrankings of the *Industrial income* criterion. Finally, in the factors 1 and 3 plot, nearly 15% of the total variance is explained by the opposition between the marginal outrankings of the *International outlook* criterion and the marginal outrankings of the *Citations* criterion.

It is, finally, interesting to similarly assess the ordinal correlation between the THE average scores-based ranking and the robust outranking digraph.

Listing 13.16 Computing the ordinal quality of the THE ranking

```

1 >>> # theScores = [(xScore_1,x_1), (xScore_2,x_2),... ] 754
2 >>> # is sorted in decreasing order of xscores 755
3 >>> theRanking = [item[1] for item in theScores] 756
4 >>> corrthe = rdg.computeRankingCorrelation(theRanking) 757
5 >>> rdg.showCorrelation(corrthe) 758
6   Correlation indexes: 759
7   Crisp ordinal correlation : +0.907 760
8   Epistemic determination   : 0.563 761
9   Bipolar-valued equivalence : +0.511 762

```

```

10 >>> rdg.showRankingConsensusQuality(theRanking) 763
11     Criterion (weight): correlation 764
12 -----
13     gtch (0.300): +0.683 765
14     gres (0.300): +0.670 766
15     gcit (0.275): +0.319 767
16     gint (0.075): +0.161 768
17     gind (0.050): +0.106 769
18 Summary: 770
19     Weighted mean marginal correlation (a): +0.511 771
20     Standard deviation (b) : +0.210 772
21     Ranking fairness (a) - (b) : +0.302 773
22

```

The THE ranking result is similarly correlated (+0.907, Line 7 in Listing 13.16) with the pairwise global robust outranking situations. By its overall weighted scoring rule, the THE ranking naturally induces marginal criterion correlations that are compatible with the given significance weight preorder (Lines 13–17). Notice that the mean marginal correlation is of a similar value (+0.51, Line 19) as the robust NETFLOWS ranking. Yet, its standard deviation is slightly higher, which leads to a less fairer balancing of the three major ranking criteria.

To conclude, let us emphasise, that, without any commensurability hypothesis and by taking, furthermore, into account, first, the always present more or less imprecision of any performance evaluation and, secondly, solely ordinal criteria significance weights, we may obtain here with our robust outranking approach a very similar ranking result with a slightly better preference modelling quality. A convincing heatmap view of the 25 first-ranked Institutions may eventually be generated in the default system browser with following command (Fig. 13.4 on the next page).

```

1 >>> rdg.showHTMLPerformanceHeatmap ( 790
2 ...     WithActionNames=True, \ 791
3 ...     outrankingModel='this', \ 792
4 ...     rankingRule='NetFlows', \ 793
5 ...     ndigits=1, \ 794
6 ...     Correlations=True, \ 795
7 ...     fromIndex=0,toIndex=25) 796

```

As an exercise, the reader is invited to try out other robust outranking based ranking heuristics. Notice also that we have not challenged in this case study the THE provided criteria significance preorder. It would be very interesting to consider the five ranking objectives as equally important and, consequently, consider the ranking criteria to be equi-significant. Curious to see the ranking results under such settings.

The next case study in Chap. 14 is tackling the absolute quantile rating of the student enrolment quality of higher education institutions.

Heatmap of Performance Tableau 'robust_the_cs_2016'

criteria	gtch	gres	gcit	gint	gind
weights	+30.00	+30.00	+27.50	+7.50	+5.00
tau(*)	+0.66	+0.64	+0.37	+0.15	+0.10
Swiss Federal Institute of Technology Zürich (ethz)	89.2	97.3	97.1	93.6	64.1
Californiia Institute of Technology (calt)	91.5	96.0	99.8	59.1	85.9
Massachusetts Institute of Technology (mit)	87.3	95.4	99.4	73.9	87.5
University of Oxford (oxf)	94.0	92.0	98.8	93.6	44.3
Carnegie Mellon University (cmel)	88.1	92.3	99.4	58.9	71.1
Georgia Institute of Technology (git)	87.2	99.7	91.3	63.0	79.5
Swiss Federal Institute of Technology Lausanne (epfl)	86.3	91.6	94.8	97.2	42.7
Imperial College London (icl)	90.1	87.5	95.1	94.3	49.9
Cornell University (cou)	81.6	94.1	99.7	55.7	45.7
Technical University of München (tum)	87.6	95.1	87.9	52.9	95.1
University of Washington (wash)	84.4	88.7	99.3	57.4	41.2
National University of Singapore (sing)	89.9	91.3	83.0	95.3	50.6
Hong Kong University of Science and Technology (hkst)	74.3	92.0	96.2	84.4	55.8
University College London (ucl)	85.5	90.3	87.6	94.7	42.4
University of Illinois at Urbana-Champagne (uiu)	85.0	83.1	99.2	51.4	42.2
University of Toronto (unt)	79.9	84.4	99.6	77.6	38.4
University of Edinburgh (ued)	85.7	85.3	89.7	95.0	38.8
Nanyang Technological University of Singapore (ntu)	76.6	87.7	90.4	92.9	86.9
University of Maryland College Park (mcp)	79.7	89.3	94.6	29.8	51.7
University Of California at San Diego (csd)	75.2	81.6	99.8	39.7	59.8
Columbia University (cbu)	81.2	78.5	94.7	66.9	45.7
University of Texas at Austin (uta)	72.6	85.3	99.6	31.6	49.7
Tsinghua University (tsu)	88.1	90.2	76.7	27.1	85.9
New York University (nyu)	71.1	77.4	99.4	78.0	39.8
University of Waterloo (uwa)	75.3	82.6	91.3	72.9	41.5

Color legend:

quantile 14.29% 28.57% 42.86% 57.14% 71.43% 85.71% 100.00%

(*) tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation

Outranking model: this, Ranking rule: NetFlows

Ordinal (Kendall) correlation between global ranking and global outranking relation: +0.901

Mean marginal correlation (a) : +0.508

Standard marginal correlation deviation (b) : +0.187

Ranking fairness (a) - (b) : +0.321

Fig. 13.4 Extract of a heatmap browser view on the NETFLOWS ranking result

References

805

- Bisdorff R (2004) Concordant outranking with multiple criteria of ordinal significance. 4OR 2(4):293–308. <http://hdl.handle.net/10993/23721> 806
807
- Bisdorff R (2010) Enumerating chordless circuits in directed graphs. In: ORBEL24-2010, 24th annual conference of the Belgian operational research society (ORBEL aka Sogesci-B.V.W.B.), January 28-29, Liège (BE), Université de Liège (BE), pp 1–12. <http://hdl.handle.net/10993/23926> 808
809
810
811
- Bisdorff R (2013) On polarizing outranking relations with large performance differences. J Multi-Crit Decis Anal Wiley 20:3–12. <http://hdl.handle.net/10993/245> 812
813

Bisdorff R (2020) Lecture 2: Introduction to statistical computing. In: Lectures of the Computational Statistics Course, University of Luxembourg. http://hdl.handle.net/10993/37870	814
	815
Bisdorff R (2021) Documentation of the Digraph3 collection of Python modules for Algorithmic Decision Theory. https://digraph3.readthedocs.io/en/latest/	816
	817
Times Higher Education (2016) World University Rankings by Computer Science Subject. http://www.timeshighereducation.com/world-university-rankings/methodology-world-university-rankings-2016-2017	818
	819
	820

Uncorrected Proof

AUTHOR QUERIES

- AQ1. Please provide complete heading for Listing 13.1
- AQ2. Please check the sentence “Showing many incomparabilities and. . .” for clarity.
- AQ3. Missing citation for “Fig. 13.4” was inserted here. Please check if appropriate. Otherwise, please provide citation for “Fig. 13.4”. Note that the order of main citations of figures in the text must be sequential.

Uncorrected Proof

Chapter 14	1
The Best Students, Where Do They	2
Study? A Rating Case Study	3

Contents	4
14.1 The Rating Problem	187 5
14.2 The 2004 Performance Quintiles	189 6
14.3 Rating-by-Ranking with Lower-Closed Quintile Limits	191 7
14.4 Rating by Quintiles Sorting	196 8

Abstract In 2004, the German magazine *Der Spiegel*, with the help of *McKinsey & Company* and *AOL*, conducted an extensive online survey, assessing the apparent quality of German University students. The eventually published results by the *Spiegel* magazine concerned nearly 50,000 students, enrolled in one of fifteen popular academic subjects, like *German Studies, Life Sciences, Psychology, Law* or *CS*. Based on this published data, we present and discuss in this chapter, how to rate with the help of our DIGRAPH3 software resources the apparent global *enrolment quality* of new performance records.

14.1 The Rating Problem

In the 2004 DER SPIEGEL survey, more than 80,000 students, by participating, were questioned on their ‘*Abitur*’ and university exams’ marks, time of studies and age, grants, awards and publications, IT proficiency, linguistic skills, practical work experience, foreign mobility, and civil engagement. Each student received in return a quality score through a specific weighing of the collected data which depended on the subject the student is mainly studying. Publishing only those subject-university combinations, where at least 18 students had correctly filled in the questionnaire, left 41 German universities where, for at least eight out of the fifteen subjects, an average enrolment quality score could be determined (DER SPIEGEL 2004; Friedmann et al. 2004).

Table 14.1 Enrolment quality scores per academic scores

Subject	U1	U2	U3	U4	U5	
bio	NA	53.10	49.70	52.20	55.20	t2.1
med	NA	NA	NA	49.50	55.50	t2.2
math	56.80	54.70	56.30	58.60	61.30	t2.3
phys	58.90	59.80	53.90	59.10	60.90	t2.4
chem	52.00	50.10	54.20	53.60	56.70	t2.5
germ	51.40	53.50	51.40	53.30	61.40	t2.6
pol	NA	54.00	NA	50.80	59.60	t2.7
soc	59.10	51.50	55.60	51.00	52.20	t2.8
psy	57.70	NA	54.40	62.70	59.80	t2.9
law	NA	NA	41.90	NA	51.10	t2.10
eco	49.60	NA	NA	NA	54.40	t2.11
mgt	54.00	53.40	50.70	49.60	NA	t2.12
info	55.40	52.60	55.80	54.60	NA	t2.13
elec	56.10	54.50	NA	57.20	NA	t2.14
mec	54.30	55.20	NA	54.40	NA	t2.15
						t2.16

We suppose in this rating case study that five German universities: U1, U2, U3, U4, and U5 conducted in 2005 a similar survey among their enrolled students which gave the following enrolment quality scores per academic subject.

In Table 14.1, the fifteen popular academic subjects are grouped into topical ‘Faculties’: *Humanities*, *Law*, *Economics & Management*, *Life Sciences & Medicine*, *Natural Sciences & Mathematics*, and *Technology*. None of the five universities have students enrolled in all the fifteen subjects. University U1 has, for instance, no students in Life Sciences & Medicine and in Law and Politology, whereas University U5 does not offer any Technology subjects.

The average enrolment quality scores of the five universities, shown in Table 14.1, are stored in a file named `ratingCaseStudy.py` of `PerformanceTableau` format.¹ The `showHTMLPerformanceTableau()` method produces in Fig. 14.1 on the facing page a colourful browser view of these performance records.

```

1 >>> from perfTabs import PerformanceTableau
2 >>> pt = PerformanceTableau('ratingCaseStudy')
3 >>> pt.showHTMLPerformanceTableau(Transposed=True, \
4 ...                                     title='Average enrolment scores')

```

With the best score in nine out of fifteen subjects, university U5 presents the highest global enrolment quality of the five, whereas University U3, with five lowest scores, shows the lowest student enrolment quality of the five.

The university administrations would like to know now how their respective enrolment quality scores are to be appreciated in view the results of the 2004 DER

¹ The file may be found in the `examples` directory of the DIGRAPH3 resources.

Fig. 14.1 Student enrolment quality scores per subject.

The light green, respectively, light red, figures indicate highest, respectively, lowest, score among the five universities

Average enrolment scores

criterion	U1	U2	U3	U4	U5
bio	NA	53.10	49.70	52.20	55.20
chem	52.00	50.10	54.20	53.60	56.70
eco	49.60	NA	NA	NA	54.40
elec	56.10	54.50	NA	57.20	NA
germ	51.40	53.50	51.40	53.30	61.40
info	55.40	52.60	55.80	54.60	NA
law	NA	NA	41.90	NA	51.10
math	56.80	54.70	56.30	58.60	61.30
mec	54.30	55.20	NA	54.40	NA
med	NA	NA	NA	49.50	55.50
mgt	54.00	53.40	50.70	49.60	NA
phys	58.90	59.80	53.90	59.10	60.90
pol	NA	54.00	NA	50.80	59.50
psy	57.70	NA	54.40	62.70	59.80
soc	59.10	51.50	55.60	51.00	52.20

SPIEGEL survey. Are they among the top universities, the midfield, or the bottom 51 group? 52

14.2 The 2004 Performance Quintiles

53

The estimated lower-closed performance quintiles of the 2004 average enrolment 54 quality per academic subject are stored in a file named historicalQuan- 55 tiles.py,² which can be reloaded with the PerformanceQuantiles class. 56

Listing 14.1 Inspecting stored historical performance quintiles

```

1  >>> from performanceQuantiles import PerformanceQuantiles      57
2  >>> pq = PerformanceQuantiles('historicalQuintiles')           58
3  >>> pq
4  *---- PerformanceQuantiles instance description ---*          60
5  Instance class      : PerformanceQuantiles                      61
6  Instance name       : historicalQuintiles                      62
7  Objectives          : 4                                         63
8  Criteria            : 15                                         64
9  Quantiles           : 5                                         65
10 History sizes      : {'germ': 39, 'pol': 34, 'psy': 34,        66
11                  'soc': 32, 'law': 32, 'eco': 21,          67
12                  'mgt': 34, 'bio': 34, 'med': 28,          68
13                  'phys': 37, 'chem': 35, 'math': 27,        69

```

² The file may be found in the examples directory of the DIGRAPH3 resources.

#	Identifier	Name	Comment	Weight	Scale		Thresholds (ax + b)			
					direction	min	max	Indifference	preference	veto
1	bio	Life Sciences	Life Sciences & Medicine	1.00	max	45.00	65.00	0.00x + 0.10	0.00x + 0.50	
2	chem	Chemistry	Natural Sciences & Mathematics	1.00	max	45.00	65.00	0.00x + 0.10	0.00x + 0.50	
3	eco	Economics	Law, Economics & Management	1.00	max	45.00	65.00	0.00x + 0.10	0.00x + 0.50	
4	elec	Electrical Engineering	Technology	1.00	max	45.00	65.00	0.00x + 0.10	0.00x + 0.50	
5	germ	German Studies	Humanities	1.00	max	45.00	65.00	0.00x + 0.10	0.00x + 0.50	
6	info	Computer Science	Technology	1.00	max	45.00	65.00	0.00x + 0.10	0.00x + 0.50	
7	law	Law Studies	Law, Economics & Management	1.00	max	35.00	65.00	0.00x + 0.10	0.00x + 0.50	
8	math	Mathematics	Natural Sciences & Mathematics	1.00	max	45.00	65.00	0.00x + 0.10	0.00x + 0.50	
9	mec	Mechanical Engineering	Technology	1.00	max	45.00	65.00	0.00x + 0.10	0.00x + 0.50	
10	med	Medicine	Life Sciences & Medicine	1.00	max	45.00	65.00	0.00x + 0.10	0.00x + 0.50	
11	mgt	Management	Law, Economics & Management	1.00	max	40.00	80.00	0.00x + 0.10	0.00x + 0.50	
12	phys	Physics	Natural Sciences & Mathematics	1.00	max	45.00	65.00	0.00x + 0.10	0.00x + 0.50	
13	pol	Politology	Humanities	1.00	max	50.00	70.00	0.00x + 0.10	0.00x + 0.50	
14	psy	Psychology	Humanities	1.00	max	50.00	70.00	0.00x + 0.10	0.00x + 0.50	
15	soc	Sociology	Humanities	1.00	max	45.00	65.00	0.00x + 0.10	0.00x + 0.50	

Fig. 14.2 The fifteen academic subjects taken into account for assessing the student enrolment quality

```

14           'info': 33, 'elec': 14, 'mec': 13}           70
15   Attributes : ['name', 'objectives', 'NA', 'criteria',      71
16           'quantilesFrequencies', 'historySizes',      72
17           'LowerClosed', 'limitingQuantiles',      73
18           'cdf', 'perfTabType']           74

```

The history sizes, reported in Listing 14.1 on the previous page, indicate the number of universities evaluated in the 2004 survey in each one of the popular fifteen subjects. German Studies, for instance, were evaluated for 39 out of 41 universities, whereas Electrical and Mechanical Engineering were only evaluated for 14, respectively, 13 institutions. None of the fifteen subjects were evaluated in all the 41 universities.³

The details of the fifteen academic subjects—the performance criteria—may be consulted in a browser view (see Fig. 14.2).

```

1 >>> pt.showHTMLCriteria()           83

```

All fifteen subjects are considered equally significant (see Column Weight). The average scores in most subjects vary from 45 to 65. In some subjects, however, like Law Studies (35.0–65.0) and Politology (50.0–70.0), a different variability is observed. The average enrolment scores per subject are hence incommensurable and global average enrolment scores over all subjects become meaningless.

To take furthermore into account the potential and very likely imprecision of the quality scores' computation, we assume that, for all subjects, an average enrolment quality score difference of 0.1 is *insignificant*, whereas a difference of 0.5 is sufficient to positively attest a *better* enrolment quality. No considerable performance difference is assumed.

³ It would have been interesting to estimate such quantile limits from the individual quality scores of all the nearly 50,000 surveyed students. But this data was not public (Friedmann et al. 2004).

The `showLimitingQuantiles()` method prints in Listing 14.2 the estimated quintile limits of the 2004 survey. 94
95

Listing 14.2 Estimated quintile limits of the 2004 survey

96								
97								
98	weights	'0.00'	'0.20'	'0.40'	'0.60'	'0.80'	'1.00'	
99								
5	'bio'	1.0	45.00	50.40	51.80	53.14	55.04	57.10
6	'chem'	1.0	45.00	53.30	54.20	55.80	57.40	58.80
7	'eco'	1.0	49.60	52.14	53.38	54.28	56.94	60.80
8	'elec'	1.0	50.10	54.08	55.34	56.54	57.64	60.20
9	'germ'	1.0	45.00	52.14	54.02	55.84	57.48	61.40
10	'info'	1.0	45.00	54.40	55.44	56.68	58.10	59.80
11	'law'	1.0	39.10	42.30	45.08	46.30	47.26	51.10
12	'math'	1.0	51.60	56.54	57.76	59.44	61.00	63.10
13	'mec'	1.0	51.90	54.02	54.48	55.18	56.54	57.80
14	'med'	1.0	45.00	49.20	49.84	51.10	52.42	60.10
15	'mgt'	1.0	47.50	52.16	52.98	54.68	55.96	68.00
16	'phys'	1.0	53.90	58.78	59.90	60.96	61.96	62.80
17	'pol'	1.0	50.80	54.62	56.18	57.78	59.78	65.90
18	'psy'	1.0	52.50	57.58	58.46	59.80	60.94	64.10
19	'soc'	1.0	45.00	51.70	53.92	55.42	56.26	59.80

We see confirmed again the incommensurability between the subjects, we noticed already in the apparent enrolment quality scoring, especially between Law Studies (39.1–51.1) and Politology (50.5–65.9).⁴ 115
116
117

14.3 Rating-by-Ranking with Lower-Closed Quintile Limits

118

We add, now, the estimated quintile limits to the enrolment quality records of the 5 universities and rank, by using the COPELAND rule, all these records conjointly together with the help of the `LearnedQuantilesRatingDigraph` class from the `sortingDigraphs` module. 119
120
121
122

1	>>> from sortingDigraphs import \	123
2	LearnedQuantilesRatingDigraph	124
3	>>> lqr = LearnedQuantilesRatingDigraph(pq,pt,\	125
4	rankingRule='Copeland')	126
5	>>> lqr	127
6	*---- Object instance description -----*	128
7	Instance class : LearnedQuantilesRatingDigraph	129
8	Instance name : learnedRatingDigraph	130
9	Criteria : 15	131
10	Quantiles : 5	132
11	Lower-closed bins : True	133
12	New actions : 5	134
13	Size : 44	135

⁴The *Spiegel* authors opted therefore for a simple 3-tiling of the universities per evaluated academic subject, followed by an average BORDA scores-based global ranking (Friedmann et al. 2004).

Heatmap of Performance Tableau 'learnedRatingDigraph'

Ranking rule: Copeland; Ranking correlation: 0.973

criteria	phys	math	germ	chem	bio	mgt	psy	info	pol	law	elec	med	eco	mec	soc
weights	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
tau ^(*)	+0.79	+0.77	+0.77	+0.66	+0.62	+0.59	+0.54	+0.52	+0.46	+0.41	+0.41	+0.40	+0.39	+0.38	+0.36
[0.80 -]	62.0	61.0	57.5	57.4	55.0	56.0	60.9	58.1	59.8	47.3	57.6	52.4	56.9	56.5	56.3
U5	60.9	61.3	61.4	56.7	55.2	NA	59.8	NA	59.5	51.1	NA	55.5	54.4	NA	52.2
[0.60 -]	61.0	59.4	55.8	55.8	53.1	54.7	59.8	56.7	57.8	46.3	56.5	51.1	54.3	55.2	55.4
[0.40 -]	59.9	57.8	54.0	54.2	51.8	53.0	58.5	55.4	56.2	45.1	55.3	49.8	53.4	54.5	53.9
U1	58.9	56.8	51.4	52.0	NA	54.0	57.7	55.4	NA	NA	56.1	NA	49.6	54.3	59.1
U4	59.1	58.6	53.3	53.6	52.2	49.6	62.7	54.6	50.8	NA	57.2	49.5	NA	54.4	51.0
U2	59.8	54.7	53.5	50.1	53.1	53.4	NA	52.6	54.0	NA	54.5	NA	NA	55.2	51.5
[0.20 -]	58.8	56.5	52.1	53.3	50.4	52.2	57.6	54.4	54.6	42.3	54.1	49.2	52.1	54.0	51.7
U3	53.9	56.3	51.4	54.2	49.7	50.7	54.4	55.8	NA	41.9	NA	NA	NA	NA	55.6
[0.00 -]	53.9	51.6	1.0	2.0	1.0	47.5	52.5	3.0	50.8	39.1	50.1	3.0	49.6	51.9	1.0

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%
----------	--------	--------	--------	--------	---------

(*) tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation.

Fig. 14.3 Heatmap view of the quintiles rating-by-ranking result

```
14  Determinateness (%) : 77.6
15  Ranking rule       : Copeland
16  Ordinal correlation : +0.97
```

The resulting ranking of the 5 universities including the lower-closed quintile score limits may be well illustrated with the `showHTMLRatingHeatmap()` method.

```
1 >>> lqr.showHTMLRatingHeatmap(colorLevels=5, \
2 ...           Correlations=True, \
3 ...           ndigits=1, rankingRule='Copeland')
```

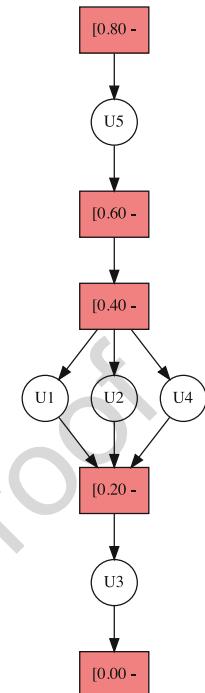
The ordinal correlation (+0.97) of the COPELAND ranking with the underlying bipolar-valued outranking digraph is very high (see Fig. 14.3 Row 1). Most correlated subjects with this *rating-by-ranking* result appear to be Physics (+0.79), Mathematics (+0.77), and German Studies (+0.77). Sociology (+0.36) is the less correlated subject (see Row 4).

From the actual ranking position of the lower 5-tiling limits, we may now immediately deduce the quintile enrolment quality equivalence classes. No university reaches the highest quintile ([0.80 –]), whereas U3 is rated into the lowest quintile ([0.00 – 0.20]). The other three universities U1, U2, and U4 are rated in the second quintile ([0.20 – 0.40]). The rating result may be easily printed out with the `showQuantilesRating()` method.

Listing 14.3 Showing the quintiling of the enrolment quality of the 5 universities

```
1 >>> lqr.showQuantilesRating()
```

Fig. 14.4 Drawing of the quintiles rating-by-ranking result



Digraph3 (graphviz)
R. Bisendorff, 2020

```

2     *---- Quantiles rating result ----*
3     [0.60 - 0.80[ ['U5']
4     [0.20 - 0.40[ ['U1', 'U4', 'U2']
5     [0.00 - 0.20[ ['U3']
  
```

A corresponding *graphviz* drawing with the special `exportRatingByRankingGraphViz()` method may well illustrate in Fig. 14.4 these enrolment quality equivalence classes.

```

1 >>> lqr.exportRatingByRankingGraphViz('ratingResult')           164
2     *---- exporting a dot file for GraphViz tools -----*      165
3     Exporting to ratingResult.dot                                166
4     dot -Grankdir=TB -Tpdf dot -o ratingResult.png             167
  
```

We have noticed in Chap. 8 that there is not a unique optimal rule for ranking from a given outranking digraph, like `digraph lqr` here. The COPELAND rule, for instance, has the advantage of being CONDORCET consistent, i.e. when the outranking digraph models a linear ranking, this ranking will necessarily be the result of the COPELAND rule. When this is not the case, and especially when the outranking digraph shows chordless circuits, all potential ranking rules may give very divergent ranking results and sometimes substantially divergent rating-by-ranking results.

The `computeChordlessCircuits()` and `showChordlessCircuits()` methods allow to check this issue.⁵ 175
methods allow to check this issue. 176

```
1 >>> lqr.computeChordlessCircuits() 177
2 >>> lqr.showChordlessCircuits() 178
3 *---- Chordless circuits ----* 179
4 1 circuits. 180
5 1: ['U1', 'U2', 'U4'] , credibility : 0.200 181
```

Indeed, there appears an outranking circuit among the three universities U1, 182
U2, and U4 rated into the 2nd quintile. It is, hence, interesting, to verify if the 183
epistemic fusion of the rating-by-ranking results, one may obtain when applying 184
three different ranking rules, like the KEMENY, COPELAND, and NETFLOWS 185
rules, does actually confirm our rating-by-ranking result shown in Fig. 14.4 186
on the preceding page. For this purpose, we make use in Listing 14.4 of the 187
RankingsFusionDigraph class. 188

Listing 14.4 Computing the epistemic fusion of three rating-by-ranking results

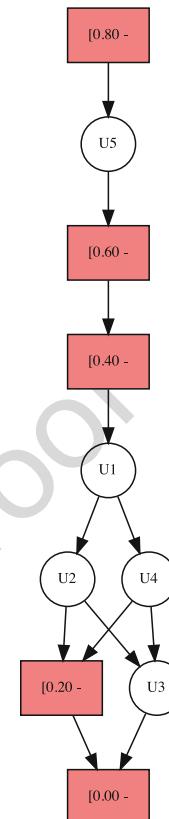
```
1 >>> # lqr.actionsRanking is Copeland ranked 189
2 >>> from linearOrders import KemenyOrder,\ 190
3 ... NetFlowsOrder 191
4 >>> ke = KemenyOrder(lqr,orderLimit=10) 192
5 >>> nf = NetFlowsOrder(lqr) 193
6 >>> from transitiveDigraphs import\ 194
7 ... RankingsFusionDigraph 195
8 >>> rankings = [lqr.actionsRanking,\ 196
9 ... nf.netFlowsRanking,\ 197
10 ... ke.kemenyRanking] 198
11 >>> rankings 199
12 [[ 'm5', 'u5', 'm4', 'm3', 'u1', 'u2', 'u4', 'm2', 'u3', 'm1'], 200
13 [ 'm5', 'u5', 'm4', 'm3', 'u1', 'u4', 'u2', 'u3', 'm2', 'm1'], 201
14 [ 'm5', 'u5', 'm4', 'm3', 'u1', 'u4', 'u2', 'm2', 'u3', 'm1']] 202
15 >>> rf = RankingsFusionDigraph(lqr,rankings) 203
16 >>> rf.exportGraphViz(fileName='fusionResult',\ 204
17 ... WithRatingDecoration=True) 205
18 exporting a dot file for GraphViz tools 206
19 Exporting to fusionResult.dot 207
20 dot -Grankdir=TB -Tpng fusionResult.dot\ 208
... -o fusionResult.png 209
```

The fusion of the three rating-by-ranking results is shown in Fig. 14.5 on 210
the facing page where we see that they diverge solely in their rating-by-ranking 211
of Universities U2, U3 and U4. In Listing 14.4, Lines 12–14, one may more 212
precisely notice that the KEMENY and COPELAND rules rate U3 in the first quintile 213
([0.00 – 0.20]), whereas the NETFLOWS rule puts U3 together with U1, U2, and U4 214

⁵ The `computeChordlessCircuits()` and `showChordlessCircuits()` methods are separate because there are various methods available for enumerating the chordless circuits in a digraph (Bisdorff 2010).

Fig. 14.5 Disjunctive fusion of the KEMENY, COPELAND and NETFLOWS rankings.

They diverge in their rating-by-ranking of universities U2, U3, and U4



Digraph3 (graphviz)
R. Bischoff, 2020

into the second quintile ([0.20 – 0.40]). And, the KEMENY rule inverts the position of U2 and U4.

In Listing 14.5, the `showRankingConsensusQuality()` method reveals finally how fairly KEMENY and COPELAND rules actually balance the fifteen academic subjects.

Listing 14.5 Checking the consensus quality of the KEMENY ranking

```

1 >>> lqr.showRankingConsensusQuality(ke.kemenyRanking) 220
2 Consensus quality of ranking: 221
3 ['m5', 'u5', 'm4', 'm3', 'u1', 'u2', 'u4', 'm2', 'u3', 'm1'] 222
4 criterion (weight): correlation 223
5 -----
6 phys (0.067): +0.833 224
7 germ (0.067): +0.789 225
8 math (0.067): +0.722 226
9 bio (0.067): +0.667 227
10 mgt (0.067): +0.633 228
11 chem (0.067): +0.611 229
12 psy (0.067): +0.544 230
13 pol (0.067): +0.500 231
  
```

14	info (0.067): +0.478	233
15	mec (0.067): +0.422	234
16	law (0.067): +0.411	235
17	soc (0.067): +0.400	236
18	med (0.067): +0.400	237
19	eco (0.067): +0.389	238
20	elec (0.067): +0.367	239
21	Summary:	240
22	Weighted mean marginal correlation (a): +0.544	241
23	Standard deviation (b) : +0.150	242
24	Ranking fairness (a)-(b) : +0.394	243

Listing 14.6 Checking the consensus quality of the COPELAND ranking

1	>>> lqr.showRankingConsensusQuality(lqr.actionsRanking)	244
2	Consensus quality of ranking:	245
3	['m5', 'u5', 'm4', 'm3', 'u1', 'u4', 'u2', 'm2', 'u3', 'm1']	246
4	criterion (weight): correlation	247
5	-----	248
6	phys (0.067): +0.789	249
7	math (0.067): +0.767	250
8	germ (0.067): +0.767	251
9	chem (0.067): +0.656	252
10	bio (0.067): +0.622	253
11	mgt (0.067): +0.589	254
12	psy (0.067): +0.544	255
13	info (0.067): +0.522	256
14	pol (0.067): +0.456	257
15	law (0.067): +0.411	258
16	elec (0.067): +0.411	259
17	med (0.067): +0.400	260
18	eco (0.067): +0.389	261
19	mec (0.067): +0.378	262
20	soc (0.067): +0.356	263
21	Summary:	264
22	Weighted mean marginal correlation (a): +0.537	265
23	Standard deviation (b) : +0.148	266
24	Ranking fairness (a)-(b) : +0.389	267

The consensus quality of the two rating-by-ranking results does not sensibly differ one of the other. They both show a similar high mean marginal correlation (+0.544, +0.537), similar standard deviations (+0.150, +0.148), and hence a similar ranking fairness (+0.394, +0.389).

To furthermore check the quality of our COPELAND rating-by-ranking result, we shall below compute a direct rating-by-sorting into the historic 2004 quintiles of the enrolment quality scores, without making use of any outranking digraph-based ranking rule.

14.4 Rating by Quintiles Sorting

276

In the case study here, the five universities U1, U2, U3, U4, and U5 represent the decision actions: *where to study*. We say now that University x is sorted into the lower-closed quintile k when the performance record of x positively outranks the lower limit record $\mathbf{q}(p_{k-1})$ of quintile q and x does not positively outrank the

upper limit record $\mathbf{q}(p_k)$ of quintile q , for $q = 1, \dots, 5$ (see Listing 14.1 on page 189).

With the help of the bipolar-valued characteristic of the outranking relation $r(x \succsim y)$, we may indeed compute the bipolar-valued characteristic of the assertion: ' x belongs to the lower-closed quintile class q_k ':

$$r(x \in \mathbf{q}_k) = \min [r(x \gtrsim \mathbf{q}(p_{k-1})), r(x \not\gtrsim \mathbf{q}(p_k))], \quad (14.1)$$

where $k = 1, \dots, 5$ and (p_k) denote the respective quintile proportions: 0.20, 0.40, 0.60, 0.80, and 1.00. As bipolar-valued outranking digraphs verify the coduality principle, $r(x \not\sim \mathbf{q}(p_k)) = r(x \text{precsim} \mathbf{q}(p_k))$ (see Sect. 9.2). 286
287
288

The `showSortingCharacteristics()` method gives a precise look in Listing 14.7 on these quintiles sorting characteristics. 290

Listing 14.7 Showing quantiles sorting characteristics

The performance record of University U5 cannot be positively rated into a precise 322 quintile, but both the fourth and the fifth quintiles are not positively excluded 323 as rating result. Otherwise, the bipolar-valued sorting characteristics verify the 324 rating-by-ranking result we obtained previously with the KEMENY and COPELAND 325 rating-by-ranking result. The `showActionsSortingResult()` method prints 326 the eventual quintiles rating result: 327

Listing 14.8 Showing a quintiles rating-by-sorting result

```

1 >>> lqr.showActionsSortingResult () 328
2   Quantiles sorting result per decision action 329
3   [0.20 - 0.40[: U1 with credibility: 0.13 = min(0.33,0.13) 330
4   [0.20 - 0.40[: U2 with credibility: 0.13 = min(0.13,0.20) 331
5   [0.00 - 0.20[: U3 with credibility: 0.13 = min(0.67,0.13) 332
6   [0.20 - 0.40[: U4 with credibility: 0.13 = min(0.47,0.13) 333
7   [0.60 - <[: U5 with credibility: 0.60 = min(0.60,1.00) 334

```

The quintiles rating-by-sorting result, shown in Listing 14.8 on the previous page, confirms the previous COPELAND rating-by-ranking result. However, University U5 is here sorted conjointly into the fourth and the fifth quintiles and as such is part of the top rated institutions. A result already convincingly illustrated in the ranked heatmap is shown in Fig. 14.3 on page 192.

Let us conclude this hypothetical rating case study, by saying that we prefer this latter rating-by-sorting approach; perhaps less precise, due the case given, to missing and contradictory performance data; yet, well grounded in a powerful bipolar-valued logic and epistemic framework.

Chapter 15 proposes a series of decision problems suitable for exercises and exam questions.

References

- Bisdorff R (2010) Enumerating chordless circuits in directed graphs. In: ORBEL24-2010, 24th annual conference of the Belgian operational research society (ORBEL aka Sogesci-B.V.W.B.), January 28-29, Liège (BE), Université de Liège (BE), pp 1–12. <http://hdl.handle.net/10993/23926>
- DER SPIEGEL (2004) Studentenbefragung des SPIEGEL Die Methode. <https://www.spiegel.de/lebenundlernen/uni/studentenbefragung-des-spiegel-die-methode-a-329082.html>
- Friedmann J, Hackenbroch V, Hipp D, Klawitter N, Koch J, Lakotta B, Mohr J, Schmitz C, Thimm K, Wüst C (2004) Die Elite von morgen. DER SPIEGEL 48. <https://www.spiegel.de/politik/die-elite-von-morgen-a-afd0507a-0002-0001-0000-000037494731>

AUTHOR QUERY

AQ1. Please check the sentence “Let us conclude this hypothetical ...” for clarity.

Uncorrected Proof

Chapter 15

Exercises

1
2

Contents

15.1	Introduction	199	4
15.2	Who Will Receive the Best Student Award? (§)	199	5
15.3	How to Fairly Rank Movies? (§)	200	6
15.4	What Is Your Best Choice Recommendation? (§§)	202	7
15.5	Planning the Next Holiday Activity (§§)	203	8
15.6	What Is the Best Public Policy? (§§)	205	9
15.7	A Fair Diploma Validation Decision (§§§)	205	10

Abstract In this chapter, we propose a series of decision problems of various difficulties which may serve as exercises and exam questions for an Algorithmic Decision Theory or Multiple-Criteria Decision Analysis course. They cover *selection*, *ranking*, and *rating* decision problems. 11
12
13
14

15.1 Introduction

15

The difficulty of the exercises is marked as follows: *warming up* (§), *home work* (§§), and *research work* (§§§). Solutions should be supported both by computational Python code using the DIGRAPH3 programming resources and by methodological and algorithmic arguments from the *Algorithmic Decision Theory* Lectures (Bisdorff 2020, 2021). 16
17
18
19
20

15.2 Who Will Receive the Best Student Award? (§)

21

Data

22

In Table 15.1 on the next page, you find the actual grades obtained by four students: Ariana (A), Bruce (B), Clare (C), and Daniel (D) in five courses: C1, C2, C3, C4, 23

Table 15.1 Grades obtained by the students

Course	CF	C2	C3	C4	C5	
ECTS	2	3	4	2	4	t2.1
Ariana (A)	11	13	9	15	11	t2.2
Bruce (B)	12	9	13	10	13	t2.3
Clare (C)	8	11	14	12	14	t2.4
Daniel (D)	15	10	12	8	13	t2.5
						t2.6

and C5 weighted by their respective ECTS points.¹

The student grades are measured on an ordinal performance scale from 0 pts (weakest) to 20 pts (highest). Assume that the grading admits a preference discrimination threshold of 1 point. No considerable performance differences are given. The more ECTS points, the more importance a course takes in the curriculum of the students. An award is to be granted to the student showing the *best* result.

Questions

1. Edit a `PerformanceTableau` instance with the data shown above. 31
2. Who would you nominate for the grant? 32
3. Explain and motivate your selection algorithm. 33
4. Assume that the grading may actually admit an indifference discrimination threshold of 1 point and a preference discrimination threshold of 2 points. How stable is your grant recommendation with respect to this preference discrimination power of the grading scale? 34
35
36
37

15.3 How to Fairly Rank Movies? (§)

Data

File `graffiti03.py`² contains a performance tableau inspired by the star-rating of movies that could be seen in the city of Luxembourg during Spring 2003 (source: Graffiti magazine, Luxembourg [February 2003](#)). Its content is shown in Fig. 15.1 on the facing page.

```

1  >>> from perfTabs import PerformanceTableau
2  >>> t = PerformanceTableau('graffiti03')
3  >>> t.showHTMLPerformanceHeatmap(WithActionNames=True, \
4  ...                               pageTitle='Graffiti Star wars', \
5  ...                               rankingRule=None, colorLevels=5,
6  ...                               ndigits=0)

```

¹ ECTS stands for *European Credit Transfer System*. It is a unit used to grade diplomas in all participating countries, https://wwwen.uni.lu/studies/ects_credits.

² The file is provided in the `examples` directory of the DIGRAPH3 resources.

Performance table graffiti03

criteria	ap	as	cf	cn	cs	dr	jh	jt	mk	mr	rr	td	vt
weight	1.00	1.00	1.00	1.00	1.00	1.00	2.00	1.00	1.00	1.00	1.00	1.00	2.00
ah	NA	NA	NA	-1	1	NA	1	1	2	NA	1	3	NA
aw	-1	NA	1	NA	2	NA	NA	-1	1	NA	1	NA	NA
bb	2	1	2	2	3	2	2	2	3	2	3	1	1
dl	-1	-1	-1	NA	-1	1	1	1	NA	1	1	1	-1
gny	2	4	2	4	2	3	3	4	2	4	3	2	3
gs	1	NA	-1	NA	1	1	NA	1	NA	-1	-1	-1	NA
hn	3	NA	3	2	2	NA	2	2	3	2	2	NA	1
la	3	2	3	3	2	2	3	3	3	4	3	NA	3
lor	2	3	3	NA	3	4	3	4	1	2	2	2	2
ma	NA	NA	NA	3	2	3	3	3	3	2	2	3	3
md	1	1	-1	NA	NA	-1	NA	1	-1	NA	-1	1	NA
mi	NA	-1	NA	1	-1	NA	1	2	NA	NA	-1	2	1
sa	NA	NA	NA	NA	2	NA	2	1	3	1	NA	NA	NA
sc	1	NA	1	NA	-1	NA	NA	1	1	NA	1	NA	NA
sha	2	-1	1	1	2	2	-1	2	1	1	1	NA	NA
ss	3	3	3	4	2	3	3	3	3	3	1	3	3
vf	NA	NA	NA	1	NA	NA	1	1	NA	NA	1	1	NA

Fig. 15.1 Star-rating of movies from February 2003

The critic's opinions are expressed on seven ordinal rating levels: -2 (two zeros, 50 *I hate*), -1 (one zero, *I do not like*), 1 (one star, *maybe*), 2 (two stars, *good*), 3 (three 51 stars, *excellent*), 4 (four stars, *not to be missed*), and 5 (five stars, *a master piece*). 52 Notice the many missing data (NA) when a critic had not seen the respective movie. 53 Mind also that the ratings of two movie critics (jh and vt) are given a higher 54 significance weight. 55

Questions

1. The GRAFFITI magazine suggests a best rated movie by computing the average 57 number of stars it received, ignoring the missing data and any significance 58 weights of the critics. What movie gets the highest average? 59
2. By taking into account missing data and varying significance weights, how may 60 one find the best rated movie without computing any average of stars? 61
3. How would one rank these movies so as to at best respect the weighted rating 62 opinions of each movie critic? 63

4. In what ranking position would appear a movie not seen by any critic? Confirm computationally your answer by adding such a fictive, *not at all evaluated*, movie to the given performance tableau instance. 64
5. How robust are the preceding results when the significance weights of the movie critics are considered to be only of ordinal type? 67

15.4 What Is Your Best Choice Recommendation? (§§)

69

Data

A person, who wants to buy a TV set, retains after a first selection eight potential TV models.³ To make their best choice, these eight models were evaluated with respect to three decision objectives of equal importance: 70

1. *Costs* of the set (to be minimised) 74
2. *Picture and Sound* quality of the TV (to be maximised) 75
3. *Maintenance contract* quality of the provider (to be maximised) 76

The *Costs* objective is assessed by the price of the TV set (criterion Pr to be minimised). *Picture* quality (criterion Pq), *Sound* quality (criterion Sq), and *Maintenance contract* quality (criterion Mq) are each one assessed on a four-level qualitative performance scale: -1 (*not good*), 0 (*average*), 1 (*good*), and 2 (*very good*). 77

The actual evaluation data are gathered in Table 15.2 on the next page. 82

The *Price* criterion Pr supports furthermore an indifference threshold of 25.00€ and a preference threshold of 75.00€. No considerable performance differences (veto thresholds) are to be considered. 83

Questions

1. Edit a new *PerformanceTableau* instance with the data shown in Table 15.2 on the facing page, and illustrate its content by best showing objectives, criteria, decision alternatives, and performance table. If needed, write adequate Python code. 87
2. What is the best TV set to recommend? 91
3. Illustrate your best choice recommendation with an adequate *graphviz* drawing. 92
4. Explain and motivate your selection algorithm. 93
5. Assume that the qualitative criteria: *Picture* quality (Pq), *Sound* quality (Sq), and *Maintenance contract* quality (Mq) are all three considered to be equi-significant and that the significance of the *Price* criterion (Pr) equals the significance of these three quality criteria taken together. How stable is your best choice recommendation with respect to thus reviewing the criteria significance weights? 94

³ A similar didactic decision problem is discussed in Vincke (1992, pp.33–35).

Table 15.2 Performance evaluations of the potential TV sets

Criteria	Pr (€)	Pq	Sq	Mq	
Significance	2	1	1	1	t4.1
Model T1	-1300	2	2	0	t4.2
Model T2	-1200	2	2	1	t4.3
Model T3	-1150	2	1	1	t4.4
Model T4	-1000	1	1	-1	t4.5
Model T5	-950	1	1	0	t4.6
Model T6	-950	0	1	-1	t4.7
Model T7	-900	1	0	-1	t4.8
Model T8	-900	0	0	0	t4.9
					t4.10

Table 15.3 The set of potential holiday activities

Identifier	Name	Comment	
ant	<i>Antequerra</i>	An afternoon excursion to Antequerra and surroundings	t7.1
ard	<i>Ardales</i>	An afternoon excursion to Ardales and El Chorro	t7.2
be	<i>Beach</i>	Sun, sea, fun, and more	t7.3
crd	<i>Cordoba</i>	A whole day visit by car to Cordoba	t7.4
dn	<i>Fa niente</i>	Do nothing	t7.5
lw	<i>Long walk</i>	A whole day hiking	t7.6
mal	<i>Malaga</i>	A whole day visit to Malaga	t7.7
sev	<i>Sevilla</i>	A whole day visit to Sevilla	t7.8
sw	<i>Short walk</i>	Less than a half-day hiking	t7.9
			t7.10

15.5 Planning the Next Holiday Activity (§§)

99

Data

100

A family, staying during their holidays in Ronda (Andalusia), is planning the next day's activity (Bisdorff 2008). The alternatives shown in Table 15.3 are considered as potential actions. 101
102
103

The family members agree to measure their preferences with respect to a set of seven performance criteria such as the time to attend the place (to be minimised), the required physical investment, the expected quality of the food, touristic interest, relaxation, sun fun, and more (see Table 15.4 on the following page) 104
105
106
107

The evaluation, agreed with all family members, of the nine alternatives on all the criteria resulted in the performance tableau shown in Table 15.5 on the next page. 108
109

All performances on the qualitative criteria are marked on a same ordinal scale 110 going from 0 (lowest) to 10 (highest). On the quantitative Distance criterion (to be 111 minimised), the required travel time to go to and return from the activity is marked 112 in negative minutes. In order to model only effective preferences, an indifference 113 threshold of 1 point and a preference threshold of 2 points are put on the qualitative 114 performance measures. On the Distance criterion, an indifference threshold of 115 20 minute and a preference threshold of 45 min are considered. Furthermore, a 116

Table 15.4 The set of performance criteria

Identifier	Name	Comment	
cult	<i>Cultural interest</i>	Andalusian heritage	t10.1
dis	<i>Distance</i>	Minutes by car to go to and return from the activity	t10.2
food	<i>Food</i>	Quality of the expected food opportunities	t10.3
sun	<i>Sun, fun, and more</i>	No comment	t10.4
phy	<i>Physical investment</i>	Contribution to physical health care	t10.5
rel	<i>Relaxation</i>	Anti-stress support	t10.6
tour	<i>Tourist attraction</i>	How many stars in the guide?	t10.7

Table 15.5 The performance tableau

Criteria	Weight	ant	ard	be	crd	dn	lw	mal	sev	sw	
cult	1	7	3	0	10	0	0	5	10	0	t13.1
dis	1	-120	-100	-30	-360	0	-90	-240	-240	0	t13.2
phy	1	3	7	0	5	0	10	5	5	5	t13.3
rel	1	1	5	8	3	10	5	3	3	6	t13.4
food	1	8	10	4	8	10	1	8	10	1	t13.5
sun	1	0	3	10	3	1	3	8	5	5	t13.6
tour	1	5	7	3	10	0	8	10	10	5	t13.7

difference of more than two hours to go to and return from the activity's place is 117
considered to raise a veto. 118

The individual criteria each reflect one or the other family member's preferential 119
point of view. Therefore, they are judged equi-significant for the best action to be 120
eventually chosen. 121

Questions

1. Edit a new `PerformanceTableau` instance with the data shown above and 123
illustrate its content. 124
2. How do the criteria express their preferential view point on the set of alternatives? 125
For instance, the Tourist Attraction criterion appears to be in its preferential 126
judgments somehow positively correlated with both the cultural interest and the 127
food criteria. It is also apparent that the distance criterion is somehow negatively 128
correlated to these latter criteria. How to explore and illustrate these intuitions? 129
3. What is the preference modelling effect of the considerable performance differ- 130
ence of 120 minutes assumed for the distance criterion? 131
4. How would you rank the potential holiday activities with the fairest balance of 132
the performance criteria? 133
5. What is the first choice you would recommend to the family members? 134

15.6 What Is the Best Public Policy? (§§)

135

Data Files

- File `perfTab_1.py`⁴ contains a 3-objectives performance tableau with 100 performance records concerning public policies evaluated with respect to an *economic*, a *societal*, and an *environmental* decision objective. 137
138
139
- File `historicalData_1.py` contains a performance tableau of the same kind with 2000 historical performance records. 140
141

Questions

- Illustrate the content of the given `perfTab_1.py` performance tableau by best showing *objectives*, *criteria*, *decision alternatives* (public policies), and *performance evaluations*. If needed, write adequate Python code. 143
144
145
- Construct the corresponding bipolar-valued outranking digraph. How *confident* and/or *robust* are the apparent outranking situations? 146
147
- What are apparently the 5 best-ranked decision alternatives in your decision problem from the different decision objectives point of views and from a global fair compromise view? Justify your ranking approach from a methodological point of view. 148
149
150
151
- How would you rate your 100 public policies into relative deciles classes? 152
- Using the given historical records in file `historicalData_1.py`, how would you rate your 100 public policies into absolute deciles classes? 153
154
- Explain the differences you may observe between the absolute and the previous relative rating results. 155
156
- Select among your 100 potential public policies a shortlist of up to 15 potential best policies, all reaching an absolute performance quantile of at least 66.67%. 157
158
- Based on the previous best policies shortlist (see Question 7), what is your eventual best choice recommendation? Is it perhaps a best choice unopposed by all three objectives? 159
160
161

15.7 A Fair Diploma Validation Decision (§§§)

162

Data

163

Use the `RandomAcademicPerformanceTableau` class from the `randomPerfTab` module for generating realistic random students' performance tableaux concerning a curriculum of nine ECTS weighted courses (Bisdorff 2021). Assume that all the gradings are done on an integer scale from 0 (weakest) to 20 (best). It is known that grading procedures are inevitably somehow imprecise; therefore assume

164

165

166

167

168

⁴ Files `perfTab_1.py` and `historicalData_1.py` are provided in the examples directory of the DIGRAPH3 resources (Bisdorff 2021).

an indifference discrimination threshold of 1 point and a preference discrimination threshold of 2 points. Furthermore, a performance difference of more than 12 points is considerable and will trigger a polarisation situation. To validate eventually their curriculum, the students are required to obtain more or less 10 points in each course.

Questions

1. Design and implement a fair diploma validation decision rule based on the grades obtained in the nine courses.
2. Run simulation tests with random students' performance tableaux for validating your design and implementation.

References

- Bisdorff R (2008) On clustering the criteria in an outranking based decision aid approach. In: Thi HAL, Bouvry P, Pham D (eds) Computation and Optimization in Information Systems and Management Sciences, Springer, CCIS, pp 409–418. <http://hdl.handle.net/10993/23718>
- Bisdorff R (2020) Lectures of the algorithmic decision theory course, University of Luxembourg. <http://hdl.handle.net/10993/37933>
- Bisdorff R (2021) Technical documentation of the Digraph3 collection of Python modules. <https://digraph3.readthedocs.io/en/latest/techDoc.html>
- Graffiti magazine, Luxembourg (February 2003) Star wars
- Vincke P (1992) Multicriteria Decision-Aid. John Wiley & Sons Ltd, Chichester

AUTHOR QUERIES

- AQ1.** Please check the hierarchy of section heading and correct if necessary.
- AQ2.** Please check if the sentence “*Picture* quality (criterion Pq), *Sound* quality (criterion Sq) ...” is fine as edited and amend if required.

Uncorrected Proof

Part IV ₁ Advanced Topics ₂

The fourth part gathers five chapters introducing and discussing further going 3
algorithmic developments. In Chap. 16, KENDALL's ordinal correlation index is 4
consistently extended to bipolar-valued digraphs. Chapter 17 explains the important 5
concept of digraph kernel and illustrates the DIGRAPH3 algorithmic approach to 6
their computation. In Chap. 18, criteria significance weights are considered to be 7
uncertain and become random variates. This idea opens the way to compute a 8
bipolar-valued likelihood of outranking and outranked situations leading to $\alpha\%$ - 9
confident outranking digraphs. In Chap. 19, the criteria significance weights are 10
considered to be only of ordinal type. This working hypothesis induces the need 11
to put into place a specific robustness analysis of the corresponding outranking 12
digraphs. The last chapter, Chap. 20, makes use of the preceding robustness analysis 13
for introducing and discussing ideas, like two-stage elections with multipartisan 14
primary selection or bipolar voting systems, for tempering plurality tyranny effects 15
in social choice problems. 16

Chapter 16

On Measuring the Fitness of a Multiple-Criteria Ranking

1
2
3

Contents

16.1	Listing Movies from Best Star-Rated to Worst	209	5
16.2	KENDALL's Ordinal Correlation Tau Index.....	212	6
16.3	Bipolar-Valued Relational Equivalence	214	7
16.4	Fitness of Ranking Heuristics	217	8
16.5	Illustrating Preference Divergences	219	9
16.6	Exploring the “ <i>better rated</i> ” and the “ <i>as well as rated</i> ” Opinions	220	11

Abstract Starting from a motivating decision problem about how to list, from the best to the worst, a set of movies that are star-rated by journalists and movie critics, the chapter shows that KENDALL's ordinal correlation index tau can be extended to a bipolar-valued relational equivalence measure of bipolar-valued digraphs. This finding gives way, on the one hand, to measure the fitness and fairness of multiple-criteria ranking rules. On the other hand, it provides a tool for illustrating preference divergences between decision objectives and criteria.

16.1 Listing Movies from Best Star-Rated to Worst

19

In a stubborn keeping with a two-valued logic, where every argument can only be true or false, there is no place for efficiently taking into account missing data or logical indeterminateness. These cases are seen as problematic and at best are simply ignored. Worst, in modern data science, missing data get often replaced with *fictive* values, potentially falsifying hence all subsequent computations.

In social choice problems, voting abstentions are, however, frequently observed and represent a social expression that may be significant for revealing non-represented social preferences. And, in marketing studies, interviewees will not always respond to all the submitted questions. Again, such abstentions do sometimes contain nevertheless valid information concerning consumer preferences.

Graffiti Star wars

criteria	AP	AS	CF	CS	DR	FG	GS	JH	JPT	MR	RR	SF	SJ	TD	VT
weight	1.00	1.00	1.00	1.00	1.00	1.00	1.00	3.00	1.00	1.00	1.00	1.00	1.00	1.00	3.00
mv_AL	3	-1	2	-1	NA	NA	3	NA	2	NA	NA	NA	2	NA	2
mv BI	1	-1	1	1	-1	NA	NA	NA	2	NA	2	NA	NA	NA	NA
mv_CM	NA	3	3	2	NA	NA	3	2	3	2	1	2	2	NA	2
mv_DF	NA	4	3	1	2	NA	1	3	2	2	2	3	-1	NA	1
mv_DG	3	2	2	3	3	1	4	3	3	3	NA	NA	-1	NA	3
mv_DI	1	2	NA	1	2	2	NA	1	2	2	NA	1	1	NA	NA
mv_DJ	3	1	3	NA	NA	NA	3	NA	2	2	NA	2	4	3	-1
mv_FC	3	2	2	1	3	NA	1	3	3	3	2	NA	2	1	3
mv_FF	2	3	2	1	2	2	NA	1	2	2	1	2	1	3	1
mv_GG	2	3	3	1	NA	NA	1	-1	NA	-1	2	NA	-1	NA	1
mv_GH	1	NA	1	-1	2	2	1	2	1	3	4	NA	NA	NA	NA
mv_HN	3	1	3	1	3	NA	4	3	2	NA	1	NA	3	3	3
mv_HP	1	NA	3	1	NA	NA	NA	1	1	2	3	NA	1	2	NA
mv_HS	2	4	2	3	2	2	2	3	4	2	3	NA	1	3	NA
mv_JB	3	4	3	NA	3	2	3	2	3	2	NA	2	2	NA	3
mv_MB	NA	2	NA	NA	1	NA	1	2	2	1	2	NA	2	2	NA
mv_NP	NA	1	3	1	NA	3	2	3	3	3	2	3	NA	NA	3
mv_PE	3	4	2	NA	3	1	NA	2	4	2	NA	3	3	NA	3
mv_QS	NA	3	2	NA	4	3	NA	4	4	3	3	4	NA	4	4
mv_RG	2	2	2	2	NA	NA	3	1	2	2	1	NA	NA	3	NA
mv_RR	3	2	NA	1	4	NA	4	3	3	4	3	3	NA	4	2
mv_SM	3	3	2	2	2	2	NA	3	2	2	3	NA	2	2	3
mv_TF	-1	NA	1	1	1	NA	NA	1	2	NA	-1	NA	-1	1	NA
mv_TM	2	1	2	2	NA	NA	2	2	2	3	NA	2	NA	4	NA
mv_TP	2	3	3	1	2	NA	NA	3	2	NA	2	NA	1	NA	2

Fig. 16.1 Star-ratings of movies from September 2007

Such a case is given with a list of star-rated movies that could be seen in town 30 in September 2007.¹ The underlying performance tableau data, stored in a file 31 named `graffiti07.py`,² is shown below with the `showHTMLPerformanceTableau()` 32 method:

```

1  >>> from outrankingDigraphs import\                                34
2  ...                                PerformanceTableau                35
3  >>> gt07 = PerformanceTableau('graffiti07')                      36
4  >>> gt07.showHTMLPerformanceTableau(\                                37
5  ...                                title='Graffiti Star wars',\ 38
6  ...                                ndigits=0)                      39

```

Figure 16.1 shows the star-ratings of 25 movies by 15 journalists and movie 40 critics: 5 stars (*masterpiece*), 4 stars (*must be seen*), 3 stars (*excellent*), 2 stars 41 (*good*), 1 star (*could be seen*), -1 star (*I do not like*), -2 stars (*I hate*), and NA 42

¹ *Graffiti Star wars*, Edition Revue Luxembourg, September 2007, p. 30.

² To be found in the `examples` directory of the DIGRAPH3 resources.

(*not seen*). Notice in the second row the higher significance (3.00) that is granted 43 to two locally renowned movie critics, namely JH and VT. Their opinion counts for 44 three times the opinion of the other critics. With six times a 4 stars (*must be seen*) 45 mark, mv_QS is best-rated, followed by mv_RR with four times a 4 stars mark. 46 Fewest stars obtain movie mv_TF with three times a -1 star (*do not like*) mark and 47 five times a 1 star mark. Notice that many movies, like movie mv_BI, are only rated 48 by some of the critics. 49

To aggregate all the critics' star-ratings, the *Graffiti* magazine computes for 50 each movie a global score—the average weighted number of stars it obtained— 51 just ignoring the *not seen* movies. Listing 16.1 illustrates below how to compute 52 these global scores using the data stored in the gt07 performance tableau. Attribute 53 gt07.actions, respectively, gt07.criteria, contains the description of the 54 25 movies, respectively, the 15 critics. The actual star-ratings are to be found in the 55 gt07.evaluation attribute. 56

Listing 16.1 Computing the average weighted number of stars per movie

```

1 >>> # gt07 = PerformanceTableau('graffiti07')          57
2 >>> globalScores = {}                                58
3 >>> for mv in gt07.actions:                         59
4 >>>     globalScores[mv] = Decimal('0')              60
5 >>>     sumWeights = Decimal('0')                  61
6 >>>     for critic in gt07.criteria:                62
7 >>>         stars = gt07.evaluation[critic][mv]    63
8 >>>         if stars != gt07.NA:                   64
9 >>>             weight = gt07.criteria[critic]['weight'] 65
10 >>>             globalScores[mv] += (stars * weight) 66
11 >>>             sumWeights += weight                67
12 >>>     globalScores[mv] /= sumWeights            68
13 >>> graffitiList = [(globalScores[mv],mv) for mv in globalScores] 69
14 >>> graffitiList.sort(reverse=True)            70
15 >>> for item in graffitiList:                   71
16 >>>     print('%s: %.2f' % (item[1],item[0]))    72
17     mv_QS: 3.60                                     73
18     mv_RR: 2.88                                     74
19     mv_PE: 2.67                                     75
20     mv_JB: 2.62                                     76
21     mv_HN: 2.62                                     77
22     mv_NP: 2.60                                     78
23     mv_HS: 2.60                                     79
24     mv_DG: 2.56                                     80
25     mv_SM: 2.53                                     81
26     mv_FC: 2.41                                     82
27     mv_TP: 2.21                                     83
28     mv_CM: 2.20                                     84
29     mv_TM: 2.17                                     85
30     mv_DF: 1.94                                     86
31     mv_RG: 1.83                                     87
32     mv_MB: 1.73                                     88
33     mv_GH: 1.67                                     89
34     mv_DJ: 1.67                                     90
35     mv_FF: 1.61                                     91
36     mv_AL: 1.60                                     92
37     mv_HP: 1.55                                     93
38     mv_DI: 1.42                                     94
39     mv_GG: 0.71                                     95
40     mv_BI: 0.71                                     96
41     mv_TF: 0.55                                     97

```

The global scores ranking confirms in Lines 17–18 both leading movies—
 mv_QS (3.60) and mv_RR (2.88)—as well as in Line 41 the weakest rated
 one—mv_TF (0.55). Mind however that these global averages, due to the numerous
 missing ratings, are not computed with commensurable denominators; some critics
 do indeed use a more or less extended range of stars. The movies not seen, for
 instance, by critic SJ are favoured, as this critic is severer than others in her rating.
 Dropping the movies that were not rated by all the critics is not possible either,
 as none of the 25 movies was actually rated by all the 15 critics. Providing a fictive
 value for the many not seen situations will as well always somehow falsify the global
 scores. What to do?

A better approach is to rank the movies on the basis of pairwise bipolar-
 valued “*rated at least as well as*” statements. Under this epistemic argumentation
 approach, missing evaluations are naturally treated as opinion abstentions and hence
 do not falsify the logical computations. Such a NETFLOWS ranking from best-
 rated to weakest-rated is provided by the **heatmap** browser view generated with the
`showHTMLPerformanceHeatmap()` method (see Listing 16.2).

Listing 16.2 Showing the movie from best- to worst-rated in a heatmap view

```

1 >>> gt07.showHTMLPerformanceHeatmap (\ 114
2 ...     pageTitle='Ranking the movies', \ 115
3 ...     rankingRule='NetFlows', \ 116
4 ...     Correlations=True, \ 117
5 ...     ndigits=0) \ 118

```

The NETFLOWS ranking shown in Fig. 16.2 on the next page confirms again that
 movie mv_QS, with 6 *must be seen* marks, is correctly first-ranked and the movie
 mv_TF is last-ranked with five *do not like* marks.

It is fair, however, to eventually mention here that the *Graffiti* magazine’s average
 stars ranking rule is actually showing a very similar result. Indeed, average scores
 usually confirm well all evident pairwise comparisons, yet *enforce* comparability
 for all less evident ones. How to judge now the fitness of a given ranking rule?

This is the purpose of the ordinal correlation *tau* indexes shown in Fig. 16.2 on
 the facing page, third row. Computing these ordinal correlation indexes is the subject
 of the next section.

16.2 KENDALL’s Ordinal Correlation Tau Index

M.G. Kendall defined his ordinal correlation τ (*tau*) index for linear orders of
 dimension n as a balancing of the number Co of correctly oriented pairs against
 the number In of incorrectly oriented pairs (Kendall 1938). The total number of
 irreflexive pairs being $n(n - 1)$, in the case of linear orders, $Co + In = n(n - 1)$.
 Hence $\tau = \left(\frac{Co}{n(n-1)} \right) - \left(\frac{In}{n(n-1)} \right)$. In case In is zero, $\tau = +1$ (all pairs
 are equivalently oriented); inversely, in case Co is zero, $\tau = -1$ (all pairs are
 differently oriented).

Ranking the movies

criteria	JH	JPT	AP	DR	MR	VT	GS	CS	SJ	RR	TD	CF	SF	AS	FG
weights	+3.00	+1.00	+1.00	+1.00	+1.00	+3.00	+1.00	+1.00	+1.00	+1.00	+1.00	+1.00	+1.00	+1.00	+1.00
tau(*)	+0.50	+0.43	+0.32	+0.26	+0.25	+0.23	+0.16	+0.14	+0.14	+0.13	+0.11	+0.11	+0.10	+0.08	+0.03
mv_QS	4	4	NA	4	3	4	NA	NA	NA	3	4	2	4	3	3
mv_RR	3	3	3	4	4	2	4	1	NA	3	4	NA	3	2	NA
mv_DG	3	3	3	3	3	3	4	3	-1	NA	NA	2	NA	2	1
mv_NP	3	3	NA	NA	3	3	2	1	NA	2	NA	3	3	1	3
mv_HN	3	2	3	3	NA	3	4	1	3	1	3	3	NA	1	NA
mv_HS	3	4	2	2	2	NA	2	3	1	3	3	2	NA	4	2
mv_SM	3	2	3	2	2	3	NA	2	2	3	2	2	NA	3	2
mv_JB	2	3	3	3	2	3	3	NA	2	NA	NA	3	2	4	2
mv_PE	2	4	3	3	2	3	NA	NA	3	NA	NA	2	3	4	1
mv_FC	3	3	3	3	3	3	1	1	2	2	1	2	NA	2	NA
mv_TP	3	2	2	2	NA	2	NA	1	1	2	NA	3	NA	3	NA
mv_CM	2	3	NA	NA	2	2	3	2	2	1	NA	3	2	3	NA
mv_DF	3	2	NA	2	2	1	1	1	-1	2	NA	3	3	4	NA
mv_TM	2	2	2	NA	3	NA	2	2	NA	NA	4	2	2	1	NA
mv_DJ	NA	2	3	NA	2	-1	3	NA	4	NA	3	3	2	1	NA
mv_AL	NA	2	3	NA	NA	2	3	-1	2	NA	NA	2	NA	-1	NA
mv_RG	1	2	2	NA	2	NA	3	2	NA	1	3	2	NA	2	NA
mv_MB	2	2	NA	1	1	NA	1	NA	2	2	2	NA	NA	2	NA
mv_GH	2	1	1	2	3	NA	1	-1	NA	4	NA	1	NA	NA	2
mv_HP	1	1	1	NA	2	NA	NA	1	1	3	2	3	NA	NA	NA
mv_BI	NA	2	1	-1	NA	NA	NA	1	NA	2	NA	1	NA	-1	NA
mv_DI	1	2	1	2	2	NA	NA	1	1	NA	NA	NA	1	2	2
mv_FF	1	2	2	2	2	1	NA	1	1	1	3	2	2	3	2
mv_GG	-1	NA	2	NA	-1	1	1	1	-1	2	NA	3	NA	3	NA
mv_TF	1	2	-1	1	NA	NA	NA	1	-1	-1	1	1	NA	NA	NA

Color legend:

quantile 20.00% 40.00% 60.00% 80.00% 100.00%

(*) tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation

Outranking model: standard, Ranking rule: NetFlows

Ordinal (Kendall) correlation between global ranking and global outranking relation: **+0.780**

Mean marginal correlation (a) : **+0.234**

Standard marginal correlation deviation (b) : **+0.147**

Ranking fairness (a) - (b) : **+0.087**

Fig. 16.2 Star-ratings of movies ranked with the NETFLOWS rule

Noticing that $\frac{Co}{n(n-1)} = 1 - \frac{In}{n(n-1)}$, and recalling that the bipolar-valued negation is operated by changing the sign of the characteristic value, 137
138

$$\tau = 1 - 2 \frac{In}{n(n-1)} = - \left(2 \frac{In}{n(n-1)} - 1 \right) = 2 \frac{Co}{n(n-1)} - 1. \quad (16.1)$$

KENDALL's original *tau* definition implemented in fact the bipolar-valued negation of the non-equivalence of two linear orders, i.e. the normalised majority margin of equivalently oriented irreflexive pairs. 139
140
141

Let $r1$ and $r2$ be two random crisp relations defined on a same set of 5 alternatives. Computing KENDALL's *tau* index may be done as shown in Listing 16.3. 142
143

Listing 16.3 Computing a relational equivalence digraph

```

1 >>> from randomDigraphs import RandomDigraph 144
2 >>> r1 = RandomDigraph(order=5, Bipolar=True) 145
3 >>> r2 = RandomDigraph(order=5, Bipolar=True) 146
4 >>> from digraphs import EquivalenceDigraph 147
5 >>> eqd = EquivalenceDigraph(r1,r2) 148
6 >>> eqd.showRelationTable(ReflexiveTerms=False) 149
7   * ---- Relation Table ---- 150
8   r(<=>) | 'a1' 'a2' 'a3' 'a4' 'a5' 151
9   -----|----- 152
10  'a1' | - -1.00 1.00 -1.00 1.00 153
11  'a2' | -1.00 - -1.00 1.00 -1.00 154
12  'a3' | -1.00 -1.00 - -1.00 1.00 155
13  'a4' | -1.00 1.00 -1.00 - -1.00 156
14  'a5' | -1.00 1.00 -1.00 1.00 - -1.00 157
15  Valuation domain: [-1.00;1.00] 158
16 >>> eqd.correlation 159
17   {'correlation': -0.1, 'determination': 1.0} 160

```

In the table of the equivalence relation ($r1 \Leftrightarrow r2$) above (see Listing 16.1 on page 211, Lines 10–14), we observe that the normalised majority margin of equivalent versus non-equivalent irreflexive pairs amounts to $(9 - 11)/20 = -0.1$, i.e. the value of KENDALL's *tau* index in this plainly determined crisp case (see Line 17).

What happens now with more or less epistemically determined and even partially indeterminate relations? May we proceed in a similar way?

16.3 Bipolar-Valued Relational Equivalence

Two random bipolar-valued digraphs $d1$ and $d2$ of order five, generated in Listing 16.4, will help exploring this idea.

Listing 16.4 Two random bipolar-valued digraphs

```

1 >>> from randomDigraphs import RandomValuationDigraph 171
2 >>> d1 = RandomValuationDigraph(order=5, seed=1) 172
3 >>> d1.showRelationTable(ReflexiveTerms=False) 173
4   * ---- Relation Table ---- 174
5   r(d1) | 'a1' 'a2' 'a3' 'a4' 'a5' 175
6   -----|----- 176
7   'a1' | - -0.66 0.44 0.94 -0.84 177
8   'a2' | -0.36 - -0.70 0.26 0.94 178
9   'a3' | 0.14 0.20 - 0.66 -0.04 179
10  'a4' | -0.48 -0.76 0.24 - -0.94 180
11  'a5' | -0.02 0.10 0.54 0.94 - 181
12  Valuation domain: [-1.00;1.00] 182
13 >>> d2 = RandomValuationDigraph(order=5, seed=2) 183
14 >>> d2.showRelationTable(ReflexiveTerms=False) 184
15   * ---- Relation Table ---- 185

```

16	<code>r(d2)</code>	'a1'	'a2'	'a3'	'a4'	'a5'	186
17	-----	-----	-----	-----	-----	-----	187
18	'a1'	-	-0.86	-0.78	-0.80	-0.08	188
19	'a2'	-0.58	-	0.88	0.70	-0.22	189
20	'a3'	-0.36	0.54	-	-0.46	0.54	190
21	'a4'	-0.92	0.48	0.74	-	-0.60	191
22	'a5'	0.10	0.62	0.00	0.84	-	192
23	Valuation domain: [-1.00;1.00]						193

In the generated random digraphs d_1 and d_2 , 9 pairs, like (a_1, a_2) or (a_3, a_2) , for instance, appear equivalently oriented (see Lines 7 and 18 or Lines 9 and 20). The `EquivalenceDigraph` class computes this bipolar-valued relational equivalence between digraphs d_1 and d_2 (see Listing 16.5). 194
195
196
197

Listing 16.5 Bipolar-valued equivalence digraph

1	<code>>>> from digraphs import EquivalenceDigraph</code>	198
2	<code>>>> eqd = EquivalenceDigraph(d1,d2)</code>	199
3	<code>>>> eqd.showRelationTable(ReflexiveTerms=False)</code>	200
4	* ----- Relation Table -----	201
5	<code>r(<=>)</code>	202
6	'a1'	202
7	'a2'	203
8	'a3'	204
9	'a4'	205
10	'a5'	206
11	Valuation domain: [-1.00;1.00]	207
12		208
		209

In our bipolar-valued epistemic logic, logical disjunctions and conjunctions are implemented as `max` and, respectively, `min` operations. Notice also that the logical equivalence $(d_1 \Leftrightarrow d_2)$ corresponds to a double implication $(d_1 \Rightarrow d_2) \wedge (d_2 \Rightarrow d_1)$ and that the implication $(d_1 \Rightarrow d_2)$ is logically equivalent to the disjunction $(\neg d_1 \vee d_2)$. 210
211
212
213
214

When $r(x \, d_1 \, y)$ and $r(x \, d_2 \, y)$ denote the bipolar-valued characteristic values of relation d_1 and, respectively, d_2 , we may hence compute as follows a majority margin $M(d_1 \Leftrightarrow d_2)$ between equivalently and not equivalently oriented irreflexive pairs (x, y) : 215
216
217
218

$$M(d_1 \Leftrightarrow d_2) = \sum_{(x \neq y)} \left[\min \left(\max(-r(x \, d_1 \, y), r(x \, d_2 \, y)), \max(-r(x \, d_2 \, y), r(x \, d_1 \, y)) \right) \right]. \quad (16.2)$$

$M(d_1 \Leftrightarrow d_2)$ is thus given by the sum of the non-reflexive terms of the relation table of `eqd`, the relational equivalence digraph computed above (see Listing 16.5). 219
220
221
222
223 In the crisp case, $M(d_1 \Leftrightarrow d_2)$ is normalised with the maximum number of possible irreflexive pairs, namely $n(n - 1)$. In the extended r -valued case, the maximal possible equivalence majority margin M corresponds to the sum D of the

conjoint determinations of $(x \text{ d1 } y)$ and $(x \text{ d2 } y)$ (see Bisdorff 2012):

224

$$D = \sum_{x \neq y} \min \left[\text{abs}(r(x \text{ d1 } y)), \text{abs}(r(x \text{ d2 } y)) \right], \quad (16.3)$$

and we obtain hence in the general r -valued case:

225

$$\tau(\text{d1}, \text{d2}) = \frac{M(\text{d1} \Leftrightarrow \text{d2})}{D}. \quad (16.4)$$

$\tau(\text{d1}, \text{d2})$ corresponds so to the classical ordinal correlation index, yet restricted to the conjointly determined parts of the given digraphs d1 and d2 . In the limit case of two crisp linear orders, D equals $n(n - 1)$, i.e. the number of irreflexive pairs, and we recover KENDALL's original *tau* index definition.

226

227

228

229

It is worthwhile noticing that the ordinal correlation index $\tau(\text{d1}, \text{d2})$ one obtains above corresponds in fact to the ratio of:

230

231

- $r(\text{d1} \Leftrightarrow \text{d2}) = \frac{M(\text{d1} \Leftrightarrow \text{d2})}{n(n-1)}$:
the normalised majority margin of the pairwise *relational* equivalence statements, also called *valued ordinal correlation*.
232
233
234
- $d = \frac{D}{n(n-1)}$:
the normalised epistemic determination of the corresponding pairwise relational equivalence statements, in fact the *determinateness* of the relational equivalence digraph.
235
236
237
238

The epistemic determination effect is thus successfully *out-factored* from the ordinal correlation effect. With completely determined relations, $\tau(\text{d1}, \text{d2}) = r(\text{d1} \Leftrightarrow \text{d2})$. The ordinal correlation with a completely indeterminate digraph, i.e. when $D = 0$, is set by convention to the indeterminate correlation value 0.0. With uniformly chosen random r -valued digraphs, the expected τ index is 0.0, denoting in fact an indeterminate relational equivalence. The corresponding expected normalised determination d is about 0.333 (see Bisdorff (2012)).

239

240

241

242

243

244

245

246

247

We verify Eq. 16.4 below with the help of the equivalence digraph `eqd` computed in Listing 16.5 on the preceding page.

Listing 16.6 Computing the ordinal correlation index from the equivalence digraph

```

1 >>> # eqd = EquivalenceDigraph(d1,d2)                                248
2 >>> M = Decimal('0'); D = Decimal('0')                                249
3 >>> n2 = eqd.order*(eqd.order - 1)                                    250
4 >>> for x in eqd.actions:                                              251
5     ...     for y in eqd.actions:                                         252
6     ...         M += eqd.relation[x][y]                                    253
7     ...         D += abs(eqd.relation[x][y])                                254
8 >>> print('r(rd1<=>rd2) = %+.3f, d = %.3f, tau = %+.3f' %\      255
9     ...     (M/n2,D/n2,M/D))                                         256
10    r(rd1<=>rd2) = +0.026, d = 0.356, tau = +0.073                  257

```

The DIGRAPH3 resources directly provide for the preceding computations the `computeOrdinalCorrelation()` method which renders a dictionary with a correlation (τ) and a determination (d) attribute. $r(d1 \Leftrightarrow d2)$ is recovered by multiplying τ with d (see Listing 16.7, Line 4). 258
259
260
261

Listing 16.7 Computing the valued ordinal correlation index

```

1 >>> corrd1d2 = d1.computeOrdinalCorrelation(d2) 262
2 >>> tau = corrd1d2['correlation'] 263
3 >>> d = corrd1d2['determination'] 264
4 >>> r = tau * d 265
5 >>> print('tau(d1,d2) = %+.3f, d = %.3f,\n 266
6 ...           r(d1<=>d2) = %+.3f' % (tau, d, r)) 267
7   tau(d1,d2) = +0.073, d = 0.356, r(d1<=>d2) = +0.026 268

```

The DIGRAPH3 resources provide for convenience a direct `showCorrelation()` method: 269
270

```

1 >>> d1.showCorrelation(\ 271
2 ...           d1.computeOrdinalCorrelation(d2) ) 272
3 Correlation indexes: 273
4   Extended Kendall tau      : +0.073 274
5   Epistemic determination   : 0.356 275
6   Bipolar-valued equivalence : +0.026 276

```

We are now ready for assessing the quality of the NETFLOWS ranking of the movies shown in the heatmap view of Fig. 16.2 on page 213. 277
278

16.4 Fitness of Ranking Heuristics

279

The NETFLOWS ranking of the movies shown in the heatmap view in Fig. 16.2 280
on page 213 is based on the bipolar-valued outranking digraph modelling the 281
pairwise global “*rated at least as well as*” relation among the 25 movies from the 282
performance tableau instance `gt07`. 283

Listing 16.8 The bipolar-valued outranking digraph of the star-rated movies

```

1 >>> bod = BipolarOutrankingDigraph(gt07) 284
2 *----- Object instance description -----* 285
3   Instance class      : BipolarOutrankingDigraph 286
4   Instance name       : rel_graffitiPerfTab.xml 287
5   Actions             : 25 288
6   Criteria            : 15 289
7   Size                : 390 290
8   Determinateness     : 65% 291
9   Valuation domain   : {'min': Decimal('-1.0'), 292
10                      'med': Decimal('0.0'), 293
11                      'max': Decimal('1.0'),} 294
12 >>> g.computeCoSize() 295
13   188 296

```

Listing 16.8 on the preceding page reveals that the outranking digraph `bod` contains 390 positively validated (Line 7), 188 positively invalidated (Line 13), and 22 indeterminate outranking situations from the potential $25 \times 24 = 600$ irreflexive movie pairs.

Listing 16.9 illustrates with the `NetFlowsOrder` class from the `linearOrders` module how to compute the global NETFLOWS ranking `nf`, as shown in the ordered heatmap of Fig. 16.2 on page 213 and the bipolar-valued relational equivalence of the `nf` ranking with each one of the individual critic's star-ratings.

AQ1

Listing 16.9 Computing marginal criterion correlations with global NETFLOWS ranking

```

1  >>> from linearOrders import NetFlowsOrder          305
2  >>> nf = NetFlowsOrder(bod)                         306
3  >>> nf.netFlowsRanking                           307
4  ['mv_QS', 'mv_RR', 'mv_DG', 'mv_NP', 'mv_HN', 'mv_HS', 'mv_SM', 308
5  'mv_JB', 'mv_PE', 'mv_FC', 'mv_TP', 'mv_CM', 'mv_DF', 'mv_TM', 309
6  'mv_DJ', 'mv_AL', 'mv_RG', 'mv_MB', 'mv_GH', 'mv_HP', 'mv_BI', 310
7  'mv_DI', 'mv_FF', 'mv_GG', 'mv_TF']           311
8  >>> for i,item in enumerate():                   312
9  ...         bod.computeMarginalVersusGlobalRankingCorrelations(\ 313
10 ...             nf.netFlowsRanking,ValuedCorrelation=True) :\ 314
11 ...     print('r(%s<=>nf) = %+.3f' % (item[1],item[0]) ) 315
12
13  r(JH<=>nf) = +0.500                         316
14  r(JPT<=>nf) = +0.430                         317
15  r(AP<=>nf) = +0.323                         318
16  r(DR<=>nf) = +0.263                         319
17  r(MR<=>nf) = +0.247                         320
18  r(VT<=>nf) = +0.227                         321
19  r(GS<=>nf) = +0.160                         322
20  r(CS<=>nf) = +0.140                         323
21  r(SJ<=>nf) = +0.137                         324
22  r(RR<=>nf) = +0.133                         325
23  r(TD<=>nf) = +0.110                         326
24  r(CF<=>nf) = +0.110                         327
25  r(SF<=>nf) = +0.103                         328
26  r(AS<=>nf) = +0.080                         329
27  r(FG<=>nf) = +0.027                         330

```

In Listing 16.9 (see Lines 13–27), we obtain the relational equivalence characteristic values shown in the third row of the ranked heatmap (see Fig. 16.2 on page 213). The global NETFLOWS ranking `nf` represents obviously a rather balanced compromise with respect to each movie critic's star-ratings, as there appears no negative correlation with anyone of them. The ranking `nf` apparently takes also correctly into account that the journalist *JH*, a locally renowned movie critic, shows a higher significance weight (see Line 13).

The ordinal correlation between the global NETFLOWS ranking and the outranking digraph `bod` may be furthermore computed as illustrated in Listing 16.10:

Listing 16.10 Computing the ordinal correlation between NETFLOWS and global outranking digraph

```

1 >>> corrgnf = bod.computeOrdinalCorrelation (nf) 341
2 >>> bod.showCorrelation (corrgnf) 342
3   Correlation indexes: 343
4     Extended Kendall tau : +0.780 344
5     Epistemic determination : 0.300 345
6     Bipolar-valued equivalence : +0.234 346

```

One may notice in Line 4 above that the ordinal correlation $\tau(\text{bod}, \text{nf})$ index 347 between the NETFLOWS ranking nf and the determined part of the outranking 348 digraph bod is quite high (+0.780). Due to the rather high number of missing 349 data, the r -valued relational equivalence between the nf and the bod digraph, 350 with a characteristic value of only +0.234, may be misleading. Yet, +0.234 still 351 corresponds to a 62% majority support of the movie critics' star-ratings. 352

It would be interesting to compare similarly the correlations one may obtain with 353 other global ranking heuristics, like the COPELAND ranking rule. 354

16.5 Illustrating Preference Divergences

355

The bipolar-valued relational equivalence indexes give us, via the `showCriteriaCorrelationTable(ValuedCorrelation=True)` method, a further 356 measure for studying how *divergent* may appear the rating opinions expressed by 357 the movie critics. 358

It is remarkable to notice in Fig. 16.3 that, due to the quite numerous missing 360 data, all pairwise valued ordinal correlation indexes $r(x \Leftrightarrow y)$ appear to be of low 361 value, except the diagonal ones. These reflexive indexes $r(x \Leftrightarrow x)$ would trivially 362 all amount to +1.0 in a plainly determined case. Here they indicate a reflexive 363 normalised determination score d , i.e. the proportion of pairs of movies each critic 364

```

1 >>> g = BipolarOutrankingDigraph(t, Normalized=True)
2 >>> g.showCriteriaCorrelationTable(ValuedCorrelation=True)
3   Criteria valued ordinal correlation index
4   AP   AS   CF   CS   DR   FG   GS   JH   JPT   MR   RR   SF   SJ   TD   VT
5   AP   +0.63 +0.04 +0.19 +0.09 +0.22 -0.01 +0.11 +0.23 +0.25 +0.88 +0.02 +0.04 +0.19 +0.04 +0.12
6   AS   +0.77 +0.12 +0.12 +0.04 -0.02 -0.06 +0.02 +0.24 -0.08 +0.07 +0.04 -0.07 -0.01 +0.02
7   CF   +0.77 +0.07 +0.11 +0.03 +0.05 +0.07 +0.10 -0.03 +0.01 +0.00 +0.06 +0.03 -0.04
8   CS   +0.63 +0.04 -0.02 +0.07 +0.13 +0.25 +0.01 +0.03 +0.00 +0.02 +0.03 +0.07
9   DR   +0.45 +0.03 +0.07 +0.17 +0.23 +0.16 +0.04 +0.03 +0.10 +0.07 +0.10
10  FG   +0.15 -0.01 +0.04 +0.01 +0.06 -0.00 +0.02 +0.01 +0.01 +0.01 +0.01 +0.02
11  GS   +0.40 +0.07 +0.07 +0.09 -0.02 +0.00 +0.06 +0.04 +0.04
12  JH   +0.77 +0.28 +0.26 +0.15 +0.12 +0.18 +0.05 +0.14
13  JPT  +0.92 +0.15 +0.06 +0.09 +0.08 +0.08 +0.08 +0.17
14  MR   +0.63 +0.10 +0.08 +0.03 +0.09 +0.10 +0.05 +0.10
15  RR   +0.51 +0.04 +0.01 +0.05 +0.05 +0.05 +0.05 +0.05
16  SF   +0.18 +0.01 +0.02 +0.05
17  SJ   +0.51 +0.03 +0.07
18  TD   +0.26 +0.00
19
20  VT   +0.40

```

Fig. 16.3 Pairwise valued correlation of the movie critics

did evaluate. Critic JPT (the editor of the Graffiti magazine), for instance, evaluated 365 all but one ($d = 24 \times 23 / 600 = 0.92$), whereas critic FG evaluated only 10 movies 366 among the 25 in discussion ($d = 10 \times 9 / 600 = 0.15$). 367

To get a picture of the actual divergence of rating opinions concerning jointly 368 seen pairs of movies, we may develop a Principal Component Analysis of the 369 corresponding τ correlation matrix.³ The 3D plot of the first 3 principal axes is 370 shown in Fig. 16.5 on page 213. 371

```
1 >>> bod.export3DplotOfCriteriaCorrelation (\ 372
2 ...           ValuedCorrelation=False) 373
```

The first 3 principal axes support together about 70% of the total inertia. Most 374 eccentric and opposed in their respective rating opinions appear, on the first principal 375 axis with 27.2% inertia, the conservative daily press against labour and public press. 376 On the second principal axis with 23.7.7% inertia, it is the people press versus the 377 cultural critical press. And, on the third axis with still 19.3% inertia, the written 378 media appear most opposed to the radio media (Fig. 16.5 on the facing page). 379

AQ2

16.6 Exploring the “*better rated*” and the “*as well as rated*” 380 Opinions 381

In order to furthermore study the quality of a ranking result, it may be interesting to 382 have a separate view on the asymmetric and symmetric parts of the “*at least as well 383 rated as*” opinions (see Sect. 2.3). 384

Let us first inspect the pairwise asymmetric part, namely the “*better rated than*” 385 and “*less well rated than*” opinions of the movie critics. 386

```
1 >>> from digraphs import AsymmetricPartialDigraph 387
2 >>> ag = AsymmetricPartialDigraph(bod) 388
3 >>> ag.showHTMLRelationTable (\ 389
4 ...           actionsList=g.computeNetFlowsRanking(), ndigits=0) 390
```

We notice in Fig. 16.4 on the next page that the NETFLOWS ranking rule inverts 391 in fact just three “*less well rated than*” opinions and four “*better rated than*” 392 ones. A similar look in Fig. 16.6 on page 222 at the symmetric part—the pairwise 393 “*as well rated as*” opinions—suggests a preordered preference structure in several 394 equivalently rated classes. 395

```
1 >>> from digraphs import SymmetricPartialDigraph 396
2 >>> sg = SymmetricPartialDigraph(bod) 397
3 >>> sg.showHTMLRelationTable (\ 398
4 ...           actionsList=g.computeNetFlowsRanking(), \ 399
5 ...           ndigits=0) 400
```

³ The 3D PCA plot method requires a running R statistics software (<https://www.r-project.org/>) installation and the Calmat matrix calculator (see the calmat directory in the DIGRAPH3 resources).

Valued Adjacency Matrix

hrs_X_h	hrs_Q_h	hrs_R_h	hrs_D_h	hrs_N_h	hrs_H_h	hrs_S_h	hrs_M_h	hrs_B_h	hrs_C_h	hrs_F_h	hrs_T_h	hrs_M_h	hrs_C_h	hrs_D_h	hrs_T_h	hrs_M_h	hrs_D_h	hrs_A_h	hrs_R_h	hrs_M_h	hrs_H_h	hrs_B_h	hrs_C_h	hrs_F_h	hrs_G_h	hrs_T_h
hrs_QS	-	11	12	11	10	9	14	9	11	13	9	10	9	9	6	9	6	7	6	5	9	15	8			
hrs_RR	-5	-	0	0	0	0	0	0	5	0	0	0	9	12	9	10	6	9	10	8	9	6	10	13	10	
hrs_DG	-6	0	-	0	0	0	0	0	0	7	7	10	10	11	6	8	10	7	9	5	6	7	9	9		
hrs_NP	-7	0	-	0	0	0	0	0	0	8	0	0	8	0	7	6	4	6	6	5	7	12	10			
hrs_HN	-10	0	0	0	-	0	0	0	0	0	0	7	0	6	0	10	7	6	8	8	5	7	13	9	1	
hrs_HS	-3	0	0	0	0	-	0	0	0	0	0	5	0	7	0	0	10	9	6	9	7	11	14	9		
hrs_SM	-8	0	0	0	0	0	-	0	0	0	12	9	0	6	0	9	10	7	9	7	11	15	11	1		
hrs_JB	-9	-1	0	0	0	0	0	0	-	0	0	0	0	6	8	0	9	9	8	8	5	11	15	12		
hrs_PE	-7	0	0	0	0	0	0	0	-	0	4	10	0	0	0	6	8	8	0	6	5	9	13	9		
hrs_FC	-9	0	0	0	0	0	0	0	0	-	10	6	0	5	3	8	0	9	9	5	7	10	12	10		
hrs_TP	-7	0	-1	0	0	0	0	-4	4	0	-	0	0	6	4	0	7	6	7	7	9	14	12	1		
hrs_CM	-6	-1	-5	-4	-3	-3	-1	0	2	-2	0	-	0	0	8	0	10	0	0	7	3	9	14	11		
hrs_DF	-9	-2	-2	0	0	0	0	0	0	0	0	-	4	6	-1	6	8	6	0	6	8	9	0			
hrs_TM	-3	-7	-6	-2	-2	-1	0	0	0	-1	-2	0	-	2	-	0	0	0	9	7	5	7	9	5		
hrs_DJ	-7	-6	-4	-3	-3	0	4	0	0	-2	-1	0	0	0	0	-	0	0	4	3	6	4	4	0	0	
hrs_AL	-4	0	-4	-6	-6	0	-1	-3	-4	-2	0	0	3	0	0	-	0	0	5	1	0	0	5	3		
hrs_RG	-7	-7	-6	-4	-5	-1	-2	-1	-5	-4	-2	-3	-2	-2	0	0	-	0	0	0	4	0	0	4		
hrs_MB	-9	-6	-5	-4	-4	-1	-9	-4	-1	-2	-1	-2	0	-2	-2	0	-	0	4	4	4	2	6			
hrs_GH	-5	-6	-7	-6	-6	-4	-3	0	0	-5	-5	0	-2	-1	-3	0	0	-	5	0	0	0	3			
hrs_HP	-4	-5	-5	-2	-4	-3	-3	-4	-4	-3	-1	-3	-6	-7	-2	-1	-2	-1	-	0	0	0	0	7		
hrs_BH	-5	-4	-6	-1	-7	-7	-5	-5	-5	-3	-1	-3	-6	-3	-2	-6	-2	0	0	0	-	0	-1	-1		
hrs_DV	-9	-6	-5	-5	-3	-3	-3	-7	-7	-6	-1	-3	-6	-5	-2	-6	-6	0	0	0	0	0	0	4		
hrs_FT	-11	-11	-7	-7	-10	-5	0	-3	-9	-9	-6	-2	-6	-6	-1	0	-1	0	0	0	5	0	0	0	1	
hrs_GG	-6	-6	-7	-4	-5	-7	-9	-10	-9	-4	-2	-7	-3	-3	-3	-2	-2	-1	-3	-5	4	2	0	-		
hrs_FF	-6	-7	-7	-5	-7	-11	-9	-8	-8	-7	-6	-8	-3	-6	-3	-1	-1	-4	-3	-3	0	0	-1	0	-	

Valuation domain: [-19.00, +19.00]

Fig. 16.4 Asymmetric part of the “*at least as well star-rated as*” statements

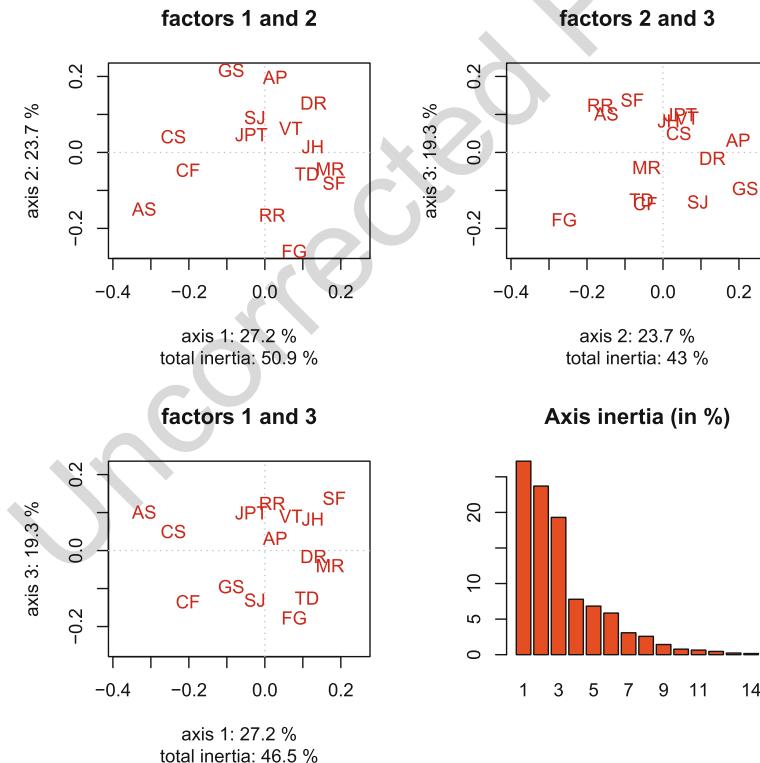


Fig. 16.5 3D PCA plot of the criteria ordinal correlation matrix

Valued Adjacency Matrix

mv_SS	mv_QS	mv_JR	mv_DG	mv_NP	mv_HN	mv_HS	mv_ID	mv_PE	mv_FC	mv_SM	mv_CM	mv_DF	mv_TM	mv_DL	mv_AL	mv_RG	mv_MB	mv_GH	mv_HP	mv_BI	mv_DI	mv_FF	mv_GG	mv_TF
mv_QS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
mv_JR	0	2	5	7	8	6	4	0	2	9	10	0	0	0	0	0	0	0	0	0	0	0	0	0
mv_DG	0	9	4	5	9	10	3	9	7	8	13	0	0	0	0	0	0	0	0	0	0	0	0	0
mv_NP	0	5	7	1	10	3	7	9	8	11	9	0	12	0	0	0	0	0	0	0	0	0	0	0
mv_HN	0	4	8	6	1	5	9	9	8	10	10	0	10	0	0	0	0	0	0	0	0	0	0	0
mv_HS	0	2	5	3	3	10	2	5	6	9	0	10	0	0	1	0	0	0	0	0	0	14	0	0
mv_SM	0	6	7	5	7	6	4	6	6	10	12	0	10	0	4	0	0	0	0	0	0	0	0	0
mv_CM	0	0	5	1	3	4	8	1	9	0	0	13	6	8	0	0	0	0	0	0	0	0	0	0
mv_DF	0	2	6	0	4	3	6	11	0	5	0	0	5	7	0	0	0	0	5	0	0	0	0	0
mv_FC	0	5	11	9	8	2	8	7	~	10	0	11	0	0	0	0	0	0	0	0	0	0	0	0
mv_TP	0	4	0	3	2	3	0	0	0	0	~	6	11	0	4	5	0	0	0	0	0	0	0	0
mv_CM	0	0	0	0	0	0	0	5	0	0	4	~	3	8	0	9	0	7	5	0	0	0	0	0
mv_DL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	6	0	13	13	0	0
mv_TM	0	0	0	0	0	0	0	0	1	0	0	2	0	~	2	2	7	6	0	0	0	0	0	0
mv_DJ	0	0	0	0	0	2	4	2	1	0	0	0	0	4	~	3	5	0	0	0	0	4	3	1
mv_AL	0	0	0	0	0	1	0	0	0	0	3	1	0	2	3	~	2	2	0	0	3	1	0	0
mv_RG	0	0	0	0	0	0	0	0	0	0	0	0	1	3	4	~	1	0	6	0	8	9	0	0
mv_MB	0	0	0	0	0	0	0	0	0	0	1	0	2	0	2	3	~	2	0	4	4	2	0	0
mv_GH	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	6	~	0	2	5	3	0
mv_HP	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	3	0	5	0	0
mv_BI	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	1	~	1	0	0	4
mv_DI	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0	1	8	5	~	6	0	0
mv_FF	0	0	0	0	0	0	0	0	0	0	0	0	1	0	5	0	9	0	1	7	0	12	~	9
mv_GG	0	0	0	0	0	0	0	0	0	0	0	0	3	0	1	0	0	0	0	0	0	3	~	2
mv_TF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	0	2	0	0

Validation domain [-19.00, +19.00]

Fig. 16.6 Symmetric part of the “at least as well star-rated as” statements

Such a kind of preordering of the movies can, for instance, be computed 401 in Listing 16.11 with the `RankingByChoosingDigraph` class from the 402 `transitiveDigraphs` module, where we iteratively extract the best remaining 403 choices—initial kernels—and the worst remaining choices—terminal kernels (see 404 Chap. 17). 405

Listing 16.11 Bipolar ranking-by-choosing the movies

```

1 >>> from transitiveDigraphs import RankingByChoosingDigraph 406
2 >>> rbc = RankingByChoosingDigraph(bod) 407
3 Threading ... # if multiple processing cores are detected 408
4 Exiting computing threads 409
5 >>> rbc.showRankingByChoosing() 410
6 Ranking by Choosing and Rejecting 411
7 1st Best Choice ['mv_QS'] 412
8 2nd Best Choice ['mv_DG', 'mv_FC', 'mv_HN', 'mv_HS', 'mv_NP', 413
9 'mv_PE', 'mv_RR', 'mv_SM'] 414
10 3rd Best Choice ['mv_CM', 'mv_JB', 'mv_TM'] 415
11 4th Best Choice ['mv_AL', 'mv_TP'] 416
12 4th Worst Choice ['mv_AL', 'mv_TP'] 417
13 3rd Worst Choice ['mv_GH', 'mv_MB', 'mv_RG'] 418
14 2nd Worst Choice ['mv_DF', 'mv_DJ', 'mv_FF', 'mv_GG'] 419
15 1st Worst Choice ['mv_BI', 'mv_DI', 'mv_HP', 'mv_TF'] 420

```

In Chap. 17, we thoroughly discuss the computation of such kernels in bipolar- 421 valued digraphs. 422

References

423

- Bisdorff R (2012) On measuring and testing the ordinal correlation between bipolar outranking relations. In: Mousseau V, Pirlot M (eds) DAP'2012 From Multiple Criteria Decision Aid to Preference Learning, University of Mons (Belgium), pp 91–100. <http://hdl.handle.net/10993/23909> 424
Kendall MG (1938) A new measure of rank correlation. *Biometrika* 30:81–93 425
426
427
428

Uncorrected Proof

AUTHOR QUERIES

- AQ1.** Please check if the sentence “Listing 16.9 illustrates with the ...” is fine as edited and amend if required.
- AQ2.** Missing citation for “Fig. 16.5” was inserted here. Please check if appropriate. Otherwise, please provide citation for “Fig. 16.5”. Note that the order of main citations of figures in the text must be sequential.

Uncorrected Proof

Chapter 17

On Computing Digraph Kernels

1

2

Contents

17.1	What Is a Graph Kernel?.....	225	4
17.2	Initial and Terminal Kernels	228	5
17.3	Kernels in Lateralized Digraphs	232	6
17.4	Computing First and Last Choice Recommendations	236	7
17.5	Tractability of Kernel Computation	239	8
17.6	Solving Kernel Equation Systems	242	9

Abstract We illustrate in this chapter, first, the concept of graph kernel, i.e. maximal independent set of vertices. In non-symmetric digraphs, the kernel concept becomes richer and separates into initial and terminal kernels. In, furthermore, lateralised outranking digraphs, initial and terminal kernels become separate and may deliver suitable first and, respectively, last choice recommendations. After commenting the tractability of kernel computations, we close the chapter with the solving of bipolar-valued kernel equation systems.

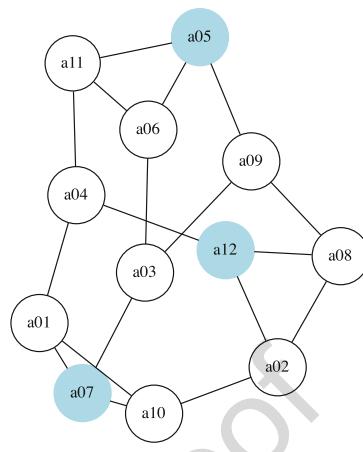
17.1 What Is a Graph Kernel?

17

We call *choice* in a graph, respectively, a digraph, a subset of its vertices, respectively, of its nodes or actions. A choice Y is called *internally stable* or *independent* when there exist no links—(edges) or relations (arcs)—between its members. Furthermore, a choice Y is called *externally stable* when for each vertex, node or action x not in Y , there exists at least a member y of Y such that x is linked or related to y . Now, an internally and externally stable choice is called a *kernel*.

A first trivial example is immediately given by the maximal independent vertices sets (MISs) of the n -cycle graph (see Fig. 21.5 on page 317). Indeed, each MIS in the n -cycle graph is by definition independent, i.e. *internally stable*, and each non-selected vertex in the n -cycle graph is in relation with either one or even two members of the MIS.

Fig. 17.1 A random MIS coloured in the random 3-regular graph `r3g12`. All non-blue vertices are covered by a blue vertex



Digraph3 (graphviz), R. Bisdorff, 2019

In all graphs or symmetric digraphs, the *maximality condition* imposed on the internal stability is equivalent to the *external stability* condition. Indeed, if there would exist a vertex or node not related to any of the elements of a choice, we may safely add this vertex or node to the given choice without violating its internal stability. All kernels must hence be maximal independent choices. In fact, in a topological sense, they correspond to maximal *holes* in the given graph.

Figure 17.1 illustrates this coincidence between MISs and kernels in graphs and symmetric digraphs with a random 3-regular graph instance generated in Listing 17.1. A random MIS in this graph may be computed by using the `MISModel` class (see Line 5 below).

Listing 17.1 Generating a random 3-regular graph of order 12

```

1  >>> from graphs import RandomRegularGraph
2  >>> r3g12 = RandomRegularGraph(order=12, \
3  ...                                     degree=3, seed=4)
4  >>> from graphs import MISModel
5  >>> mg = MISModel(r3g12)
6  Iteration: 1
7  Running a Gibbs Sampler for 660 step !
8  {'a05', 'a07', 'a12'} is maximal !
9  >>> mg.exportGraphViz('random3RegularGraph-mis')
10 *---- exporting a dot file for GraphViz tools ---
11 Exporting to random3RegularGraph-mis.dot
12 fdp -Tpng random3RegularGraph-mis.dot \
13           -o random3RegularGraph-mis.png

```

It is easily verified in Fig. 17.1 that the computed MIS renders indeed a valid kernel of the given graph. The complete set of kernels of this 3-regular graph instance coincides hence with the set of its MISs.

Listing 17.2 Printing out all maximal independent sets of the random 3-regular graph

```

1 >>> g.showMIS()                                55
2     *--- Maximal Independent Sets ---*          56
3     ['a05', 'a07', 'a12']                      57
4     ['a01', 'a06', 'a08']                      58
5     ['a07', 'a08', 'a11']                      59
6     ['a01', 'a09', 'a11', 'a12']                60
7     ['a01', 'a02', 'a09', 'a11']                61
8     ['a01', 'a06', 'a09', 'a12']                62
9     ['a01', 'a02', 'a06', 'a09']                63
10    ['a07', 'a09', 'a11', 'a12']                64
11    ['a02', 'a07', 'a09', 'a11']                65
12    ['a06', 'a07', 'a09', 'a12']                66
13    ['a04', 'a06', 'a07', 'a08']                67
14    ['a02', 'a04', 'a05', 'a07']                68
15    ['a04', 'a05', 'a07', 'a08']                69
16    ['a09', 'a10', 'a11', 'a12']                70
17    ['a06', 'a09', 'a10', 'a12']                71
18    ['a04', 'a06', 'a08', 'a10']                72
19    ['a04', 'a06', 'a09', 'a10']                73
20    ['a01', 'a03', 'a11', 'a12']                74
21    ['a01', 'a02', 'a03', 'a11']                75
22    ['a01', 'a03', 'a08', 'a11']                76
23    ['a01', 'a03', 'a05', 'a12']                77
24    ['a01', 'a02', 'a03', 'a05']                78
25    ['a01', 'a03', 'a05', 'a08']                79
26    ['a02', 'a03', 'a04', 'a05']                80
27    ['a03', 'a10', 'a11', 'a12']                81
28    ['a03', 'a08', 'a10', 'a11']                82
29    ['a03', 'a05', 'a10', 'a12']                83
30    ['a02', 'a04', 'a06', 'a07', 'a09']          84
31    ['a03', 'a04', 'a05', 'a08', 'a10']          85
32    number of solutions: 29                      86
33    cardinality distribution                    87
34    card.: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] 88
35    freq.: [0, 0, 0, 3, 24, 2, 0, 0, 0, 0, 0, 0, 0] 89
36    stability number : 5                      90
37    execution time: 0.00056 sec.            91
38    Results in self.misset                    92
39 >>> g.misset                                93
40     [frozenset({'a05', 'a07', 'a12'}),          94
41     frozenset({'a11', 'a01', 'a12', 'a09'}),      95
42     frozenset({'a11', 'a01', 'a02', 'a09'}),      96
43     ...                                         97
44     ...                                         98
45     frozenset({'a03', 'a10', 'a08', 'a11'}),      99
46     frozenset({'a03', 'a10', 'a12', 'a05'}),      100
47     frozenset({'a03', 'a10', 'a04', 'a05', 'a08'}) 101

```

All graphs and symmetric digraphs admit MISs, hence also kernels. In the context of digraphs, i.e. *oriented* graphs, the kernel concept gets much richer and separates from the symmetric MIS concept. 102
103
104

17.2 Initial and Terminal Kernels

105

In an oriented graph context, the internal stability condition of the kernel concept 106
 remains untouched; however, the external stability condition gets indeed split up by 107
 the orientation into two lateral cases: 108

1. A *dominant* stability condition, where each non selected node is dominated by at 109
 least one member of the kernel 110
2. An *absorbent* stability condition, where each non-selected node is absorbed by 111
 at least one member of the kernel 112

A both internally stable **and** dominant, respectively, absorbent choice is called a 113
 dominant or *initial*, respectively, an absorbent or *terminal* kernel. From a topological 114
 perspective, the initial kernel concept looks from the outside of the digraph into its 115
 interior, whereas the terminal kernel looks from the interior of a digraph towards its 116
 outside. From an algebraic perspective, the initial kernel is a prefix operand, and the 117
 terminal kernel is a postfix operand in the kernel equation systems (see Sect. 17.6 118
 on page 242). 119

Furthermore, as the kernel concept involves conjointly a positive logical refutation 120
 (the internal stability) and a positive logical affirmation (the external stability), 121
 it appeared rather quickly necessary in our operational developments to adopt a 122
 bipolar characteristic $[-1.0, 1.0]$ valuation domain, modelling logical negation by 123
 a change of numerical sign and including explicitly a third median value (0.0), 124
 expressing logical *indeterminateness*—neither positive nor negative (Bisdorff 2000, 125
 2002, 2004). 126

In such a bipolar-valued context, we call *prekernel* a choice which is *externally* 127
stable and for which the internal stability condition is *valid or indeterminate*. We 128
 say that the independence condition is in this case only *weakly validated*. Notice 129
 that all kernels are hence prekernels, but not vice versa. 130

In graphs or symmetric digraphs, where there is essentially no apparent *laterality*, 131
 all prekernels are initial and terminal at the same time. A universal example is given 132
 by the *complete* digraph. 133

Listing 17.3 The prekernels of a complete digraph

```

1  >>> from digraphs import CompleteDigraph
2  >>> u = CompleteDigraph(order=5)
3  >>> u
4  *----- Digraph instance description -----*
5  Instance class      : CompleteDigraph
6  Instance name       : complete
7  Digraph Order       : 5
8  Digraph Size        : 20
9  Valuation domain   : [-1.00 ; 1.00]
10 -----
11 >>> u.showPreKernels()
12 *** Computing preKernels ***
13 Dominant kernels :
```

14	['1'] ind. : 1.0; dom. : 1.0; abs. : 1.0	147
15	['2'] ind. : 1.0; dom. : 1.0; abs. : 1.0	148
16	['3'] ind. : 1.0; dom. : 1.0; abs. : 1.0	149
17	['4'] ind. : 1.0; dom. : 1.0; abs. : 1.0	150
18	['5'] ind. : 1.0; dom. : 1.0; abs. : 1.0	151
19	Absorbent kernels :	152
20	['1'] ind. : 1.0; dom. : 1.0; abs. : 1.0	153
21	['2'] ind. : 1.0; dom. : 1.0; abs. : 1.0	154
22	['3'] ind. : 1.0; dom. : 1.0; abs. : 1.0	155
23	['4'] ind. : 1.0; dom. : 1.0; abs. : 1.0	156
24	['5'] ind. : 1.0; dom. : 1.0; abs. : 1.0	157
25	----- statistics -----	158
26	graph name: complete	159
27	number of solutions	160
28	dominant kernels : 5	161
29	absorbent kernels: 5	162
30	cardinality frequency distributions	163
31	cardinality : [0, 1, 2, 3, 4, 5]	164
32	dominant kernel : [0, 5, 0, 0, 0, 0]	165
33	absorbent kernel: [0, 5, 0, 0, 0, 0]	166
34	Execution time : 0.00004 sec.	167
35	Results in sets: dompreKernels	168
36	and abspreKernels.	169

In a complete digraph, each single node is indeed both an initial and a terminal prekernel candidate and there is no definite begin or end of the digraph to be detected. Laterality is here entirely relative to a specific singleton chosen as reference point of view.

The same absence of laterality is apparent (see Listing 17.4) in two other universal digraph models, the *empty* and the *indeterminate* digraph.

Listing 17.4 The prekernels of the empty or indeterminate digraph

1	>>> from digraphs import EmptyDigraph	176
2	>>> ed = EmptyDigraph(order=5)	177
3	>>> ed.showPreKernels()	178
4	----- Computing preKernels -----*	179
5	Dominant kernel :	180
6	['1', '2', '3', '4', '5']	181
7	independence : 1.0	182
8	dominance : 1.0	183
9	absorbency : 1.0	184
10	Absorbent kernel :	185
11	['1', '2', '3', '4', '5']	186
12	independence : 1.0	187
13	dominance : 1.0	188
14	absorbency : 1.0	189
15	>>> from digraphs import IndeterminateDigraph	190
16	>>> id = IndeterminateDigraph(order=5)	191
17	>>> id.showPreKernels()	192
18	----- Computing preKernels -----*	193
19	Dominant prekernel :	194
20	['1', '2', '3', '4', '5']	195

21	independence	:	0.0	196
22	dominance	:	1.0	197
23	absorbency	:	1.0	198
24	Absorbent prekernel	:		199
25	['1', '2', '3', '4', '5']			200
26	independence	:	0.0	201
27	dominance	:	1.0	202
28	absorbency	:	1.0	203

In the empty digraph, the whole set of nodes gives indeed at the same time 204
 the unique initial and terminal prekernel (see Lines 6 and 11), similarly, for the 205
indeterminate digraph (see Lines 20 and 25). 206

Both these results make sense, as in a completely empty or indeterminate 207
 digraph, there is no *interior* of the digraph defined, only a *border* which is hence 208
 at the same time an initial and terminal prekernel (see Sect. 2.4). Notice, however, 209
 that in the latter indeterminate case, the complete set of nodes verifies only weakly 210
 the internal stability condition (see Lines 21 and 26). 211

Other common digraph models, although being clearly oriented, may show 212
 nevertheless no apparent laterality, like *chordless circuits*, i.e. holes surrounded by 213
 an oriented cycle—a circuit—of odd length. They do not admit in fact any initial or 214
 terminal prekernel. 215

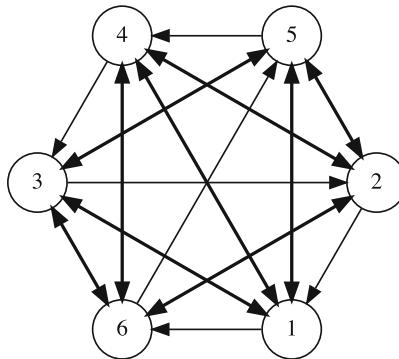
Listing 17.5 The prekernels of the 5-circuit digraph

1	>>> from	digraphs	import	CirculantDigraph	216
2	>>> c5 =	CirculantDigraph	(order=5,circulants=[1])		217
3	>>> c5.showPreKernels()				218
4	-----	statistics	-----		219
5	digraph	name:	c5		220
6	number	of	solutions		221
7	dominant	prekernels	:	0	222
8	absorbent	prekernels	:	0	223

Chordless circuits of *even* length $2 \times k$, with $k > 1$, contain however two 224
 isomorphic prekernels of cardinality k which qualify conjointly as initial and 225
 terminal candidates. 226

Listing 17.6 The prekernels of the 6-circuit digraph

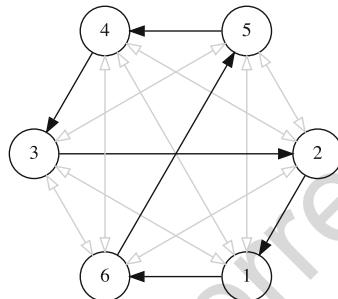
1	>>> c6 =	CirculantDigraph	(order=6,circulants=[1])	227	
2	>>> c6.showPreKernels()			228	
3	---	Computing	preKernels	---	229
4	Dominant	preKernels	:		230
5	['1', '3', '5']	ind.	:	1.0, dom. : 1.0, abs. : 1.0	231
6	['2', '4', '6']	ind.	:	1.0, dom. : 1.0, abs. : 1.0	232
7	Absorbent	preKernels	:		233
8	['1', '3', '5']	ind.	:	1.0, dom. : 1.0, abs. : 1.0	234
9	['2', '4', '6']	ind.	:	1.0, dom. : 1.0, abs. : 1.0	235



Digraph3 (graphviz), R. Bisdorff, 2020

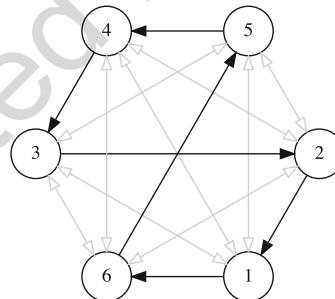
Fig. 17.2 The dual of the chordless 6-circuit. No initial or terminal prekernel—weakly independent and dominant, respectively, absorbent subset of nodes—may be found in this kind of digraphs

Dual of weak 6-circuit



Digraph3 (graphviz), R. Bisdorff, 2020

Converse of weak 6-circuit



Digraph3 (graphviz), R. Bisdorff, 2020

Fig. 17.3 Dual and converse transforms of the weak 6-circuit give the same digraph

Chordless circuits of even length may thus be indifferently oriented along two opposite directions. Notice by the way in Fig. 17.2 that the duals of all chordless circuits of odd or even length, i.e. *filled* circuits also called *anti-holes* (see Fig. 17.3), never contain any potential prekernel candidates.

Listing 17.7 The prekernels of the dual of the 6-circuit digraph

```

1 >>> dc6 = -c6      # dc6 = DualDigraph(c6)          240
2 >>> dc6.showPreKernels()                         241
3     ----- statistics -----
4     graph name: dual_c6                           242
5     number of solutions                         243
6     dominant prekernels : 0                      244

```

```

7  absorbent prekernels: 0
8  >>> dc6.exportGraphViz(fileName='dualChordlessCircuit')
9  *---- exporting a dot file for GraphViz tools ----*
10 Exporting to dualChordlessCircuit.dot
11 circo -Tpng dualChordlessCircuit.dot\
12   -o dualChordlessCircuit.png

```

We call *weak*, a chordless circuit with indeterminate inner part.

In Listing 17.8, we use the `IndeterminateInnerPart` parameter of the `CirculantDigraph` class for constructing such a weak chordless 6-circuit digraph. It is worth noticing in Fig. 17.3 on the previous page that the *dual* version of a weak circuit corresponds to its *converse* version.¹

Listing 17.8 The weak 6-circuit digraph

```

1  >>> from digraphs import CirculantDigraph
2  >>> c6 = CirculantDigraph(order=6, circulants=[1], \
3  ...                           IndeterminateInnerPart=True)
4  >>> (-c6).exportGraphViz()
5  *---- exporting a dot file for GraphViz tools ----*
6  Exporting to dual_c6.dot
7  circo -Tpng dual_c6.dot -o dual-c6.png
8  >>> (~c6).exportGraphViz()
9  *---- exporting a dot file for GraphViz tools ----*
10 Exporting to converse_c6.dot
11 circo -Tpng converse_c6.dot -o converse-c6.png

```

Weak chordless circuits of length n are in fact part of the class of digraphs that are invariant under the *codual* transform, $cn = -(\sim cn) = \sim(-cn)$.² When digraph cn is a weak chordless n -circuit, cn , $-cn$, $\sim cn$, and $\sim(-cn)$ will all admit the same set of prekernels.

17.3 Kernels in Lateralized Digraphs

Humans do live in an apparent physical space of plain transitive lateral orientation, fully empowered in finite geometrical 3D models with linear orders, where first, respectively, last ranked, nodes deliver unique initial, respectively, terminal, kernels.

¹ Not to be confused with the dual graph of a plane graph g that has a vertex for each face of g . Here we mean the *less than* (strict converse) relation corresponding to a *greater than or equal to* relation, or the *less than or equal to* relation corresponding to a (strict converse) *greater than* relation.

² The class of *self-codual* bipolar-valued digraphs consists of all weakly asymmetric digraphs, i.e. digraphs containing only asymmetric and/or indeterminate links. Limit cases consist of, on the one side, full tournaments with indeterminate reflexive links and, on the other side, fully indeterminate digraphs. In this class, the converse (inverse \sim) operator is indeed identical to the dual (negation $-$) one.

Similarly, in finite preorders, the first, respectively, last, equivalence classes deliver 276
 the unique initial, respectively, unique terminal, kernels. More generally, in finite 277
 partial orders, i.e. asymmetric and transitive digraphs, topological sort algorithms 278
 will easily reveal on the first, respectively, last, level all unique initial, respectively, 279
 terminal, kernels. 280

In genuine random digraphs, however, we may need to check for each of its 281
 MISs, whether one, both, or none of the lateralized external stability conditions may 282
 be satisfied. Consider, for instance, in Listing 17.9, the following random digraph 283
 instance of order 7 and generated with an arc probability of 30%. 284

Listing 17.9 Generating a random digraph `rd` of order 7 and arc probability 0.3

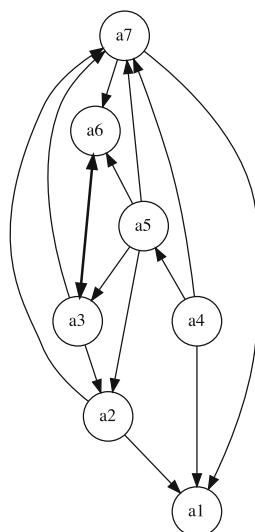
```

1 >>> from randomDigraphs import RandomDigraph 285
2 >>> rd = RandomDigraph(order=7, arcProbability=0.3, seed=5) 286
3 >>> rd.exportGraphViz('randomLaterality') 287
4     *---- exporting a dot file for GraphViz tools ---* 288
5     Exporting to randomLaterality.dot 289
6     dot -Grankdir=BT -Tpng randomLaterality.dot\ 290
7         -o randomLaterality.png 291

```

The random digraph shown in Fig. 17.4 has no apparent special properties, 292
 except from being connected (see Line 3 in Listing 17.10 on the following page). 293

Fig. 17.4 A random digraph instance of order 7 and arc probability 0.3. The digraph instance is neither asymmetric ($a_3 \leftrightarrow a_6$) nor symmetric ($a_2 \rightarrow a_1, a_1 \not\rightarrow a_2$), and the digraph is not transitive ($a_5 \rightarrow a_2 \rightarrow a_1$, but $a_5 \not\rightarrow a_1$)



Digraph3 (graphviz), R. Bisdorff, 2020

Listing 17.10 Inspecting the properties of random digraph rd

1 >>> rd.showComponents()	294
2 *--- Connected Components ---*	295
3 1: ['a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7']	296
4 >>> rd.computeSymmetryDegree(Comments=True)	297
5 Symmetry degree (%) of digraph <randomDigraph>:	298
6 arcs x>y: 14, symmetric: 1, asymmetric: 13	299
7 symmetric/arcs = 0.071	300
8 >>> rd.computeChordlessCircuits()	301
9 [] # no chordless circuits detected	302
10 >>> rd.computeTransitivityDegree(Comments=True)	303
11 Transitivity degree (%) of graph <randomDigraph>:	304
12 triples x>y>z: 23, closed: 11, open: 12	305
13 closed/triples = 0.478	306

There are no chordless circuits (see Line 9 above), and more than half of the required transitive closure is missing (see Line 13 above).

Now, we know that its potential prekernels must be among its set of maximal independent choices.

Listing 17.11 Inspecting the prekernels of random digraph rd

1 >>> rd.showMIS()	311
2 *--- Maximal independent choices ---*	312
3 ['a2', 'a4', 'a6']	313
4 ['a6', 'a1']	314
5 ['a5', 'a1']	315
6 ['a3', 'a1']	316
7 ['a4', 'a3']	317
8 ['a7']	318
9 >>> rd.showPreKernels()	319
10 *--- Computing preKernels ---*	320
11 Dominant preKernels :	321
12 ['a2', 'a4', 'a6']	322
13 independence : 1.0	323
14 dominance : 1.0	324
15 absorbency : -1.0	325
16 covering : 0.500	326
17 ['a4', 'a3']	327
18 independence : 1.0	328
19 dominance : 1.0	329
20 absorbency : -1.0	330
21 covering : 0.600	331
22 Absorbent preKernels :	332
23 ['a3', 'a1']	333
24 independence : 1.0	334
25 dominance : -1.0	335
26 absorbency : 1.0	336
27 covering : 0.500	337
28 ['a6', 'a1']	338
29 independence : 1.0	339
30 dominance : -1.0	340

31	absorbency : 1.0	341
32	covering : 0.600	342

Among the six MIs contained in random digraph `rd`, we discover in 343 Listing 17.11 on the preceding page 344 two initial and two terminal prekernels. 345 Notice by the way the covering values (between 0.0 and 1.0) shown by the 346 `showPreKernels()` method (Lines 16, 21, 27, and 32). The higher this value, 347 the more the corresponding prekernel candidate makes apparent the digraph's 348 laterality. In Fig. 17.5, the same digraph `rd` is redrawn by looking into its interior 349 via the best covering initial prekernel candidate: the dominant choice $\{a_3, a_4\}$ 350 (coloured in yellow), and looking out of it via the best covered terminal prekernel 351 candidate: the absorbent choice $\{a_1, a_6\}$ (coloured in blue). 351

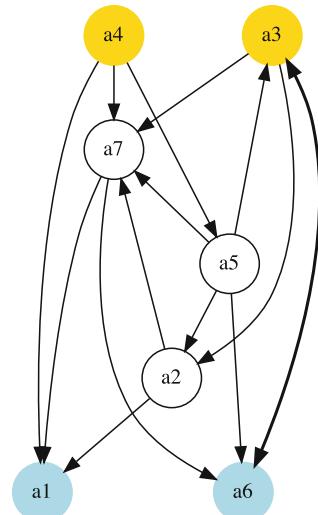
```

1 >>> rd.exportGraphViz(fileName='orientedLaterality', \
2 ...                         firstChoice=set(['a3', 'a4']), \
3 ...                         lastChoice=set(['a1', 'a6'])) \
4 *---- exporting a dot file for GraphViz tools ----*
5 Exporting to orientedLaterality.dot
6 dot -Grankdir=BT -Tpng orientedLaterality.dot \
7 -o orientedLaterality.png

```

As all reasonable bipolar-valued outranking digraphs usually show some 359 laterality—the marginal criteria preferences being not all totally contradictory— 360 initial and terminal prekernels provide convincing first, respectively, last, choice 361 recommendations as illustrated in Chap. 4 on page 41. 362

Fig. 17.5 A random digraph 359 oriented by best covering 360 initial and best covered 361 terminal prekernels 362



Digraph3 (graphviz), R. Bisdorff, 2020

17.4 Computing First and Last Choice Recommendations

363

To illustrate this idea, let us finally compute in Listing 17.12 first and last choice recommendations in a random bipolar-valued outranking digraph. 364
365

Listing 17.12 Generating a random bipolar-valued outranking digraph

```

1 >>> from outrankingDigraphs import\ 366
2 ... 367
3 >>> g = RandomBipolarOutrankingDigraph(368
4 >>> g 369
5 *----- Object instance description -----* 370
6 Instance class : RandomBipolarOutrankingDigraph 371
7 Instance name : randomOutranking 372
8 Actions : 7 373
9 Criteria : 7 374
10 Size : 26 375
11 Determinateness : 34.275 376
12 Valuation domain : {'min': -100.0, 'med': 0.0, 'max': 100.0} 377
13 >>> g.showHTMLPerformanceTableau() 378

```

The associated random performance tableau shown in Fig. 17.6 reveals the 379 performance evaluations of 7 potential decision alternatives with respect to 7 380 decision criteria supporting each an increasing performance scale from 0.0 to 100.0. 381 Notice the missing performance data concerning decision alternatives a2 and a5. 382 The resulting *strict outranking*—i.e. a weighted majority supported—‘*better than* 383 *without considerable counter-performance*’—digraph is shown in Fig. 17.7 on the 384 facing page. 385

```

1 >>> gcd = ~(-g) # Codual: the converse of the negation 386
2 >>> gcd.exportGraphViz(fileName='tutOutRanking') 387
3 *---- exporting a dot file for GraphViz tools ----* 388
4 Exporting to tutOutranking.dot 389
5 dot -Grankdir=BT -Tpng tutOutranking.dot\ 390
       -o tutOutranking.png 391

```

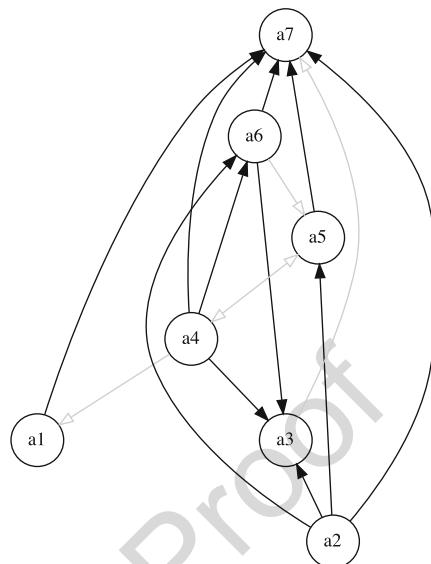
Performance table randomOutranking

criterion	g1	g2	g3	g4	g5	g6	g7
a1	64.90	1.31	13.88	98.24	94.10	14.57	31.00
a2			61.75	87.24	69.06	6.51	81.85
a3	11.32	27.95	12.67	28.93	96.66	30.14	48.07
a4	46.91	91.63	0.18	96.15	89.37	60.31	31.58
a5		76.57	87.14	53.92	29.88	0.34	48.12
a6	54.38	15.96	20.95	67.78	36.12	67.79	70.47
a7	57.39	79.71	21.55	20.48	16.60	33.79	5.70

Fig. 17.6 The performance tableau of a random outranking digraph instance

Fig. 17.7 A random strict outranking digraph instance.

All decision alternatives appear strictly better performing than alternative a_7 . We call it a CONDORCET loser and it is an evident terminal prekernel candidate. On the other side, three alternatives: a_1 , a_2 , and a_4 are not dominated. They give together an initial prekernel candidate



Digraph3 (graphviz), R. Bisdorff, 2020

Listing 17.13 Computing the prekernels of the strict outranking digraph gcd

```

1 >>> gcd.showPreKernels()
2     *--- Computing preKernels ---*
3     Dominant preKernels :
4         ['a1', 'a2', 'a4']
5             independence : 0.00
6             dominance   : 6.98
7             absorbency   : -48.84
8             covering     : 0.667
9     Absorbent preKernels :
10        ['a3', 'a7']
11            independence : 0.00
12            dominance   : -74.42
13            absorbency   : 16.28
14            covered       : 0.800

```

With such unique disjoint initial and terminal prekernels (see Lines 4 and 10 in Listing 17.13), the random digraph gcd is hence clearly lateralized. Indeed, these initial and terminal prekernels of the codual outranking digraph reveal *first*, respectively, *last*, choice recommendations one may formulate on the basis of a given outranking digraph instance.

Listing 17.14 Computing a first and last choice recommendation from digraph gcd

```

1 >>> g.showBestChoiceRecommendation()
2     Rubis best choice recommendation(s) (BCR)
3         (in decreasing order of determinateness)

```

```

4   Credibility domain: [-100.00,100.00] 414
5   === >> potential first choice(s) 415
6   * choice : ['a1', 'a2', 'a4'] 416
7   independence : 0.00 417
8   dominance : 6.98 418
9   absorbency : -48.84 419
10  covering (%) : 66.67 420
11  determinateness (%) : 57.97 421
12  - most credible action(s) = 422
13    {'a4': 20.93, 'a2': 20.93} 423
14  === >> potential last choice(s) 424
15  * choice : ['a3', 'a7'] 425
16  independence : 0.00 426
17  dominance : -74.42 427
18  absorbency : 16.28 428
19  covered (%) : 80.00 429
20  determinateness (%) : 64.62 430
21  - most credible action(s) = { 'a7': 48.84, } 431

```

Notice in Lines 13 and 21 in Listing 17.14 on the preceding page that solving the valued kernel equation system provides furthermore a positive characterisation of the most credible decision alternatives in each respective choice recommendation. Alternatives a_2 and a_4 give equivalent candidates for a unique *first* choice, and alternative a_7 is clearly confirmed as the *last* choice.

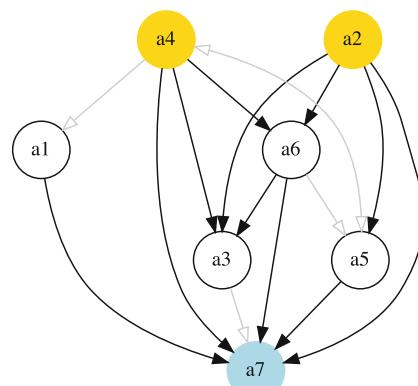
In Fig. 17.8, we orient the drawing of the strict outranking digraph instance with the help of these first and last choice recommendations.

```

1 >>> gcd.exportGraphViz(fileName='firstLastOrientation', \
2 ...                           firstChoice=['a2', 'a4'], \
3 ...                           lastChoice=['a7']) 439
4 *---- exporting a dot file for GraphViz tools ----* 440
5 Exporting to firstLastOrientation.dot 441
6 dot -Grankdir=BT -Tpng firstLastOrientation.dot\ 442
7   -o firstLastOrientation.png 443

```

Fig. 17.8 The strict outranking digraph oriented by its *first* and *last* choice recommendations. The grey arrows, like the one between alternatives a_4 and a_1 , represent indeterminate preferential situations. Alternative a_1 appears hence to be rather incomparable to all the other, except alternative a_7



DiGraph3 (graphviz), R. Bisдорff, 2020

Heatmap of Performance Tableau 'randomOutranking'

criteria	g4	g7	g5	g6	g1	g2	g3
weights	9	10	6	5	4	8	1
tau ^(*)	+0.64	+0.40	+0.29	+0.17	+0.02	-0.05	-0.10
a4	96	32	89	60	47	92	0
a2	87	82	69	7	NA	NA	62
a6	68	70	36	68	54	16	21
a1	98	31	94	15	65	1	14
a5	54	48	30	0	NA	77	87
a3	29	48	97	30	11	28	13
a7	20	6	17	34	57	80	22

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%
----------	--------	--------	--------	--------	---------

(*) tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation**Ranking rule:** **Copeland**Ordinal (Kendall) correlation between global ranking and global outranking relation: **+0.848****Fig. 17.9** Heatmap with Copeland ranking of the performance tableau

It may be interesting to compare this result with a COPELAND ranking of the underlying performance tableau (see Sect. 8.2 on ranking with incommensurable criteria).

```
1 >>> g.showHTMLPerformanceHeatmap(colorLevels=5, ndigits=0, \
2 ... Correlations=True, rankingRule='Copeland')
```

In the resulting linear ranking (see Fig. 17.9), alternative a4 is set at first rank, followed by alternative a2. This makes sense as alternative a4 shows three performances in the first quintile, whereas a2 is only partially evaluated and shows only two such excellent performances. But a4 also shows a very weak performance in the first quintile. Both decision actions, hence, do not show eventually a performance profile that would make apparent a clear preference situation in favour of one or the other. In this sense, the prekernels based best choice recommendations may appear more faithful with respect to the actually definite strict outranking relation than any 'forced' linear ranking result as shown in Fig. 17.9.

17.5 Tractability of Kernel Computation

Let us give some hints on the *tractability* of prekernel computations. Detecting all prekernels in a digraph is a computationally difficult problem. Checking external stability conditions for an independent choice is equivalent to checking its maximality and may be done in the linear complexity of the order of the digraph. However, checking all independent choices contained in a digraph may get hard already for tiny sparse digraphs of order $n > 20$ (Bisdorff 2006). Indeed, the worst case is given by an empty or indeterminate digraph where the set of all potential independent choices to check is in fact the power set of the vertices.

```

1 >>> from digraphs import EmptyDigraph 469
2 >>> e = EmptyDigraph(order=20) 470
3 >>> e.showMIS() # by visiting all 2^20 independent choices 471
4     *--- Maximal independent choices ---* 472
5     [ '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 473
6     '11', '12', '13', '14', '15', '16', '17', '18', '19', '20'] 474
7     number of solutions: 1 475
8     execution time: 1.47640 sec. # <----- 476
9 >>> 2**20 477
10    1048576 478

```

Now, there exist more efficient specialised algorithms for directly enumerating MISs and dominant or absorbent kernels contained in specific digraph models without visiting all independent choices (Bisdorff 2006). *Alain Hertz* provided kindly such a MISs enumeration algorithm for the DIGRAPH3 project (the `showMIS_AH()` method). When the number of independent choices is big compared to the actual number of MISs, like in very sparse or empty digraphs, the performance difference may be dramatic (see Line 8 above and Line 16 below).

Listing 17.15 Enumerating MISs by visiting only maximal independent choices (*A. Hertz*)

```

1 >>> e.showMIS_AH() 486
2     # by visiting only maximal independent choices 487
3     *-----* 488
4     * Python implementation of Hertz's * 489
5     * algorithm for generating all MISs * 490
6     * R.B. version 7(6)-25-Apr-2006 * 491
7     *-----* 492
8     ==>>> Initial solution : 493
9     [ '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', 494
10    '12', '13', '14', '15', '16', '17', '18', '19', '20'] 495
11    *--- results ---* 496
12    [ '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', 497
13    '12', '13', '14', '15', '16', '17', '18', '19', '20'] 498
14    *--- statistics ---* 499
15    MIS solutions : 1 500
16    execution time : 0.00026 sec. # <---- 501
17    iteration history: 1 502

```

For more or less dense strict outranking digraphs of modest order, as facing usually in MCDA applications, enumerating all independent choices remains however in most cases tractable, especially by using a very efficient iterator generator with the `independentChoices()` method shown in Listing 17.16.

Listing 17.16 Generating all independent choices in a digraph

```

1 def independentChoices (self,U): 507
2     """
3         Generator for all independent choices with associated 508
4         dominated, absorbed and independent neighborhoods 509
5         of digraph instance self. 510

```

```

6  Initiate with U = self.singletons().                                512
7  Yields [(independent choice, domnb, absnb, indnb)].            513
8  """
9  if U == []:
10     yield [(frozenset(), set(), set(), set(self.actions))]      515
11  else:
12      x = list(U.pop())
13      for S in self.independentChoices(U):                          516
14          yield S
15          if x[0] <= S[0][3]:                                         517
16              Sxgamdom = S[0][1] | x[1]                                518
17              Sxgamabs = S[0][2] | x[2]                                519
18              Sxindep = S[0][3] & x[3]                                520
19              Sxchoice = S[0][0] | x[0]                                521
20              Sx = [(Sxchoice, Sxgamdom, Sxgamabs, Sxindep)]        522
21          yield Sx

```

And, checking maximality of independent choices via the external stability conditions during their enumeration with the `computePreKernels()` method shown in Listing 17.17 provides the effective advantage of computing all initial and terminal prekernels in a single loop (see Line 10 and Bisdorff 2006).

Listing 17.17 Computing dominant and absorbent prekernels

```

1 def computePreKernels(self):                                         532
2     """
3     computing dominant and absorbent preKernels:
4     Result in self.dompreKernels and self.abspreKernels
5     """
6     actions = set(self.actions)                                       533
7     n = len(actions)                                                 534
8     dompreKernels = set()                                             535
9     abspreKernels = set()                                             536
10    for choice in self.independentChoices(self.singletons()):        537
11        restactions = actions - choice[0][0]                           538
12        if restactions <= choice[0][1]:                                 539
13            dompreKernels.add(choice[0][0])                            540
14        if restactions <= choice[0][2]:                                 541
15            abspreKernels.add(choice[0][0])                            542
16    self.dompreKernels = dompreKernels                                543
17    self.abspreKernels = abspreKernels

```

Finally, we use our bipolar-valued epistemic logic framework for computing the credibility that an individual node of the digraph fits with being a member of an initial or terminal prekernel. For this purpose, we use kernel equation systems (Schmidt and Ströhlein 1985).

17.6 Solving Kernel Equation Systems

553

Let $G(X, R)$ be a crisp irreflexive digraph defined on a finite set X of nodes and where R is the corresponding $\{-1, +1\}$ -valued adjacency matrix. Let Y be the $\{-1, +1\}$ -valued membership characteristic (row) vector of a choice in X .

When Y satisfies the following equation system:

556

$$Y \circ R = -Y, \quad (17.1)$$

where for all x in X ,

558

$$(Y \circ R)(x) = \max_{y \in X, x \neq y} (\min(Y(x), R(x, y))), \quad (17.2)$$

then Y characterises an *initial kernel* (Bisdorff 2006; Bisdorff et al. 2006).

559

When transposing now the membership characteristic vector Y into a column vector Y^t , the following equation system: $R \circ Y^t = -Y^t$ makes Y^t similarly characterise a *terminal kernel*.

561

Let us verify this result in Listing 17.18 on a tiny random digraph.

563

Listing 17.18 Verifying the kernel equation system on a tiny random digraph

```

1 >>> from digraphs import RandomDigraph
2 >>> g = RandomDigraph(order=3, seed=1)
3 >>> g.showRelationTable()
4 * ---- Relation Table ----
5   R | 'a1'  'a2'  'a3'
6   ---|-----
7   'a1' | -1    +1    -1
8   'a2' | -1    -1    +1
9   'a3' | +1    +1    -1
10 >>> g.showPreKernels()
11 *--- Computing preKernels ---*
12 Dominant preKernels :
13   ['a3']
14 Absorbent preKernels :
15   ['a2']

```

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

It is easy to verify by hand that the characteristic vector $[-1, -1, +1]$ satisfies the initial kernel equation system; node a_3 gives an *initial kernel*. Similarly, the characteristic vector $[-1, +1, -1]$ verifies indeed the terminal kernel equation system and node a_2 gives hence a *terminal kernel*.

579

580

581

582

We succeeded now in generalising crisp kernel equation systems to genuine bipolar-valued digraphs (Bisdorff 2006; Bisdorff et al. 2006). The constructive proof, found by *Marc Pirlot*, is based on the following *fixpoint equation* that may be used for computing bipolar-valued kernel membership vectors:

583

584

585

586

$$T(Y) := -(Y \circ R) = Y. \quad (17.3)$$

John von Neumann showed indeed that, when a digraph $G(X, R)$ is acyclic with a unique initial kernel K characterised by its membership characteristics vector Y_k , then the following double fixpoint equation:

$$T^2(Y) := -(- (Y \circ R) \circ R) = Y \quad (17.4)$$

will admit a stable *high* and a stable *low* fixpoint solutions that converge both to Y_k (Schmidt and Ströhlein 1985).

Inspired by the crisp double fixpoint equation 17.4, we observed that for a given bipolar-valued digraph $G(X, R)$, each one of its dominant or absorbent prekernels K_i in X determines an induced partial digraph $G(X, R/K_i)$ which is acyclic and admits K_i as unique prekernel (Bisdorff 1997).

Following the von Neumann fixpoint algorithm, a similar bipolar-valued extended double fixpoint algorithm, applied to $G(X, R/K_i)$, allows us to compute the associated bipolar-valued kernel characteristic vectors Y_i in polynomial complexity.

Algorithm 17.1 Computing bipolar-valued kernel characteristic vectors

in : bipolar-valued digraph $G(X, R)$,

out : set $\{Y_1, Y_2, \dots\}$ of bipolar-valued kernel membership characteristic vectors.

1. Enumerate all initial and terminal prekernels K_1, K_2, \dots in the given bipolar-valued digraph (see Listing 17.17 on page 241);
 2. For each crisp initial kernel K_i :
 - a. Construct a partially determined subgraph $G(X, R/K_i)$ supporting exactly this unique initial kernel K_i ;
 - b. Use the double fixpoint equation T^2 (17.4) with the partially determined adjacency matrix R/K_i for computing a stable low and a stable high fixpoint;
 - c. Determine the bipolar-valued K_i -membership characteristic vector Y_i with an epistemic disjunction of the previous low and high fixpoints;
 3. Repeat step (2) for each terminal kernel K_j by using the double fixpoint equation T^2 with the transpose of the adjacency matrix R/K_j .
-

Time for a Practical Illustration

We reconsider the random digraph g generated in Listing 17.12 on page 236. Digraph g models the pairwise outranking situations between seven decision alternatives evaluated on seven incommensurable performance criteria. We recompute its corresponding bipolar-valued prekernels on the associated codual digraph gcd .

```

1 >>> gcd = ~(-g) # strict outranking digraph
2 >>> gcd.showPreKernels()
3     *** Computing prekernels ***
4     Dominant prekernels :
5         ['a1', 'a4', 'a2']
6         independence : +0.000

```

600

601

602

603

604

605

606

607

608

609

610

```

7      dominance      : +0.070          611
8      absorbency    : -0.488          612
9      covering       : +0.667          613
10     Absorbent prekernels :          614
11     ['a7', 'a3']      615
12     independence   : +0.000          616
13     dominance      : -0.744          617
14     absorbency     : +0.163          618
15     covered         : +0.800          619
16     ----- statistics -----          620
17     graph name: converse-dual_rel_randomperftab 621
18     number of solutions          622
19     dominant kernels : 1          623
20     absorbent kernels: 1          624
21     cardinality frequency distributions          625
22     cardinality      : [0, 1, 2, 3, 4, 5, 6, 7] 626
23     dominant kernel   : [0, 0, 0, 1, 0, 0, 0, 0] 627
24     absorbent kernel: [0, 0, 1, 0, 0, 0, 0, 0] 628
25     Execution time   : 0.00022 sec.          629

```

The codual outranking digraph `gcd`, modelling a strict outranking relation, admits an initial prekernel $\{a_1, a_2, a_4\}$ and a terminal one $\{a_3, a_7\}$ (see Lines 5 and 11 above).

In Listing 17.19, we now compute, with the `domkernelrestrict()` method, the initial prekernel-restricted adjacency table (see Fig. 17.10).

Listing 17.19 Computing a dominant prekernel restricted adjacency table

```

1 >>> k1Relation = gcd.domkernelrestrict(['a1','a2','a4'])          635
2 >>> gcd.showHTMLRelationTable(\                                     636
3 ...     actionsList=['a1','a2','a4','a3','a5','a6','a7'],\          637
4 ...     relation=k1Relation,\                                     638
5 ...     tableTitle='K1 restricted adjacency table')          639

```

The corresponding initial prekernel membership characteristic vector may be computed with the `computeKernelVector()` method.

Fig. 17.10 Initial kernel $\{a_1, a_2, a_4\}$ restricted adjacency table

The outranking situation between alternatives a_4 and a_1 being *indeterminate*, this initial prekernel is only weakly independent

K1 restricted adjacency table

r(x S y)	a1	a2	a4	a3	a5	a6	a7
a1	-	-0.23	-1.00	0.00	0.00	0.00	0.16
a2	-0.21	-	-0.21	0.21	0.44	0.05	0.49
a4	0.00	-0.21	-	0.21	0.00	0.07	0.58
a3	-0.28	-0.21	-0.74	-	0.00	0.00	0.00
a5	-0.26	-0.67	0.00	0.00	-	0.00	0.00
a6	-0.12	-0.49	-0.49	0.00	0.00	-	0.00
a7	-0.51	-0.49	-0.86	0.00	0.00	0.00	-

Valuation domain: [-1.00; +1.00]

Listing 17.20 Fixpoint iterations for initial prekernel {a1, a2, a4}

```

1 >>> gcd.computeKernelVector(['a1','a2','a4'], \
2 ...                               Initial=True,Comments=True)
3 --> Initial prekernel: {'a1', 'a2', 'a4'}
4 initial low vector: [-1.00,-1.00,-1.00,-1.00,-1.00,-1.00]
5 initial high vector: [+1.00,+1.00,+1.00,+1.00,+1.00,+1.00]
6 1st low vector     : [ 0.00,+0.21,-0.21, 0.00,-0.44,-0.07,-0.58]
7 1st high vector   : [+1.00,+1.00,+1.00,+1.00,+1.00,+1.00]
8 2nd low vector    : [ 0.00,+0.21,-0.21, 0.00,-0.44,-0.07,-0.58]
9 2nd high vector   : [ 0.00,+0.21,-0.21,+0.21,-0.21,-0.05,-0.21]
10 3rd low vector   : [ 0.00,+0.21,-0.21,+0.21,-0.21,-0.07,-0.21]
11 3rd high vector  : [ 0.00,+0.21,-0.21,+0.21,-0.21,-0.05,-0.21]
12 4th low vector   : [ 0.00,+0.21,-0.21,+0.21,-0.21,-0.07,-0.21]
13 4th high vector : [ 0.00,+0.21,-0.21,+0.21,-0.21,-0.07,-0.21]
14 Iterations       : 4
15 low & high fusion: [ 0.00,+0.21,-0.21,+0.21,-0.21,-0.07,-0.21]
16 Choice vector for initial prekernel: {'a1', 'a2', 'a4'}
17   'a2': +0.21
18   'a4': +0.21
19   'a1':  0.00
20   'a6': -0.07
21   'a3': -0.21
22   'a5': -0.21
23   'a7': -0.21

```

In Listing 17.20, we start the fixpoint computation with an empty set characterisation as first low vector and a complete set X characterising high vector. After each iteration, the low vector is set to the negation of the previous high vector and the high vector is set to the negation of the previous low vector.

A unique stable prekernel characteristic vector Y_1 is here attained at the fourth iteration with positive members $a2: +0.21$ and $a4: +0.21$ (60.5% criteria significance majority), $a1: 0.00$ being an ambiguous potential member. Alternatives $a3, a5, a6$, and $a7$ are all negative members, i.e. positive *non members* of this outranking prekernel.

Let us also compute the restricted adjacency table for the outranked, i.e. the *terminal* prekernel {a3, a7} (Fig. 17.11).

```

1 >>> k2Relation = gcd.abskernelrestrict(['a3','a7'])
2 >>> gcd.showHTMLRelationTable('
3 ...   actionsList=['a3','a7','a1','a2','a4','a5','a6'], \
4 ...   relation=k2Relation, \
5 ...   tableTitle='K2 restricted adjacency table')

```

Fig. 17.11 Terminal prekernel {a3, a7} restricted adjacency table. Again, we notice that this terminal prekernel is only weakly independent

K2 restricted adjacency table

rx s y	a3	a7	a1	a2	a4	a5	a6
a3	-	0.00	-0.28	-0.21	-0.74	-0.40	-0.53
a7	-1.00	-	-0.51	-0.49	-0.86	-0.67	-0.63
a1	0.00	0.16	-	0.00	0.00	0.00	0.00
a2	0.21	0.49	0.00	-	0.00	0.00	0.00
a4	0.21	0.58	0.00	0.00	-	0.00	0.00
a5	0.00	0.16	0.00	0.00	0.00	-	0.00
a6	0.30	0.26	0.00	0.00	0.00	0.00	-

Valuation domain: [-1.00; +1.00]

The corresponding bipolar-valued characteristic vector Y_2 may be computed as follows: 681
682

```

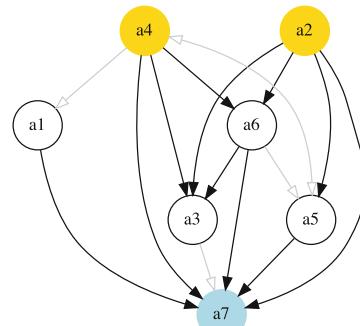
1 >>> gcd.computeKernelVector(['a3','a7'], \
2                                Initial=False,Comments=True) 683
3 --> Terminal prekernel: {'a3', 'a7'} 684
4 initial low vector : [-1.00,-1.00,-1.00,-1.00,-1.00,-1.00,-1.00] 685
5 initial high vector : [+1.00,+1.00,+1.00,+1.00,+1.00,+1.00,+1.00] 686
6 1st low vector : [-0.16,-0.49, 0.00,-0.58,-0.16,-0.30,+0.49] 687
7 1st high vector : [+1.00,+1.00,+1.00,+1.00,+1.00,+1.00,+1.00] 688
8 2nd low vector : [-0.16,-0.49, 0.00,-0.58,-0.16,-0.30,+0.49] 689
9 2nd high vector : [-0.16,-0.49, 0.00,-0.49,-0.16,-0.26,+0.49] 690
10 3rd low vector : [-0.16,-0.49, 0.00,-0.49,-0.16,-0.26,+0.49] 691
11 3rd high vector : [-0.16,-0.49, 0.00,-0.49,-0.16,-0.26,+0.49] 692
12 Iterations : 3 693
13 high & low fusion : [-0.16,-0.49, 0.00,-0.49,-0.16,-0.26,+0.49] 694
14 Choice vector for terminal prekernel: {'a3','a7'} 695
15 'a7': +0.49 696
16 'a3': 0.00 697
17 'a1': -0.16 698
18 'a5': -0.16 699
19 'a6': -0.26 700
20 'a2': -0.49 701
21 'a4': -0.49 702
22

```

A unique stable bipolar-valued high and low fixpoint is attained at the third iteration with alternative a_7 positively confirmed (about $(1.0 + 0.49)/2 = 0.75\%$ criteria significance majority, see Line 15 above) as member of this terminal prekernel, whereas the membership of alternative a_3 in this prekernel appears indeterminate. All the remaining nodes have negative membership characteristic values and are hence positively excluded from this prekernel. 703

When we reconsider the graphviz drawing of this outranking digraph in Fig. 17.12, it becomes obvious here why alternative a_1 is *neither included nor excluded* from the initial prekernel. The same observation is applicable to alternative a_3 that can *neither be included nor excluded* from the terminal prekernel. It may even happen, in case of more indeterminate outranking situations, that no alternative is positively included or excluded from a weakly independent prekernel, the corresponding bipolar-valued membership characteristic vector being completely indeterminate. 710
711
712
713
714
715
716
717

Fig. 17.12 The strict outranking digraph oriented by its initial and terminal prekernels. Notice the indeterminate outranking situation between alternatives a_4 and a_1 and the same indeterminate outranking situation between alternatives a_3 and a_7



Digraph3 (graphviz), R. Bisдорff, 2020

To illustrate finally why sometimes it is necessary to operate an epistemic disjunctive fusion of unequal stable low and high membership characteristic vectors (see Algorithm 17.1 on page 243, Step 2c), let us consider, for instance, the following crisp 7-cycle graph:

```

1 >>> from digraphs import CirculantDigraph 722
2 >>> c7 = CirculantDigraph(order=7,circulants=[-1,1]) 723
3 >>> c7 724
4 *----- Digraph instance description -----* 725
5 Instance class : CirculantDigraph 726
6 Instance name : c7 727
7 Digraph Order : 7 728
8 Digraph Size : 14 729
9 Valuation domain : [-1.00;1.00] 730
10 Determinateness (%) : 100.00 731
11 Attributes : ['name','order','circulants', 732
12 'actions','valuationdomain', 733
13 'relation','gamma','notGamma'] 734

```

Digraph $c7$ is a symmetric crisp digraph showing, among others, the maximal independent set $\{2,5,7\}$, i.e. an initial as well as terminal kernel. The `computeKernelVector()` method computes the corresponding initial kernel characteristic vector.

```

1 >>> c7.computeKernelVector(['2','5','7'],\ 739
2 ... 740
3 --> Initial kernel: {'2', '5', '7'} 741
4 initial low vector : [-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0] 742
5 initial high vector : [+1.0, +1.0, +1.0, +1.0, +1.0, +1.0, +1.0] 743
6 1 st low vector : [-1.0, 0.0, -1.0, -1.0, 0.0, -1.0, 0.0] 744
7 1 st high vector : [+1.0, +1.0, +1.0, +1.0, +1.0, +1.0, +1.0] 745
8 2 nd low vector : [-1.0, 0.0, -1.0, -1.0, 0.0, -1.0, 0.0] 746
9 2 nd high vector : [ 0.0, +1.0, 0.0, 0.0, +1.0, 0.0, +1.0] 747
10 stable low vector : [-1.0, 0.0, -1.0, -1.0, 0.0, -1.0, 0.0] 748
11 stable high vector : [ 0.0, +1.0, 0.0, 0.0, +1.0, 0.0, +1.0] 749
12 Iterations : 3 750
13 low & high fusion : [-1.0, +1.0, -1.0, -1.0, +1.0, -1.0, +1.0] 751
14 Choice vector for initial prekernel: {'2', '5', '7'} 752
15 '7': +1.00 753
16 '5': +1.00 754
17 '2': +1.00 755
18 '6': -1.00 756
19 '4': -1.00 757
20 '3': -1.00 758
21 '1': -1.00 759

```

Notice that the stable low vector characterises the *negative membership* part, whereas the stable high vector characterises the *positive membership* part (see Lines 10–11 above). The bipolar epistemic fusion assembles eventually both stable parts into the correct prekernel characteristic vector (Line 13).

The adjacency matrix of a symmetric digraph staying *unchanged* by the transition operator, the previous computations, when qualifying the same kernel as a terminal instance, will hence produce exactly the same result.

It is worthwhile noticing the essential computational role, the logical indeterminate value 0.0 is playing in this double fixpoint algorithm. To implement such

kind of algorithms without a logical *neutral* term would be like implementing 769 numerical algorithms without a possible usage of the number 0. Infinitely many 770 trivial impossibility theorems and dubious logical results come up. 771

Notes

772

Following the observation that an independent absorbent choice in an acyclic 773 digraph corresponds to the zero values of the associated GRUNDY function, Riguet 774 (1948) introduced the name “noyau” (kernel) for such a choice. Terminal kernels 775 where in the sequel studied by Berge (1962) in the context of Combinatorial Game 776 Theory. Initial kernels—Independent and dominating choices—were introduced 777 under the name game solutions by von Neumann and Morgenstern (1944). The 778 absorbent version of the crisp kernel equation system 17.1 on page 242 was 779 first introduced by Schmidt and Ströhlein (1985) in the context of their thorough 780 exploration of relational algebra. 781

The fuzzy version of kernel equation systems was first investigated by Kitainik 782 (1993). Commenting on this work at a meeting in Spring 1995 of the EURO 783 Working Group on Multicriteria Decision Aiding in Lausanne (Switzerland), Marc 784 Roubens feared that solving such fuzzy kernel equation systems could be computationally 785 difficult. Triggered by his pessimistic remark and knowing about kernel 786 equation systems and the VON NEUMANN fixpoint theorem (Schmidt and Ströhlein 787 1985; von Neumann and Morgenstern 1944), I immediately started to implement in 788 Prolog a solver for the valued version of Eq. 17.4 on page 243, the equation system 789 serving as constraints for a discrete labelling of all possible rational solution vectors. 790 And in Summer 1995, we luckily obtained with a commercial finite domain solver 791 the very first valued initial and terminal kernels from a didactic outranking digraph 792 of order 8, well known in the multiple-criteria decision aiding community. The 793 computation took several seconds on a CRAY 6412 superserver with 12 processors 794 operating in a nowadays ridiculous CPU speed of 90 MHz. The labelled solution 795 vectors, obtained in the sequel for any outranking digraph with a single initial or 796 terminal kernel, were structured in a way that suggested the converging stages of the 797 VON NEUMANN fixpoint algorithm and so gave the initial hint for Algorithm 17.1 798 (Bisdorff 1996, 1997; Bisdorff and Roubens 2004). 799

In our present Python 3.9 implementation, such a tiny problem is solved in less 800 than a thousandth of a second on a common laptop. And this remains practically the 801 same for any relevant example of outranking digraph observed in a real decision- 802 aiding problem. Several times we wrote in our personal journal that there is certainly 803 now no more potential for any substantial improvement of this computational 804 efficiency; Only to discover, shortly later, that following a new theoretical idea 805 or choosing a more efficient implementation—using, for instance, the amazing 806 instrument of iterator generators in Python—execution times could well be divided 807 by 20. 808

This nowadays available computational efficiency confers the bipolar-valued kernel concept a methodological premium for solving specific decision problems on the basis of the bipolar-valued outranking digraph (see Chaps. 4 and 12). But it also opens new opportunities for verifying and implementing kernel extraction algorithms for more graph theoretical purposes. New results, like enumerating the non-isomorphic maximal independent sets—the kernels—of known difficult graph instances like the n -cycle, could be obtained (see Sect. 21.7 and Bisдорff and Marichal 2008). 819
810
811
812
813
814
815
816

References

817

- Berge C (1962) The theory of graphs (original title: The theory of graphs and its applications). Dover 2001 (originally published by Wiley 1962). Translated from a French edition by Dunod, 1958 818
819
820
- Bisдорff R (1996) On computing kernels on fuzzy simple graphs by combinatorial enumeration using a CPL(FD) system. In: 8th Benelux workshop on logic programming, Louvain-la-Neuve (BE), 9 September 1996, Université catholique de Louvain. <http://hdl.handle.net/10993/46933> 821
822
823
- Bisдорff R (1997) On computing kernels on l-valued simple graphs. In: Proceedings of EUFIT'97, 5th European congress on fuzzy and intelligent technologies, Aachen, September 8–11, 1997, pp 97–103 824
825
826
- Bisдорff R (2000) Logical foundation of fuzzy preferential systems with application to the electre decision aid methods. Comput Oper Res 27:673–687. <http://hdl.handle.net/10993/23724> 827
828
- Bisдорff R (2002) Logical foundation of multicriteria preference aggregation. In: Bouyssou D, Jacquet-Lagrèze E, Perny P, Stowiński R, Vanderpooten D, Vincke P (eds) Aiding decisions with multiple criteria. Kluwer Academic Publishers, pp 379–403. <http://hdl.handle.net/10993/45313> 829
830
831
832
- Bisдорff R (2004) On a natural fuzzification of Boolean logic. In: Klement EP, Pap E (eds) Linz seminar on fuzzy set theory, mathematics of fuzzy systems. Bildungszentrum St. Magdalena, Linz (Austria), pp 20–26. <http://hdl.handle.net/10993/23721> 833
834
835
- Bisдорff R (2006) On enumerating the kernels in a bipolar-valued digraph. Annales du Lamsade 6:1–38. <http://hdl.handle.net/10993/38741> 836
837
- Bisдорff R, Marichal J (2008) Counting non-isomorphic maximal independent sets of the n -cycle graph. J Integer Seq 11(Art. 08.5.7):1–16. <https://cs.uwaterloo.ca/journals/JIS/VOL11/Marichal/marichal.html> 838
839
840
- Bisдорff R, Roubens M (2004) Choice procedures in pairwise comparison multiple-attribute decision making methods. In: Berghammer R, Möller B, Struth G (eds) Relational and Kleene-algebraic methods in computer science: 7th international seminar on relational methods in computer science and 2nd international workshop on applications of Kleene algebra, Bad Malente, May 12–17, 2003. Lecture notes in computer science, vol 3051. Springer, Heidelberg, pp 1–7 841
842
843
844
845
846
- Bisдорff R, Pirlot M, Roubens M (2006) Choices and kernels from bipolar valued digraphs. Eur J Oper Res 175:155–170. <http://hdl.handle.net/10993/23720> 847
848
- Kitainik L (1993) Fuzzy decision procedures with binary relations: towards a unified theory. Kluwer Academic Publisher, Boston 849
850
- Riguet J (1948) Relations binaires, fermetures, correspondances de galois. Bull Soc Math France 76:114–155 851
852
- Schmidt G, Ströhlein T (1985) On kernels of graphs and solutions of games: a synopsis based on relations and fixpoints. SIAM, J Algebraic Discrete Methods 6:54–65 853
854
- von Neumann J, Morgenstern O (1944) Theory of games and economic behaviour. Princeton University Press, Princeton 855
856

AUTHOR QUERY

- AQ1.** Missing citation for “Fig. 17.11” was inserted here. Please check if appropriate. Otherwise, please provide citation for “Fig. 17.11”. Note that the order of main citations of figures in the text must be sequential.

Uncorrected Proof

Chapter 18

On Confident Outrankings with Uncertain Criteria Significance Weights

1
2
3

Contents

18.1 Modelling Uncertain Criteria Significance Weights	251	5
18.2 Bipolar-Valued Likelihood of Outranking Situations	253	6
18.3 Confidence level Of Outranking Digraphs	255	7

Abstract When modelling preferences following the outranking approach, the signs of the majority margins do sharply distribute validation and invalidation of pairwise outranking situations. How can we be confident in the resulting outranking digraph, when we acknowledge the usual imprecise knowledge of criteria significance weights coupled with small majority margins? In this chapter we propose to link the qualifying significance majority with a required $\alpha\%$ -confidence level. We model therefore the significance weights as random variables following more or less widespread distributions around an average significance value that corresponds to the given deterministic weight. As the bipolar-valued random credibility of an outranking statement hence results from the simple sum of positive or negative independent random variables, we may apply the Central Limit Theorem (CLT) for computing the bipolar likelihood that the expected majority margin will indeed be positive and, respectively, negative.

AQ1

18.1 Modelling Uncertain Criteria Significance Weights

21

Let us consider the significance weights of a family F of m criteria to be independent random variables W_j , distributing the potential significance weight of each criterion $j = 1, \dots, m$ around a mean value $E(W_j)$ with variance $V(W_j)$.

Choosing a specific stochastic model of uncertainty is usually application specific. In the limited scope of this chapter, we will illustrate the consequence of this design decision on the resulting outranking modelling with four slightly different models for taking into account the uncertainty with which we know the

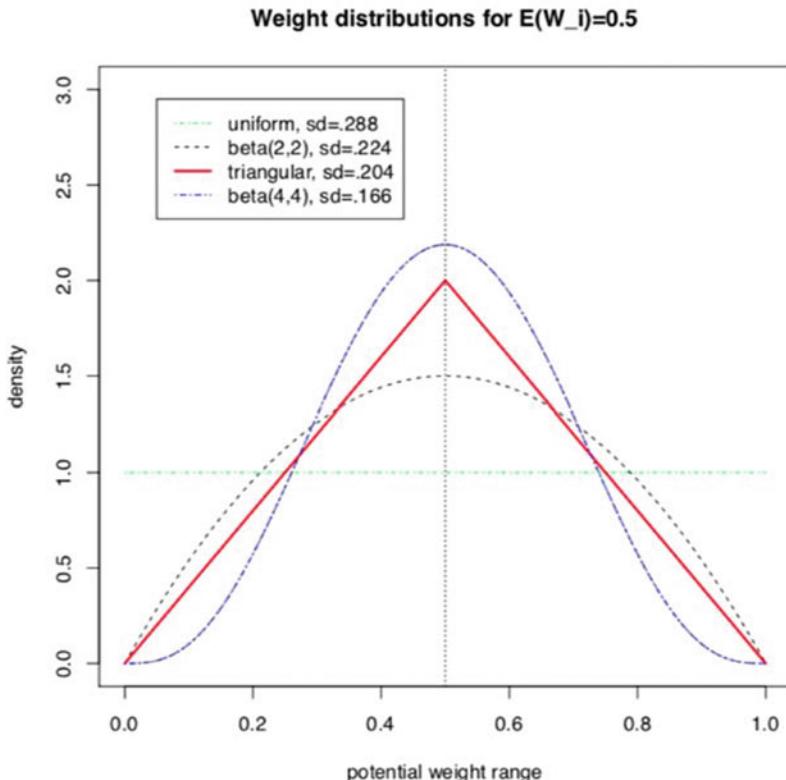


Fig. 18.1 Four models of uncertain criteria significance weights

numerical significance weights: *uniform*, *triangular*, and two models of *Beta laws*, one more widespread and, the other, more concentrated.

When considering, for instance, that the potential range of a significance weight is distributed between 0 and two times its mean value, we obtain the following random variates (Bisdorff 2020):

- A continuous *uniform* distribution on the range 0 to $2E(W_j)$. Thus, $W_j \sim U(0, 2E(W_j))$ and $V(W_j) = 1/3(E(W_j))^2$.
- A *symmetric beta* distribution with, for instance, parameters $\alpha = 2$ and $\beta = 2$. Thus, $W_i \sim Beta(2, 2) \times 2E(W_j)$ and $V(W_j) = 1/5(E(W_j))^2$.
- A *symmetric triangular* distribution on the same range with mode $E(W_j)$. Thus $W_j \sim Tr(0, 2E(W_j), E(W_j))$ with $V(W_j) = 1/6(E(W_j))^2$.
- A *narrower beta* distribution with, for instance, parameters $\alpha = 4$ and $\beta = 4$. Thus $W_j \sim Beta(4, 4) \times 2E(W_j)$ and $V(W_j) = 1/9(E(W_j))^2$.

It is worthwhile noticing in Fig. 18.1 that these four uncertainty models all admit the same expected value, $E(W_j)$, however, with a respective variance which goes decreasing from $1/3$ to $1/9$ of the square of $E(W_j)$ (Bisdorff 2014).

18.2 Bipolar-Valued Likelihood of Outranking Situations

Let $A = \{x, y, z, \dots\}$ be a finite set of n potential decision actions, evaluated on $F = \{1, \dots, m\}$ —a finite and coherent family of m performance criteria. On each criterion $j \in F$, the decision actions are evaluated on a real performance scale $[0; M_j]$, supporting an upper-closed indifference threshold ind_j and a lower-closed preference threshold pr_j such that $0 \leq ind_j < pr_j \leq M_j$. The marginal performance of object x on criterion j is denoted x_j . Each criterion j is thus characterising a marginal double threshold order \geq_j on A :

$$r(x \geq_j y) = \begin{cases} +1 & \text{if } x_j - y_j \geq -ind_j, \\ -1 & \text{if } x_j - y_j \leq -pr_j, \\ 0 & \text{otherwise.} \end{cases} \quad (18.1)$$

Semantics of the marginal bipolar-valued characteristic function:

- $+1$ signifies x is at least as well evaluated as y on criterion j .
- -1 signifies that x is not at least as well evaluated as y on criterion j .
- 0 signifies that it is unclear whether, on criterion j , x is at least as well evaluated as y (Fig. 18.2).

AQ2

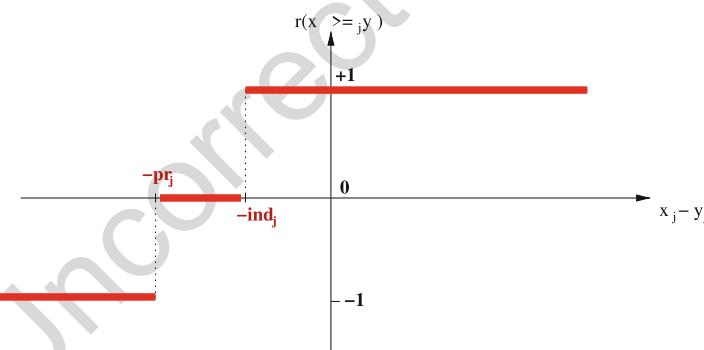


Fig. 18.2 Bipolar-valued outranking characteristic function

Each criterion j in F contributes the random significance W_j of his ‘*at least as well evaluated as*’ characteristic $r(x \geq_j y)$ to the global characteristic $\tilde{r}(x \geq y)$ in the following way:

$$\tilde{r}(x \geq y) = \sum_{j \in F} W_j \times r(x \geq_j y). \quad (18.2)$$

Thus, $\tilde{r}(x \geq y)$ becomes a simple sum of positive or negative independent random variables with known means and variances where $\tilde{r}(x \geq y) > 0$ signifies x is globally at least as well evaluated as y , $\tilde{r}(x \geq y) < 0$ signifies that x is not globally at least as well evaluated as y , and $\tilde{r}(x \geq y) = 0$ signifies that it is unclear whether x is globally at least as well evaluated as y .

From the *Central Limit Theorem* (CLT), we know that such a sum of random variables leads, with m getting large, to a Gaussian distribution Y with

$$E(Y) = \sum_{j \in F} (E(W_j) \times r(x \geq_j y)), \text{ and} \quad (18.3)$$

$$V(Y) = \sum_{j \in F} (V(W_j) \times |r(x \geq_j y)|). \quad (18.4)$$

And the *likelihood of validation*, respectively, *invalidation*, of an ‘*at least as well evaluated as*’ situation, denoted $lh(x \geq y)$, may hence be assessed by the probability $P(Y > 0) = 1.0 - P(Y \leq 0)$ that Y takes a positive value, respectively, $P(Y < 0)$ takes a negative value. In the bipolar-valued case here, we can judiciously make usage of the standard Gaussian *error function*, i.e. the bipolar $2P(Z) - 1.0$ version of the standard Gaussian $P(Z)$ probability distribution function (Press et al. 2007):

$$lh(x \geq y) = -\text{erf}\left(\frac{1}{\sqrt{2}} \frac{-E(Y)}{\sqrt{V(Y)}}\right). \quad (18.5)$$

The range of the bipolar-valued $lh(x \geq y)$ hence becomes $[-1.0; +1.0]$, and $-lh(x \geq y) = lh(x \not\geq y)$, i.e. a *negative likelihood* represents the likelihood of the correspondent *negated ‘*at least as well evaluated as*’* situation. A likelihood of $+1.0$ (respectively, -1.0) means the corresponding preferential situation appears *certainly validated* (respectively, *invalidated*).

Example 18.1 Let x and y be evaluated with respect to 7 equi-significant criteria; four criteria positively support that x is *at least as well assessed as* y and three criteria support that x is *not at least as well evaluated as* y . Suppose $E(W_j) = w$ for $j = 1, \dots, 7$ and $W_j \sim Tr(0, 2w, w)$ for $j = 1, \dots, 7$. The expected value of the global ‘*at least as well evaluated as*’ characteristic value becomes $E(\tilde{r}(x \geq y)) = 4w - 3w = w$ with a variance $V(\tilde{r}(x \geq y)) = 7\frac{1}{6}w^2$.

If $w = 1$, $E(\tilde{r}(x \geq y)) = 1$ and $sd(\tilde{r}(x \geq y)) = 1.08$. By the CLT, the bipolar likelihood of the *at least as well evaluated as* situation becomes $lh(x \geq y) = 0.66$,

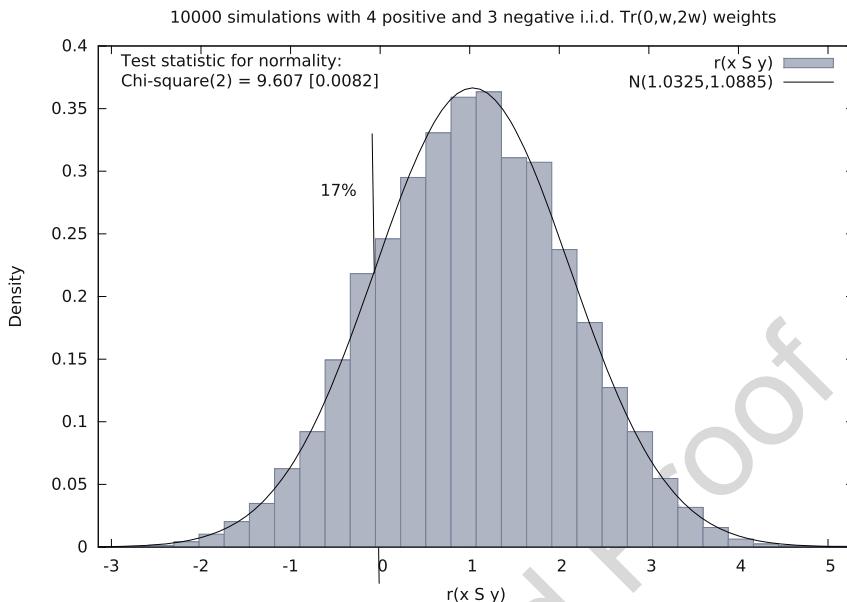


Fig. 18.3 Distribution of 10 000 random outranking characteristic values

which corresponds to a global support of $(0.66 + 1.0)/2 = 83\%$ of the criteria 88
significance weights. 89

A Monte Carlo simulation with 10 000 runs empirically confirms the effective 90
convergence to a Gaussian (see Fig. 18.3 realised with gretl.¹) 91

Indeed, $\tilde{r}(x \geq y) \rightsquigarrow Y = \mathcal{N}(1.03, 1.089)$, with an empirical probability of 92
observing a negative majority margin of about 17%. 93

18.3 Confidence level Of Outranking Digraphs

Definition 18.1 (Confident Outranking Situation) Following the classical outranking approach (see (Bisdorff 2013)), we say, from an epistemic perspective, that 95
decision action x outranks decision action y at confidence level $\alpha\%$, when: 96

- An expected majority of criteria validates, at confidence level $\alpha\%$ or higher, a 98
global ‘at least as well evaluated as’ situation between x and y . 99
- No considerably less performing is observed on a discordant criterion. 100

¹ The GNU Regression, Econometrics and Time-Series Library, <http://gretl.sourceforge.net/>.

Dually, decision action x *does not outrank* decision action y at confidence level $\alpha\%$, 101
 when: 102

- An expected minority only of criteria at confidence level $-\alpha\%$ or lesser validates a global ‘*at least as well evaluated as*’ situation between x and y . 103
 - No considerably better performing situation is observed on a discordant criterion. 104
 - No considerably better performing situation is observed on a discordant criterion. 105

Otherwise, the situation is indeterminate.

Time for a coded example

Let us consider the following random performance tableau:

```

1 >>> from randomPerfTabs import RandomPerformanceTableau
2 >>> t = RandomPerformanceTableau(
3 ...         numberOfActions=7,\n4 ...         numberOfCriteria=7,seed=100)
5 >>> t.showPerformanceTableau(Transposed=True)
6     **** performance tableau ****
7 criteria | weights | 'a1'  'a2'  'a3'  'a4'  'a5'  'a6'  'a7'
8 -----
9   'g1'    | 1      | 15.17 44.51 57.87 58.00 24.22 29.10 96.58
10  'g2'   | 1      | 82.29 43.90  NA     35.84 29.12 34.79 62.22
11  'g3'   | 1      | 44.23 19.10 27.73 41.46 22.41 21.52 56.90
12  'g4'   | 1      | 46.37 16.22 21.53 51.16 77.01 39.35 32.06
13  'g5'   | 1      | 47.67 14.81 79.70 67.48  NA     90.72 80.16
14  'g6'   | 1      | 69.62 45.49 22.03 33.83 31.83  NA     48.80
15  'g7'   | 1      | 82.88 41.66 12.82 21.92 75.74 15.45  6.05

```

For the corresponding confident outranking digraph, we assume uncertain criteria 124 significance weights of *triangular* distribution and require a confidence level of $\alpha =$ 125 90%. The `ConfidentBipolarOutrankingDigraph` class provides such a 126 construction. 127

Listing 18.1 Computing a 90% confident outranking digraph

```

1 >>> from outrankingDigraphs import\
2 ...     ConfidentBipolarOutrankingDigraph
3 >>> g90 = ConfidentBipolarOutrankingDigraph(t,\
4 ...     distribution ='triangular',confidence=90)
5 >>> g90
6 *----- Object instance description -----*
7 Instance class      : ConfidentBipolarOutrankingDigraph
8 Instance name       : rel_randomperftab_CLT
9 Actions             : 7
10 Criteria            : 7
11 Size                : 15
12 Uncertainty model  : triangular(a=0,b=2w)
13 Likelihood domain   : [-1.0;+1.0]
14 Confidence level    : 0.80 (90.0%)
15 Confident majority  : 0.14 (57.1%)
16 Determinateness (%) : 62.07
17 Valuation domain    : [-1.00;1.00]
18 Attributes          : ['name', 'bipolarConfidenceLevel',
19 ...                         'distribution', 'betaParameter', 'actions',
20 ...                         'order', 'valuationdomain', 'criteria',
21 ...                         'evaluation', 'concordanceRelation',
22 ...                         'concordanceRelation', 'concordanceRelation']
```

```

22             'vetos', 'negativeVetos',
23             'largePerformanceDifferencesCount',
24             'likelihoods', 'confidenceCutLevel',
25             'relation', 'gamma', 'notGamma'] 149
150
151
152

```

The resulting 90%-confident expected outranking relation is shown below.

153

Listing 18.2 90%-confident outranking relation with triangular distributed significance weights

```

1 >>> g90.showRelationTable(LikelihoodDenotation=True) 154
2 * ---- Outranking Relation Table ---- 155
3 r/(lh) | 'a1'   'a2'   'a3'   'a4'   'a5'   'a6'   'a7' 156
4 -----|----- 157
5 'a1' | +0.00  +0.71  +0.29  +0.29  +0.29  +0.29  +0.00 158
6 | ( - )  (+1.00) (+0.95) (+0.95) (+0.95) (+0.95) (+0.65) 159
7 'a2' | -0.71  +0.00  -0.29  +0.00  +0.00  +0.29  -0.57 160
8 | (-1.00) ( - )  (-0.95) (-0.65) (+0.73) (+0.95) (-1.00) 161
9 'a3' | -0.29  +0.29  +0.00  -0.29  +0.00  +0.00  -0.29 162
10 | (-0.95) (+0.95) ( - )  (-0.95) (-0.73) (-0.00) (-0.95) 163
11 'a4' | +0.00  +0.00  +0.57  +0.00  +0.29  +0.57  -0.43 164
12 | (-0.00) (+0.65) (+1.00) ( - )  (+0.95) (+1.00) (-0.99) 165
13 'a5' | -0.29  +0.00  +0.00  +0.00  +0.00  +0.29  -0.29 166
14 | (-0.95) (-0.00) (+0.73) (-0.00) ( - )  (+0.99) (-0.95) 167
15 'a6' | -0.29  +0.00  +0.00  -0.29  +0.00  +0.00  +0.00 168
16 | (-0.95) (-0.00) (+0.73) (-0.95) (+0.73) ( - )  (-0.00) 169
17 'a7' | +0.00  +0.71  +0.57  +0.43  +0.29  +0.00  +0.00 170
18 | (-0.65) (+1.00) (+1.00) (+0.99) (+0.95) (-0.00) ( - ) 171
19 Valuation domain : [-1.000; +1.000] 172
20 Uncertainty model : triangular(a=2.0,b=2.0) 173
21 Likelihood domain : [-1.0;+1.0] 174
22 Confidence level : 0.80 (90.0%) 175
23 Confident majority : 0.14 (57.1%) 176
24 Determinateness : 0.24 (62.1%) 177

```

The (lh) figures, indicated in Listing 18.2 above, correspond to bipolar likelihoods and the required bipolar confidence level equals $(0.90 + 1.0)/2 = 0.80$ (see Line 22 above). Action a_1 thus confidently outranks all other actions, except a_7 where the actual likelihood $(+0.65)$ is lower than the required one $(+0.80)$, and we furthermore observe a considerable counter-performance on criterion g_1 .

Notice also the lack of confidence in the outranking situations we observe between action a_2 and actions a_4 and a_5 . In the deterministic case, we would have $r(a_2 \geq a_4) = -0.143$ and $r(a_2 \geq a_5) = +0.143$. All outranking situations with a characteristic value lower than or equal to $abs(0.143)$, i.e. a majority support of $1.143/2 = 57.1\%$ and less, appear indeed to be *not confident* at α level 90% (see Line 23 above).

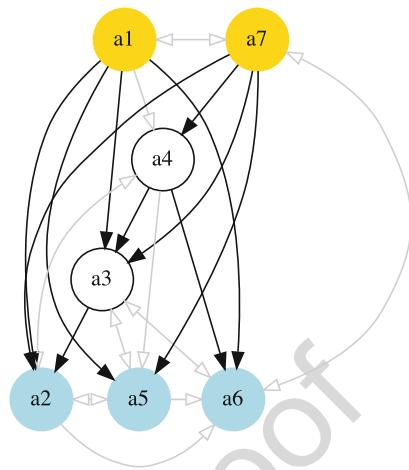
Figure 18.4 on the next page shows the corresponding strict 90%-confident outranking digraph, oriented by its initial and terminal strict prekernels.

```

1 >>> gcd90 = ~ (-g90) 191
2 >>> gcd90.showPreKernels() 192
3 *--- Computing preKernels ---* 193
4 Dominant preKernels : 194
5   ['a1', 'a7'] 195
6     independence : 0.0 196
7     dominance : 0.2857 197
8     absorbency : -0.7143 198

```

Fig. 18.4 90%-confident strict outranking digraph oriented by its initial and terminal prekernels



Digraph3 (graphviz), R. Bisdorff, 2020

```

9      covering      :  0.800
10     Absorbent preKernels :
11     ['a2', 'a5', 'a6']
12     independence   :  0.0
13     dominance     : -0.2857
14     absorbency    :  0.2857
15     covered        :  0.583
16 >>> gcd90.exportGraphViz(fileName='confidentOutranking',
17 ...           firstChoice=['a1', 'a7'], \
18 ...           lastChoice=['a2', 'a5', 'a6'])
19 *---- exporting a dot file for GraphViz tools ----*
20 Exporting to confidentOutranking.dot
21 dot -Grankdir=BT -Tpng confidentOutranking.dot\
22           -o confidentOutranking.png

```

Now, what becomes this 90%-confident outranking digraph when we require a stronger confidence level, say 99%?

Listing 18.3 99%-confident outranking relation

```

1 >>> g99 = ConfidentBipolarOutrankingDigraph(t,\ 215
2 ...     distribution='trinangular',confidence=99) 216
3 >>> g99.showRelationTable() 217
4 *---- Outranking Relation Table ---- 218
5 r/(lh) | 'a1'   'a2'   'a3'   'a4'   'a5'   'a6'   'a7' 219
6 -----|----- 220
7 'a1' | +0.00  +0.71  +0.00  +0.00  +0.00  +0.00  +0.00 221
8 | ( - )  (+1.00) (+0.95) (+0.95) (+0.95) (+0.95) (+0.65) 222
9 'a2' | -0.71  +0.00  +0.00  +0.00  +0.00  +0.00  -0.57 223
10 | (-1.00) ( - )  (-0.95) (-0.65) (+0.73) (+0.95) (-1.00) 224
11 'a3' | +0.00  +0.00  +0.00  +0.00  +0.00  +0.00  +0.00 225
12 | (-0.95) (+0.95) ( - )  (-0.95) (-0.73) (-0.00) (-0.95) 226
13 'a4' | +0.00  +0.00  +0.57  +0.00  +0.00  +0.57  -0.43 227
14 | (-0.00) (+0.65) (+1.00) ( - )  (+0.95) (+1.00) (-0.99) 228
15 'a5' | +0.00  +0.00  +0.00  +0.00  +0.00  +0.29  +0.00 229

```

16	'a6'	(-0.95)	(-0.00)	(+0.73)	(-0.00)	(-)	(+0.99)	(-0.95)	230	
17		+0.00	+0.00	+0.00	+0.00	+0.00	+0.00	+0.00	231	
18		(-0.95)	(-0.00)	(+0.73)	(-0.95)	(+0.73)	(-)	(-0.00)	232	
19	'a7'	+0.00	+0.71	+0.57	+0.43	+0.00	+0.00	+0.00	233	
20		(-0.65)	(+1.00)	(+1.00)	(+0.99)	(+0.95)	(-0.00)	(-)	234	
21	Valuation domain	: [-1.000; +1.000]								235
22	Uncertainty model	: triangular(a=0.0,b=2.0)								236
23	Likelihood domain	: [-1.0;+1.0]								237
24	Confidence level	: 0.98 (99.0%)								238
25	Confident majority	: 0.29 (64.3%)								239
26	Determinateness	: 0.13 (56.6%)								240

At 99% confidence, the minimal required significance majority support amounts to 64.3% (see Listing 18.3, Line 25 above). As a result, most outranking situations do not get anymore validated, like the outranking situations between action a1 and actions a3, a4, a5, and a6 (see Line 5 above). The overall epistemic determination of the digraph consequently drops from 62.1% to 56.6% (see Line 26).

Finally, what becomes the previous 90%-confident outranking digraph when the uncertainty concerning the criteria significance weights is modelled with a larger variance, like *uniform* variates (see Line 2 below).

Listing 18.4 90%-confident outranking digraph with uniform variates

1	>>> gu90 = ConfidentBipolarOutrankingDigraph(t, \	249
2	... confidence=90,distribution='uniform')	250
3	>>> gu90.showRelationTable()	251
4	* ----- Outranking Relation Table -----	252
5	r/(lh) 'a1' 'a2' 'a3' 'a4' 'a5' 'a6' 'a7'	253
6	-----	254
7	'a1' +0.00 +0.71 +0.29 +0.29 +0.29 +0.29 +0.00	255
8	(-) (+1.00) (+0.84) (+0.84) (+0.84) (+0.84) (+0.49)	256
9	'a2' -0.71 +0.00 -0.29 +0.00 +0.00 +0.29 -0.57	257
10	(-1.00) (-) (-0.84) (-0.49) (+0.56) (+0.84) (-1.00)	258
11	'a3' -0.29 +0.29 +0.00 -0.29 +0.00 +0.00 -0.29	259
12	(-0.84) (+0.84) (-) (-0.84) (-0.56) (-0.00) (-0.84)	260
13	'a4' +0.00 +0.00 +0.57 +0.00 +0.29 +0.57 -0.43	261
14	(-0.00) (+0.49) (+1.00) (-) (+0.84) (+1.00) (-0.95)	262
15	'a5' -0.29 +0.00 +0.00 +0.00 +0.00 +0.29 -0.29	263
16	(-0.84) (-0.00) (+0.56) (-0.00) (-) (+0.92) (-0.84)	264
17	'a6' -0.29 +0.00 +0.00 -0.29 +0.00 +0.00 +0.00	265
18	(-0.84) (-0.00) (+0.56) (-0.84) (+0.56) (-) (-0.00)	266
19	'a7' +0.00 +0.71 +0.57 +0.43 +0.29 +0.00 +0.00	267
20	(-0.49) (+1.00) (+1.00) (+0.95) (+0.84) (-0.00) (-)	268
21	Valuation domain : [-1.000; +1.000]	269
22	Uncertainty model : uniform(a=0.0,b=2.0)	270
23	Likelihood domain : [-1.0;+1.0]	271
24	Confidence level : 0.80 (90.0%)	272
25	Confident majority : 0.14 (57.1%)	273
26	Determinateness : 0.24 (62.1%)	274

Despite lower likelihood values (see the g90 digraph in Listing 18.2 on page 257), we keep here the same confident majority level of 57.1% (Line 25) and, hence, also the same 90%-confident outranking digraph.

For concluding, it is worthwhile noticing again that it is the *neutral* value of the bipolar-valued epistemic logic that allows to easily handle $\alpha\%$ confidence or not of outranking situations when confronted with uncertain criteria significance weights. Remarkable furthermore is the usage, the standard Gaussian error function (erf) provides by delivering *signed* likelihood values immediately concerning either

a positive relational statement or when negative its negated version (Press et al. 283
2007). 284

Chapter 19 analyses the robustness of the outranking digraph when the criteria 285
significance weights faithfully indicate solely an order of importance. 286

References

287

- Bisdorff R (2013) On polarizing outranking relations with large performance differences. Journal 288
of Multi-Criteria Decision Analysis, Wiley 20:3–12, <http://hdl.handle.net/10993/245> 289
- Bisdorff R (2014) On confident outrankings with multiple criteria of uncertain significance. In: 290
Mousseau V, Pirlot M (eds) DAP'2014 From Multiple Criteria Decision Aid to Preference 291
Learning, Ecole Centrale de Paris, pp 1–6. <http://hdl.handle.net/10993/23910> 292
- Bisdorff R (2020) Lecture 3: continuous random variables. In: Lectures of the computational 293
statistics course. University of Luxembourg. <http://hdl.handle.net/10993/37870> 294
- Press W, Teukolsky S, Vetterling W, Flannery B (2007) Numerical recipes: the art of scientific 295
computing, 3rd edn., Special Functions, Section 6.2. Cambridge University Press, Cambridge, 296
pp 209–214 297

AUTHOR QUERIES

- AQ1.** Please check if the sentence “As the bipolar-valued random ...” is fine as edited and amend if required.
- AQ2.** Missing citation for “Fig. 18.2” was inserted here. Please check if appropriate. Otherwise, please provide citation for “Fig. 18.2”. Note that the order of main citations of figures in the text must be sequential.
- AQ3.** Please check if the sentence “And the *likelihood of validation* ...” is fine as edited and amend if required.

Uncorrected Proof

Chapter 19	1
Robustness Analysis of Outranking	2
Digraphs	3

Contents	4
19.1 Cardinal or Ordinal Criteria Significance Weights?	261 5
19.2 Qualifying the Stability of Outranking Situations	263 6
19.3 Computing the Stability Denotation of Outranking Situations	267 7
19.4 Robust Bipolar-Valued Outranking Digraphs	269 8
19.5 Characterising Unopposed Multiobjective Outranking Situations	272 10
19.6 Computing Pareto Efficient Multiobjective Choices	275 11

Abstract The required cardinal significance weights of the performance criteria represent the ‘Achilles’ heel of the outranking approach. Rarely will indeed a decision maker be cognitively competent for suggesting precise decimal-valued criteria significance weights. More often, the decision problem will involve more or less equally important decision objectives with more or less equi-significant criteria per objective. In this chapter, we study the robustness of the outranking digraph when the criteria significance weights faithfully indicate solely an order of importance. 12
13
14
15
16
17
18
19

19.1 Cardinal or Ordinal Criteria Significance Weights? 20

A random example of such a decision problem with equally important decision objectives and equi-significant criteria may be generated with the Random3-ObjectivesPerformanceTableau class. 21
22
23

Listing 19.1 Generate a random 3-objectives performance tableau

```

1 >>> from randomPerfTabs import \
2 ...             Random3ObjectivesPerformanceTableau
3 >>> pt = Random3ObjectivesPerformanceTableau(\
4 ...             numberOfActions=7, \
5 ...             numberOfCriteria=9, seed=102)
6 >>> pt

```

```

7  *----- PerformanceTableau instance description -----* 30
8  Instance class      : Random3ObjectivesPerformanceTableau 31
9  Seed                : 102 32
10 Instance name       : random3ObjectivesPerfTab 33
11 Actions              : 7 34
12 Objectives           : 3 35
13 Criteria             : 9 36
14 NA proportion (%) : 0.0 37
15 Attributes           : ['name', 'valueDigits', 'BigData', 38
16           'OrdinalScales', 'missingDataProbability', 39
17           'negativeWeightProbability', 'randomSeed', 40
18           'sumWeights', 'valuationPrecision', 41
19           'commonScale', 'objectiveSupportingTypes', 42
20           'actions', 'objectives', 'criteriaWeightMode', 43
21           'criteria', 'evaluation', 'weightPreorder'] 44
22 >>> pt.showObjectives() 45
23 *----- show objectives -----* 46
24 Eco: Economical aspect 47
25   ec1 criterion of objective Eco 8 48
26   ec4 criterion of objective Eco 8 49
27   ec8 criterion of objective Eco 8 50
28   Total weight: 24.00 (3 criteria) 51
29 Soc: Societal aspect 52
30   so2 criterion of objective Soc 12 53
31   so7 criterion of objective Soc 12 54
32   Total weight: 24.00 (2 criteria) 55
33 Env: Environmental aspect 56
34   en3 criterion of objective Env 6 57
35   en5 criterion of objective Env 6 58
36   en6 criterion of objective Env 6 59
37   en9 criterion of objective Env 6 60
38   Total weight: 24.00 (4 criteria) 61

```

In Listing 19.1 above, one may notice a performance tableau `pt` describing seven decision alternatives that are assessed with respect to three equally important decision objectives concerning:

1. An *economical* aspect with a coalition of three performance criteria of significance weight 8 (Line 24)
2. A *societal* aspect with a coalition of two performance criteria of significance weight 12 (Line 29)
3. An *environmental* aspect with a coalition of four performance criteria of significance weight 6 (Line 33)

The question we tackle in this chapter is the following: how *dependent* on the actual values of the significance weights appears to be the corresponding bipolar-valued outranking digraph? In the previous chapter, Chap. 18, we assumed that the criteria significance weights were random variables. Here, we shall assume that we know for sure only the preordering of the significance weights.

In the random performance tableau `pt`, we observe three increasing weight equivalence classes.

Listing 19.2 The significance weights preorder

```

1 >>> pt.showWeightPreorder()
2      ['en3', 'en5', 'en6', 'en9'] (6) <
3      ['ec1', 'ec4', 'ec8'] (8) <
4      ['so2', 'so7'] (12)

```

78
79
80
81

How robust will appear now the validated outranking and outranked situations when allowing all possible significance weights that are compatible with the weights preorder shown above (Bisdorff et al. 2009, 2014)?

82
83
84

19.2 Qualifying the Stability of Outranking Situations

85

Let us construct the bipolar-valued outranking digraph corresponding to the random 3-objectives performance tableau pt .

86
87**Listing 19.3** Example bipolar outranking digraph

```

1 >>> from outrankingDigraphs import BipolarOutrankingDigraph
2 >>> g = BipolarOutrankingDigraph(pt)
3 >>> g.showRelationTable()
4 * ----- Relation Table -----
5   r(>=) | 'p1'   'p2'   'p3'   'p4'   'p5'   'p6'   'p7'
6   -----|-----
7   'p1' | +1.00  -0.42  +0.00  -0.69  +0.39  +0.11  -0.06
8   'p2' | +0.58  +1.00  +0.83  +0.00  +0.58  +0.58  +0.58
9   'p3' | +0.25  -0.33  +1.00  +0.00  +0.50  +1.00  +0.25
10  'p4' | +0.78  +0.00  +0.61  +1.00  +1.00  +1.00  +0.67
11  'p5' | -0.11  -0.50  -0.25  -0.89  +1.00  +0.11  -0.14
12  'p6' | +0.22  -0.42  +0.00  -1.00  +0.17  +1.00  -0.11
13  'p7' | +0.22  -0.50  +0.17  -0.06  +0.78  +0.42  +1.00

```

88
89
90
91
92
93
94
95
96
97
98
99
100

In Listing 19.3, Lines 7–13, we notice on the principal diagonal of the relation table the certainly validated reflexive terms +1.00; they are trivially independent of any significance weights. Now, we know for sure that *unanimous* outranking situations are also completely independent of the significance weights. And, all outranking situations that are supported by a majority significance in each coalition of equi-significant criteria are as well independent of the actual importance we attach to each individual criteria coalition. We are furthermore able to effectively test if an outranking situation remains valid with all potential significance weights that are compatible with the given preordering shown in Listing 19.2 on the facing page (Bisdorff 2014). Mind that usually one obtains more or less numerous outranking situations that are in fact *dependent* on the precise cardinal significance weights given to the criteria.

101
102
103
104
105
106
107
108
109
110
111
112

Such a *stability denotation* of outranking situations can be inspected using the `StabilityDenotation=True` flag with the `showRelationTable()` method.

113
114
115

Listing 19.4 Bipolar-valued outranking relation table with stability denotation

>>> g.showRelationTable (StabilityDenotation=True)								116	
* ---- Relation Table ----								117	
r/(stab)	'p1'	'p2'	'p3'	'p4'	'p5'	'p6'	'p7'	118	
-----	-----	-----	-----	-----	-----	-----	-----	119	
	'p1'	+1.00 (+4)	-0.42 (-2)	+0.00 (+0)	-0.69 (-3)	+0.39 (+2)	+0.11 (+2)	-0.06 (-1)	120
	'p2'	+0.58 (+2)	+1.00 (+4)	+0.83 (+3)	0.00 (+2)	+0.58 (+2)	+0.58 (+2)	+0.58 (+2)	122
	'p3'	+0.25 (+2)	-0.33 (-2)	+1.00 (+4)	0.00 (0)	+0.50 (+2)	+1.00 (+2)	+0.25 (+1)	124
	'p4'	+0.78 (+3)	0.00 (-1)	+0.61 (+3)	+1.00 (+4)	+1.00 (+4)	+1.00 (+4)	+0.67 (+2)	126
	'p5'	-0.11 (-2)	-0.50 (-2)	-0.25 (-2)	-0.89 (-3)	+1.00 (+4)	+0.11 (+2)	-0.14 (-2)	128
	'p6'	+0.22 (+2)	-0.42 (-2)	0.00 (+1)	-1.00 (-2)	+0.17 (+2)	+1.00 (+4)	-0.11 (-2)	130
	'p7'	+0.22 (+2)	-0.50 (-2)	+0.17 (+1)	-0.06 (-1)	+0.78 (+3)	+0.42 (+2)	+1.00 (+4)	132

In Listing 19.4 above, we may hence distinguish the following bipolar-valued stability levels:

- ±4: *unanimous* outranking, respectively, outranked, situation. The pairwise trivial reflexive outrankings, for instance, all show this stability level.
- ±3: validated outranking, respectively, outranked, situation in *each* coalition of equi-significant criteria. This is, for instance, the case for the outranking situation observed between alternatives p1 and p4 (see Lines 6 and 12).
- ±2: validated outranking, respectively, outranked situation with *all* potential significance weights that are *compatible* with the given significance preorder (see Listing 19.2 on page 262). This is the case when comparing alternatives p1 and p2 (see Lines 6 and 8).
- ±1: validated outranking, respectively, outranked situation with the *precisely given decimal* significance weights, a situation we may observe between alternatives p3 and p7 (see Lines 10 and 18).
- 0: indeterminate outranking situation, like the one between alternatives p1 and p3 (see Lines 6 and 10).

It is worthwhile noticing that, in the one limit case where all performance criteria appear equi-significant – there is given a single equivalence class containing all the criteria significance weights – one may only distinguish stability levels ±4 and ±3. In the other limit case, when all performance criteria admit different significance weights, the significance weights may be linearly ordered without ties and no stability level ±3 can be observed.

As mentioned above, all reflexive comparisons trivially confirm an unanimous outranking situation: all decision alternatives are indeed always “*at least as well evaluated as*” themselves. But there appear also two non-reflexive unanimous outranking situations: when comparing, for instance, alternative p4 with alternatives

p5 and p6 (see Listing 19.4 on the facing page, Lines 14 and 16). Let us inspect 160
 the details of how alternatives p4 and p5 are compared. 161

AQ2

```

1 >>> g.showPairwiseComparison('p4','p5') 162
2 *----- pairwise comparison -----* 163
3 Comparing actions : (p4, p5) 164
4 crit. wght. g(x) g(y) diff | ind pref r() 165
5 ec1 8.00 85.19 46.75 +38.44 | 5.00 10.00 +8.00 166
6 ec4 8.00 72.26 8.96 +63.30 | 5.00 10.00 +8.00 167
7 ec8 8.00 44.62 35.91 +8.71 | 5.00 10.00 +8.00 168
8 en3 6.00 80.81 31.05 +49.76 | 5.00 10.00 +6.00 169
9 en5 6.00 49.69 29.52 +20.17 | 5.00 10.00 +6.00 170
10 en6 6.00 66.21 31.22 +34.99 | 5.00 10.00 +6.00 171
11 en9 6.00 50.92 9.83 +41.09 | 5.00 10.00 +6.00 172
12 so2 12.00 49.05 12.36 +36.69 | 5.00 10.00 +12.00 173
13 so7 12.00 55.57 44.92 +10.65 | 5.00 10.00 +12.00 174
14 Valuation in range: -72.00 to +72.00; ----- 175
15 global concordance: +72.00 176

```

Alternative p4 is indeed unanimously “*at least as well evaluated as*” alternative 177
 p5 and $r(p4 \succsim p5) = 72/72 = +1.00$ (see Listing 19.4 on the preceding page, 178
 Line 11). The converse comparison does not, however, reveal such an unanimous 179
 outranked situation. 180

```

1 >>> g.showPairwiseComparison('p5','p4') 181
2 *----- pairwise comparison -----* 182
3 Comparing actions : (p5, p4) 183
4 crit. wght. g(x) g(y) diff | ind pref r() 184
5 ec1 8.00 46.75 85.19 -38.44 | 5.00 10.00 -8.00 185
6 ec4 8.00 8.96 72.26 -63.30 | 5.00 10.00 -8.00 186
7 ec8 8.00 35.91 44.62 -8.71 | 5.00 10.00 +0.00 187
8 en3 6.00 31.05 80.81 -49.76 | 5.00 10.00 -6.00 188
9 en5 6.00 29.52 49.69 -20.17 | 5.00 10.00 -6.00 189
10 en6 6.00 31.22 66.21 -34.99 | 5.00 10.00 -6.00 190
11 en9 6.00 9.83 50.92 -41.09 | 5.00 10.00 -6.00 191
12 so2 12.00 12.36 49.05 -36.69 | 5.00 10.00 -12.00 192
13 so7 12.00 44.92 55.57 -10.65 | 5.00 10.00 -12.00 193
14 Valuation in range: -72.00 to +72.00; ----- 194
15 global concordance: -64.00 195

```

The converse comparison only qualifies at stability level -3 (see Listing 19.4 196
 on the facing page, Line 13); $r(p5 \succsim p4) = -64/72 = -0.89$. On criterion 197
 ec8, we observe in fact a small negative performance difference of -8.71 (see 198
 Line 7 above) which is effectively below the supposed preference discrimination 199
 threshold of 10.00. Yet, the outranked situation is supported by a majority of criteria 200
 in each decision objective. Hence, the reported preferential situation is completely 201
 independent of any chosen significance weights. 202

Let us now consider a comparison, like the one between alternatives p2 and p1, 203
 that is qualified at stability level $+2$, respectively, -2 . 204

Listing 19.5 Comparison of alternatives p2 and p1

```

1 >>> g.showPairwiseOutrankings('p2','p1') 205
2 *----- pairwise comparison -----* 206
3 Comparing actions : (p2, p1) 207
4 crit. wght. g(x) g(y) diff | ind pref r() 208
5 ec1 8.00 89.77 38.11 +51.66 | 5.00 10.00 +8.00 209
6 ec4 8.00 86.00 22.65 +63.35 | 5.00 10.00 +8.00 210
7 ec8 8.00 89.43 77.02 +12.41 | 5.00 10.00 +8.00 211
8 en3 6.00 20.79 58.16 -37.37 | 5.00 10.00 -6.00 212
9 en5 6.00 23.83 31.40 -7.57 | 5.00 10.00 +0.00 213
10 en6 6.00 18.66 11.41 +7.25 | 5.00 10.00 +6.00 214
11 en9 6.00 26.65 44.37 -17.72 | 5.00 10.00 -6.00 215
12 so2 12.00 89.12 22.43 +66.69 | 5.00 10.00 +12.00 216
13 so7 12.00 84.73 28.41 +56.32 | 5.00 10.00 +12.00 217
14 Valuation in range: -72.00 to +72.00; ----- 218
15 global concordance: +42.00 219
16 *----- pairwise comparison -----* 220
17 Comparing actions : ('p1', 'p2') 221
18 crit. wght. g(x) g(y) diff | ind pref r() 222
19 ec1 8.00 38.11 89.77 -51.66 | 5.00 10.00 -8.00 223
20 ec4 8.00 22.65 86.00 -63.35 | 5.00 10.00 -8.00 224
21 ec8 8.00 77.02 89.43 -12.41 | 5.00 10.00 -8.00 225
22 en3 6.00 58.16 20.79 +37.37 | 5.00 10.00 +6.00 226
23 en5 6.00 31.40 23.83 +7.57 | 5.00 10.00 +6.00 227
24 en6 6.00 11.41 18.66 -7.25 | 5.00 10.00 +0.00 228
25 en9 6.00 44.37 26.65 +17.72 | 5.00 10.00 +6.00 229
26 so2 12.00 22.43 89.12 -66.69 | 5.00 10.00 -12.00 230
27 so7 12.00 28.41 84.73 -56.32 | 5.00 10.00 -12.00 231
28 Valuation in range: -72.00 to +72.00; ----- 232
29 global concordance: -30.00 233

```

In both comparisons, the evaluations observed with respect to the environmental decision objective are not validating with a significant majority the outranking, respectively, outranked, situation. Hence, the stability of the reported preferential situations is in fact dependent on choosing significance weights that are compatible with the given significance weights preorder (see Listing 19.2 on page 262).

Let us finally inspect a comparison that is only qualified at stability level +1, like the one between alternatives p7 and p3.

Listing 19.6 Comparison of alternatives p7 and p3

```

1 >>> g.showPairwiseOutrankings('p7','p3') 241
2 *----- pairwise comparison -----* 242
3 Comparing actions : ('p7', 'p3') 243
4 crit. wght. g(x) g(y) diff | ind pref r() 244
5 ec1 8.00 15.33 80.19 -64.86 | 5.00 10.00 -8.00 245
6 ec4 8.00 36.31 68.70 -32.39 | 5.00 10.00 -8.00 246
7 ec8 8.00 38.31 91.94 -53.63 | 5.00 10.00 -8.00 247
8 en3 6.00 30.70 46.78 -16.08 | 5.00 10.00 -6.00 248
9 en5 6.00 35.52 27.25 +8.27 | 5.00 10.00 +6.00 249
10 en6 6.00 69.71 1.65 +68.06 | 5.00 10.00 +6.00 250
11 en9 6.00 13.10 14.85 -1.75 | 5.00 10.00 +6.00 251

```

12	so2	12.00	68.06	58.85	+9.21		5.00	10.00	+12.00	252
13	so7	12.00	58.45	15.49	+42.96		5.00	10.00	+12.00	253
14	Valuation in range:	-72.00	to	+72.00;		-----				254
15						global concordance:	+12.00			255
16	*----- pairwise comparison -----*									256
17	Comparing actions : ('p3', 'p7')									257
18	crit. wght.	g(x)	g(y)	diff			ind	pref	r()	258
19	ec1	8.00	80.19	15.33	+64.86		5.00	10.00	+8.00	259
20	ec4	8.00	68.70	36.31	+32.39		5.00	10.00	+8.00	260
21	ec8	8.00	91.94	38.31	+53.63		5.00	10.00	+8.00	261
22	en3	6.00	46.78	30.70	+16.08		5.00	10.00	+6.00	262
23	en5	6.00	27.25	35.52	-8.27		5.00	10.00	+0.00	263
24	en6	6.00	1.65	69.71	-68.06		5.00	10.00	-6.00	264
25	en9	6.00	14.85	13.10	+1.75		5.00	10.00	+6.00	265
26	so2	12.00	58.85	68.06	-9.21		5.00	10.00	+0.00	266
27	so7	12.00	15.49	58.45	-42.96		5.00	10.00	-12.00	267
28	Valuation in range:	-72.00	to	+72.00;		-----				268
29						global concordance:	+18.00			269

In both cases, choosing only significances that are compatible with the given weights preorder will not always result in positively validated outranking situations.

19.3 Computing the Stability Denotation of Outranking Situations

Stability levels ± 4 and ± 3 are, the case given, easy to detect. Detecting a stability level ± 2 is far less obvious. Now, it is precisely again the bipolar-valued epistemic characteristic domain that will give us a way to implement an effective test for stability level $+2$ and -2 (Bisdorff 2004a,b).

Let us consider the significance equivalence classes we observe in the given weights preorder. Here, we observe three weight classes: 6, 8, and 12, in an increasing order (see Listing 19.2 on page 262). In the pairwise comparisons, shown above, these equivalence classes may appear positively or negatively, besides the indeterminate significance of value 0.00. We thus get the following ordered bipolar list of significance weights: $W = [-12, -8, -6, 0, 6, 8, 12]$.

In all the pairwise marginal comparisons shown in the previous section, we may observe that each one of the nine criteria assigns one precise item out of this list W . Let us denote by $q[i]$ the number of criteria assigning item $W[i]$ and $Q[i]$ the cumulative sums of these $q[i]$ counts, where i is an index in the range of the length of list W .

In the comparison of alternatives p_2 and p_1 , for instance (see Listing 19.5 on page 265), we observe the following counts: 289
290

$W[i]$	-12	-8	-6	0	6	8	12	t3.1
$q[i]$	0	0	2	1	1	3	2	t3.2
$Q[i]$	0	0	2	3	4	7	9	t3.3

Let us denote by $-q$ and $-Q$ the reversed versions of the q and the Q lists. We 291
thus obtain the following result: 292

$W[i]$	-12	-8	-6	0	6	8	12	t6.1
$-q[i]$	2	3	1	1	2	0	0	t6.2
$-Q[i]$	2	5	6	7	9	9	9	t6.3

Now, a pairwise outranking situation will be qualified at stability level $+2$, i.e. 293
positively validated with any significance weights that are compatible with the given 294
weights preorder, when for all i , we observe $Q[i] \leq -Q[i]$, and there exists one i 295
such that $Q[i] < -Q[i]$. Similarly, a pairwise outranked situation will be qualified 296
at stability level -2 , when for all i , we observe $Q[i] \geq -Q[i]$, and there exists one 297
 i such that $Q[i] > -Q[i]$ (Bisdorff 2004b). 298

Let us verify that the outranking situation observed, for instance, between 299
alternatives p_2 and p_1 does indeed verify this *first-order distributional dominance* 300
condition. 301

$W[i]$	-12	-8	-6	0	6	8	12	t9.1
$Q[i]$	0	0	2	3	4	7	9	t9.2
$-Q[i]$	2	5	6	7	9	9	9	t9.3

Notice that outranking situations qualified at stability levels ± 4 and ± 3 evidently 302
verify the stability level ± 2 test above. The outranking situation between alterna- 303
tives p_7 and p_3 does not, however, verify this test (see Listing 19.6 on page 266). 304

$W[i]$	-12	-8	-6	0	6	8	12	t12.1
$q[i]$	0	3	1	0	3	0	2	t12.2
$Q[i]$	0	3	4	4	7	7	9	t12.3
$-Q[i]$	2	2	5	5	6	9	9	t12.4

AQ3 This time, not all the $Q[i]$ are *lower than or equal to* the corresponding $-Q[i]$ 305
terms. Hence the outranking situation between p_7 and p_3 is not positively validated 306

with all potential significance weights that are compatible with the given weights 307
preorder. 308

Using this stability denotation, we may, hence, define the following *robust* 309
version of a bipolar-valued outranking digraph. 310

19.4 Robust Bipolar-Valued Outranking Digraphs

311

Definition 19.1 (Robust Outranking Situation)

312

- We say that decision alternative x *robustly outranks* decision alternative y 313
when: 314
 - x is at least as well evaluated as y at stability level +2 or higher. 315
 - We do not observe any considerable counter-performance of x on a discordant 316
criterion. 317
- Dually, we say that decision alternative x *does not robustly outrank* decision 318
alternative y when: 319
 - x is not at least as well evaluated as y at stability level -2 or lower. 320
 - We do not observe any considerable better performance of x on a discordant 321
criterion. 322
- Otherwise, the outranking situation is indeterminate. 323

The corresponding *robust* outranking digraph can be computed as follows with 324
the RobustOutrankingDigraph class: 325

Listing 19.7 Computing a robust outranking digraph

```

1 >>> from outrankingDigraphs import\
2 ...           RobustOutrankingDigraph
3 >>> rg = RobustOutrankingDigraph(pt) # same pt
4 >>> rg
5 *----- Object instance description -----*
6 Instance class      : RobustOutrankingDigraph
7 Instance name       : robust_random3ObjectivesPerfTab
8 Actions             : 7
9 Criteria            : 9
10 Size               : 22
11 Determinateness (%) : 68.45
12 Valuation domain   : [-1.00;1.00]
13 Attributes          : ['name', 'methodData', 'actions',
14           'order', 'criteria', 'evaluation',
15           'vetos', 'valuationdomain',
16           'cardinalRelation', 'ordinalRelation',
17           'equisignificantRelation', 'unanimousRelation',
18           'relation', 'gamma', 'notGamma']
19 >>> rg.showRelationTable()
20 * ---- Relation Table ----

```

21	$r / (\text{stab})$	'p1'	'p2'	'p3'	'p4'	'p5'	'p6'	'p7'	346
22	-----	-----	-----	-----	-----	-----	-----	-----	347
23	'p1'	+1.00	-0.42	+0.00	-0.69	+0.39	+0.11	+0.00	348
24		(+4)	(-2)	(+0)	(-3)	(+2)	(+2)	(-1)	349
25	'p2'	+0.58	+1.00	+0.83	+0.00	+0.58	+0.58	+0.58	350
26		(+2)	(+4)	(+3)	(+2)	(+2)	(+2)	(+2)	351
27	'p3'	+0.25	-0.33	+1.00	+0.00	+0.50	+1.00	+0.00	352
28		(+2)	(-2)	(+4)	(+0)	(+2)	(+2)	(+1)	353
29	'p4'	+0.78	+0.00	+0.61	+1.00	+1.00	+1.00	+0.67	354
30		(+3)	(-1)	(+3)	(+4)	(+4)	(+4)	(+2)	355
31	'p5'	-0.11	-0.50	-0.25	-0.89	+1.00	+0.11	-0.14	356
32		(-2)	(-2)	(-2)	(-3)	(+4)	(+2)	(-2)	357
33	'p6'	+0.22	-0.42	+0.00	-1.00	+0.17	+1.00	-0.11	358
34		(+2)	(-2)	(+1)	(-2)	(+2)	(+4)	(-2)	359
35	'p7'	+0.22	-0.50	+0.00	+0.00	+0.78	+0.42	+1.00	360
36		(+2)	(-2)	(+1)	(-1)	(+3)	(+2)	(+4)	361

All outranking situations, qualified at stability level ± 1 , are now put to an 362 *indeterminate* status (see Listing 19.7, Lines 23–36 above). In the example here, 363 three positive outrankings get dropped: between p3 and p7, between p7 and p3, 364 and between p6 and p3, where the last situation is actually already doubtful because 365 of a veto situation. Three negative outrankings get dropped as well: between p1 and 366 p7, between p4 and p2, and between p7 and p4. 367

Notice by the way that outranking or outranked situations, although qualified 368 at level ± 2 or ± 3 , may nevertheless become doubtful because of considerable 369 performance differences. We observe such a doubtful situation when comparing, 370 for instance, alternatives p2 and p4 (see Listing 19.8, Lines 4–8). 371

Listing 19.8 Inspecting polarised outranking situations

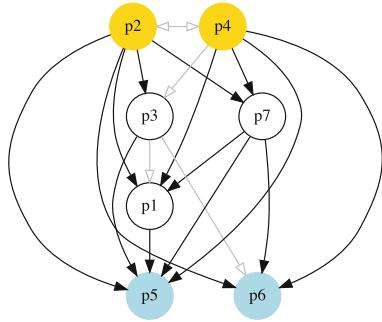
```

1 >>> rg.showPolarisations()
2 *---- Negative polarisations ----*
3 number of negative polarisations : 3
4 1:  $r(p2 \geq p4) = 0.33$ 
5 criterion: en3
6 Considerable performance difference : -60.02
7 Veto discrimination threshold : -60.00
8 Polarisation:  $r(p2 \geq p4) = 0.33 \Rightarrow 0.00$ 
9 2:  $r(p6 \geq p3) = 0.06$ 
10 criterion: ec8
11 Considerable performance difference : -62.31
12 Veto discrimination threshold : -60.00
13 Polarisation:  $r(p6 \geq p3) = 0.06 \Rightarrow 0.00$ 
14 3:  $r(p6 \geq p4) = -0.36$ 
15 criterion: en3
16 Considerable performance difference : -68.27
17 Veto discrimination threshold : -60.00
18 Polarisation:  $r(p6 \geq p4) = -0.36 \Rightarrow -1.00$ 

```

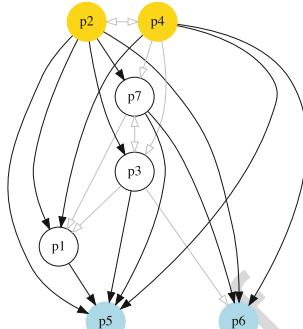
Despite being robust, the ‘*at least as well evaluated as*’ situation between 390 alternatives p2 and p4 becomes doubtful because of a considerable counter- 391 performance (-60.02) of p2 observed on criterion en3, a negative difference which 392 exceeds slightly the assumed veto discrimination threshold $v = 60.00$. The same 393 polarisation to 0.00 (indeterminate) happens when comparing alternatives p6 and 394

Standard strict outranking digraph



Digraph3 (graphviz), R. Bisdorff, 2020

Robust strict outranking digraph



Digraph3 (graphviz), R. Bisdorff, 2020

Fig. 19.1 Standard versus robust strict outranking digraphs oriented by their initial and terminal prekernels

p3 (Line 13). Notice the polarisation to -1.0 (certainly false) of the “*at least as well evaluated as*” situation between alternatives p6 and p4 (Line 18).

One may finally compare in Fig. 19.1 the *standard* and the *robust* versions of the corresponding strict outranking digraphs, both oriented by their respective identical initial and terminal prekernels.

The robust version (right in Fig. 19.1) drops two strict outranking situations: between p4 and p7 and between p7 and p1. The remaining 14 strict outranking (respectively, outranked) situations are now all verified at a stability level of $+2$ and more (respectively, -2 and less). They remain valid, hence, with all potential significance weights that are compatible with the given significance weights preordering (see Listing 19.2 on page 262).

To appreciate the apparent partial ordering of both the standard and the robust strict outranking digraphs shown in Fig. 19.1, let us have in Fig. 19.2 on the following page a final heatmap view on the underlying performance tableau ordered by the NETFLOWS ranking rule actually applied to the robust version of the outranking digraph (see the outrankingModel='this' flag in Line 4 below).

Listing 19.9 Computing a robust performance heatmap view

```

1 >>> rg.showHTMLPerformanceHeatmap (\ 411
2 ...           Correlations=True,\ 412
3 ...           colorLevels=5,\ 413
4 ...           outrankingModel='this',\ 414
5 ...           rankingRule='NetFlows') 415

```

As the initial prekernel $\{p_2, p_4\}$ is in the robust outranking digraph validated at least at stability level ± 2 , recommending alternatives p4 and p2 as potential best choices appears robustly justified. Alternative p4 represents indeed an overall *best*

Heatmap of Performance Tableau 'robust_random3ObjectivesPerfTab'

criteria	ec4	so2	ec1	ec8	en9	en3	so7	en6	en5
weights	+8.00	+12.00	+8.00	+8.00	+6.00	+6.00	+12.00	+6.00	+6.00
tau ^(*)	+0.55	+0.52	+0.45	+0.38	+0.31	+0.29	+0.24	+0.05	+0.05
p4	72.26	49.05	85.19	44.62	50.92	80.81	55.57	66.21	49.69
p2	86.00	89.12	89.77	89.43	26.65	20.79	84.73	18.66	23.83
p3	68.70	58.85	80.19	91.94	14.85	46.78	15.49	1.65	27.25
p7	36.31	68.06	15.33	38.31	13.10	30.70	58.45	69.71	35.52
p1	22.65	22.43	38.11	77.02	44.37	58.16	28.41	11.41	31.40
p6	75.76	48.59	21.06	29.63	32.96	12.54	26.40	36.04	43.09
p5	8.96	12.36	46.75	35.91	9.83	31.05	44.92	31.22	29.52

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%
	orange	yellow	light green	green	dark green

(*) tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation

Outranking model: this, Ranking rule: NetFlows

Ordinal (Kendall) correlation between global ranking and global outranking relation: +0.960

Mean marginal correlation (a) : +0.338

Standard marginal correlation deviation (b) : +0.168

Ranking fairness (a) - (b) : +0.170

Fig. 19.2 Robust heatmap of the random 3-objectives performance tableau ordered by the NETFLOWS ranking rule

compromise choice between all decision objectives, whereas alternative p2 gives an 419 unanimous best choice with respect to two out of three decision objectives. It is up 420 to the decision maker to make his final choice. 421

19.5 Characterising Unopposed Multiobjective Outranking Situations

When facing a performance tableau involving multiple decision objectives, the 424 robustness level ± 3 may lead to distinguishing what we call *unopposed* outranking 425 situations, like the one shown in the previous section between alternatives p4 and 426 p1 ($r(p4 \succsim p1) = +0.78$, see Listing 19.4 on page 264, Line 11), namely 427 preferential situations that are more or less validated or invalidated by all the 428 decision objectives. 429

Definition 19.2 (Unopposed Outranking Situation)

- We say that decision alternative x *unopposedly outranks* decision alternative y 431 when x positively outranks y on one or more decision objectives without x being 432 positively outranked by y on any decision objective. 433

- Dually, we say that decision alternative x is *unopposedly outranked* by decision alternative y when x is positively outranked by y on one or more decision objectives without x positively outranking y on any decision objective. 434
- 435
- 436

Let us reconsider, for instance, the performance tableau pt with three decision objectives already seen in Listing 19.1 on page 261: 437

```

1 >>> pt.showObjectives() 439
2     ----- show objectives -----
3     Eco: Economical aspect 440
4         ec1 criterion of objective Eco 8 441
5         ec4 criterion of objective Eco 8 442
6         ec8 criterion of objective Eco 8 443
7     Total weight: 24.00 (3 criteria) 444
8     Soc: Societal aspect 445
9         so2 criterion of objective Soc 12 446
10        so7 criterion of objective Soc 12 447
11     Total weight: 24.00 (2 criteria) 448
12     Env: Environmental aspect 449
13        en3 criterion of objective Env 6 450
14        en5 criterion of objective Env 6 451
15        en6 criterion of objective Env 6 452
16        en9 criterion of objective Env 6 453
17     Total weight: 24.00 (4 criteria) 454

```

We notice in this example three decision objectives of equal importance (see 456
Lines 3, 13, and 17). What will be the outranking situations that are positively 457
(respectively, negatively) validated for each one of the decision objectives taken 458
individually? 459

Such unopposed multiobjective outranking situations may be obtained by operating 460
an epistemic *average fusion* (see the `symmetricAverage()` method) of the 461
marginal outranking digraphs restricted to the coalition of criteria supporting each 462
one of the decision objectives (see Listing 19.10). 463

Listing 19.10 Computing unopposed outranking situations

```

1 >>> from outrankingDigraphs import\ 464
2     ...     BipolarOutrankingDigraph 465
3 >>> geco = BipolarOutrankingDigraph(pt,\ 466
4     ...         objectivesSubset=['Eco']) 467
5 >>> gsoc = BipolarOutrankingDigraph(pt,\ 468
6     ...         objectivesSubset=['Soc']) 469
7 >>> genv = BipolarOutrankingDigraph(pt,\ 470
8     ...         objectivesSubset=['Env']) 471
9 >>> from digraphs import FusionLDigraph 472
10 >>> objectiveWeights = \ 473
11     ...         [pt.objectives[obj] ['weight']\ 474
12     ...             for obj in t.objectives] 475
13 >>> uopg = FusionLDigraph([geco,gsoc,genv],\ 476
14     ...             operator='o-average',\ 477
15     ...             weights=objectiveWeights) 478
16 >>> uopg.showRelationTable(ReflexiveTerms=False) 479

```

Positive (respectively, negative) $r(x \succsim y)$ characteristic values, like $r(p1 \succsim p5) = +0.39$ (see Listing 19.10, Line 20 above), show hence only outranking situations being validated (respectively, invalidated) by one or more decision objectives without being invalidated (respectively, validated) by any other decision objective.

For easily computing this kind of *unopposed multiobjective outranking* digraphs, the `outrankingDigraphs` module conveniently provides a corresponding `UnOpposedBipolarOutrankingDigraph` constructor .

Listing 19.11 Computing unopposed outranking digraphs

```
1 >>> from outrankingDigraphs import\  
2 ...                         UnOpposedBipolarOutrankingDigraph  
3 >>> uopg = UnOpposedBipolarOutrankingDigraph(pt)  
4 >>> uopg  
5 *----- Object instance description -----*  
6 Instance class      : UnOpposedBipolarOutrankingDigraph  
7 Instance name       : unopposed_outrankings  
8 Actions             : 7  
9 Criteria            : 9  
10 Size               : 13  
11 Oppositeness (%)  : 43.48  
12 Determinateness (%) : 61.71  
13 Valuation domain   : [-1.00;1.00]  
14 Attributes          : ['name', 'actions',  
15                      'valuationdomain', 'objectives',  
16                      'criteria', 'methodData',  
17                      'evaluation', 'order', 'runTimes',  
18                      'relation', ...  
19                      'gamma', 'notGamma']  
20 >>> uopg.computeOppositeness(InPercent=True)  
21   {'standardSize': 23, 'unopposedSize': 13,  
22   'oppositeness': 43.47826086956522}
```

The resulting *unopposed* outranking digraph keeps in fact 13 (see Listing 19.11, Lines 10–11) out of the 23 positively validated *standard* outranking situations, leading to a degree of *oppositeness* – preferential disagreement between decision objectives – of $(1.0 - 13/23) = 0.4348$.

We can now, for instance, verify the unopposed status of the outranking situation observed between alternatives p_1 and p_5 . 525
526

Listing 19.12 Example of unopposed multiobjective outranking situation

```

1 >>> uopg.showPairwiseComparison('p1', 'p5')                                527
2 *----- pairwise comparison -----*                                         528
3 Comparing actions : ('p1', 'p5')                                         529
4 crit. wght. g(x) g(y) diff    | ind  pref   r()                         530
5 ec1  8.00  38.11  46.75  -8.64 | 5.00 10.00 +0.00                     531
6 ec4  8.00  22.65  8.96   +13.69 | 5.00 10.00 +8.00                     532
7 ec8  8.00  77.02  35.91  +41.11 | 5.00 10.00 +8.00                     533
8 en3  6.00  58.16  31.05  +27.11 | 5.00 10.00 +6.00                     534
9 en5  6.00  31.40  29.52   +1.88 | 5.00 10.00 +6.00                     535
10 en6 6.00  11.41  31.22  -19.81 | 5.00 10.00 -6.00                     536
11 en9 6.00  44.37  9.83   +34.54 | 5.00 10.00 +6.00                     537
12 so2 12.00  22.43  12.36   +10.07 | 5.00 10.00 +12.00                   538
13 so7 12.00  28.41  44.92  -16.51 | 5.00 10.00 -12.00                   539
14 Valuation in range: -72.00 to +72.00;                                     540
15                                     global concordance: +28.00           541

```

In Listing 19.12, we see that alternative p_1 does indeed positively outrank p_5 from the economic perspective ($r(p_1 \succsim_{Eco} p_5) = +16/24$) as well as from the environmental perspective ($r(p_1 \succsim_{Env} p_5) = +12/24$), whereas, from the societal perspective, both alternatives appear incomparable ($r(p_1 \succsim_{Soc} p_5) = 0/24$).

When proportionally equal criteria significance weights per objective are given, these outranking situations appear hence *stable* with respect to all possible importance weights we could allocate to the decision objectives.

This gives way for computing multiobjective *Pareto efficient* choice recommendations.

19.6 Computing Pareto Efficient Multiobjective Choices

552

Indeed, best choice recommendations, computed from an *unopposed multiobjective* outranking digraph, deliver the case given *Pareto efficient* choices.

553

554

Listing 19.13 Pareto efficient multiobjective choice

```

1 >>> uopg.showBestChoiceRecommendation()                                555
2 Best choice recommendation(s) (BCR)                                     556
3 (in decreasing order of determinateness)                                557
4 Credibility domain: [-1.00,1.00]                                         558
5 === >> potential first choice(s)                                     559
6 choice : ['p2', 'p4', 'p7']                                              560
7     independence : 0.00                                                 561
8     dominance : 0.33                                                 562
9     absorbency : 0.00                                                 563
10    covering (%) : 33.33                                              564
11    determinateness (%) : 50.00                                         565
12 === >> potential last choice(s)                                     566
13 choice : ['p3', 'p5', 'p6', 'p7']                                         567

```

14	independence	:	0.00	568
15	dominance	:	-0.61	569
16	absorbency	:	0.11	570
17	covered (%)	:	33.33	571
18	determinateness (%)	:	50.00	572

Our previous *robust* best choice recommendation (p2 and p4, see Fig. 19.1 on page 271) remains, in this example here, *stable*. We recover indeed the best choice recommendation [p2, p4, p7] (see Listing 19.13 on the previous page, Line 6). Yet, notice that decision alternative p7 appears to be at the same time a potential *first* as well as a potential *last* choice recommendation (see Line 13), a consequence of p7 being completely *incomparable* to the other decision alternatives when restricting the comparability to only unopposed strict outranking situations.

This kind of Pareto efficient result is shown in Fig. 19.3.

```

1 >>> (~(-uopg)).exportGraphViz(fileName = 'unopDigraph', \
2 ...                           firstChoice = ['p2','p4'], \
3 ...                           lastChoice = ['p3','p5','p6']) \
4 *---- exporting a dot file for GraphViz tools ----* \
5 Exporting to unopDigraph.dot \
6 dot -Grankdir=BT -Tpng unopDigraph.dot -o unopDigraph.png

```

In order to make now an eventual best unique choice, a decision maker will necessarily have to weigh, in a second stage of the decision aiding process, the relative importance of the individual decision objectives.

For concluding, let us mention that it is precisely again our bipolar-valued logical characteristic framework that provides us here with a first-order distributional dominance test for effectively qualifying the stability level ± 2 robustness of an

Standard strict outranking digraph

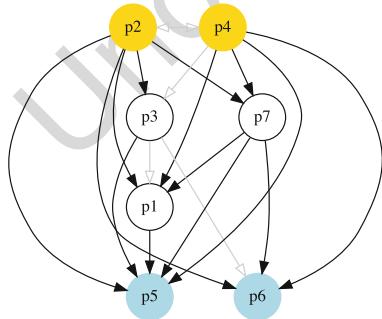


Diagram3 (graphviz), R. Bisдорff, 2020

Unopposed strict outranking digraph

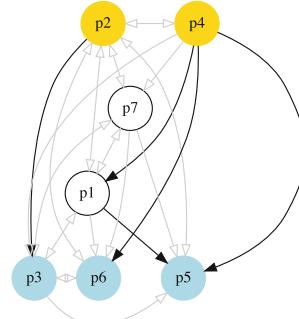


Diagram3 (graphviz), R. Bisдорff, 2020

Fig. 19.3 Standard versus *unopposed* strict outranking digraphs oriented by first and last choice recommendations

outranking digraph when facing performance tableaux with criteria of only ordinal-valued significance. A real-world application of our stability analysis with such a kind of performance tableau may be consulted in Bisdorff (2015). 593
594
595

In a *social choice* context, where decision objectives would match different political parties, Pareto efficient best choice recommendations represent in fact *multipartisan* social choices that may judiciously temper plurality tyranny effects. 597
598
599

References

600

- Bisdorff R (2004a) Concordant outranking with multiple criteria of ordinal significance. 4OR 601
2(4):293–308. <http://hdl.handle.net/10993/23721> 602
- Bisdorff R (2004b) Preference aggregation with multiple criteria of ordinal significance. Annales 603
du LAMSADE 3:25–44. <http://hdl.handle.net/10993/42420> 604
- Bisdorff R (2014) On confident outrankings with multiple criteria of uncertain significance. In: 605
Mousseau V, Pirlot M (eds) DAP'2014 from multiple criteria decision aid to preference 606
learning. Ecole Centrale de Paris, pp 1–6. <http://hdl.handle.net/10993/23910> 607
- Bisdorff R (2015) The EURO 2004 best poster award: choosing the best poster in a scientific 608
conference. In: Bisdorff R, Dias L, Meyer P, Mousseau V, Pirlot M (eds) Evaluation and 609
decision models with multiple criteria: case studies. Springer, Berlin, pp 117–166 610
- Bisdorff R, Meyer P, Veneziano T (2009) Inverse analysis from a CONDORCET robustness 611
denotation of valued outranking relations. In: Rossi F, Tsoukiàs A (eds) Algorithmic decision 612
theory. Lecture notes in artificial intelligence (LNAI), vol 5783. Springer, Berlin, pp 180–191. 613
<http://hdl.handle.net/10993/23454> 614
- Bisdorff R, Meyer P, Veneziano T (2014) Elicitation of criteria weights maximising the stability 615
of pairwise outranking statements. J Multi-Criteria Decision Analy (Wiley) 21:113–124. <http://hdl.handle.net/10993/23701> 616
617

AUTHOR QUERIES

- AQ1.** Please check if the section title “Cardinal or Ordinal Criteria Significance Weights?” is fine and amend if required.
- AQ2.** Please check if the sentence “Let us inspect the details of ...” is fine as edited and amend if required.
- AQ3.** Please check if the sentence “This time, not all the ...” is fine as edited and amend if required.
- AQ4.** Please check if the edits made in the sentence “As the initial prekernel ...” convey the intended meaning and amend if required.

Uncorrected Proof

Chapter 20

Tempering Plurality Tyranny Effects

in Social Choice

The choice of a voting procedure shapes the democracy in which we live 4
— *Baujard, Gavrel, Iggersheim, Laslier, and Lebon (2013)* 5

Contents	7
20.1 Introduction	279 8
20.2 Two-Stage Elections with Multipartisan Primary Selection	280 9
20.3 Bipolar Approval–Disapproval Voting Systems	287 10
20.4 Pairwise Comparison of Approval–Disapproval Votes	290 11
20.5 Three-Valued Evaluative Voting Systems	293 12
20.6 Favouring Multipartisan Candidates	296 13

Abstract In a *social choice* context, where decision objectives would match 14 different political parties, Pareto efficient choice recommendations represent in fact 15 *multipartisan* social choices that may judiciously deliver the primary selection in 16 a two-stage election system. Our bipolar-valued outranking model is based on 17 approvals-disapprovals of ‘*at least as well evaluated as*’ statements. A similar 18 approach is put into practice with approval–disapproval voting systems. When con- 19 verting such approval–disapproval voting ballots into corresponding performance 20 records, we obtain a $(-1, 0, 1)$ -valued evaluative voting system. We eventually 21 show that in such an approval–disapproval voting system, the winner tends to be 22 among the more or less multipartisan candidates. 23

20.1 Introduction

From the seminal work by *Bernard Roy*, tempering plurality tyranny effects is 25 achieved in the outranking approach by taking into account considerable negative 26 performance differences that render doubtful otherwise positive ‘*at least as well 27 evaluated as*’ situations (Benayoun et al. 1966). In the social choice context of 28

general elections, such a polarisation is not feasible as the genuine uninominal voting ballots do not contain rich enough preferential information. Yet, when matching decision objectives with political parties, the Pareto efficient outranking situations seen in Sect. 19.6 correspond to multipartisan ‘*at least as well approved as*’ situations when pairwisely comparing the eligible candidates. A best choice recommendation based on the corresponding multipartisan strict ‘*better approved as*’ situations may, the case given, deliver a convincing primary selection in a two-stage election.

20.2 Two-Stage Elections with Multipartisan Primary Selection

To compute multipartisan social choices, we need to, first, convert a given linear voting profile with pre-election polls into a corresponding performance tableau. We shall illustrate this point with a voting profile we discussed already in Chap. 7.

Listing 20.1 Example of a 3 parties voting profile

```

1 >>> from votingProfiles import RandomLinearVotingProfile
2 >>> lvp = RandomLinearVotingProfile(numberOfCandidates=15,
3 ...                               numberOfVoters=1000,
4 ...                               WithPolls=True,
5 ...                               partyRepartition=0.5,
6 ...                               other=0.1,
7 ...                               seed=0.9189670954954139)
8 >>> lvp
9      *----- VotingProfile instance description -----*
10     Instance class   : RandomLinearVotingProfile
11     Instance name    : randLinearProfile
12     Candidates       : 15
13     Voters          : 1000
14     Attributes       : ['name', 'seed', 'candidates',
15                           'voters', 'WithPolls', 'RandomWeights',
16                           'sumWeights', 'poll1', 'poll2',
17                           'other', 'partyRepartition',
18                           'linearBallot', 'ballot']
19 >>> lvp.showRandomPolls()
20     Random repartition of voters
21     Party_1 supporters : 460 (46.0%)
22     Party_2 supporters : 436 (43.6%)
23     Other voters       : 104 (10.4%)
24     *----- random polls -----*
25     Party-1(46.0%) | Party-(43.6%) | expected
26     -----
27     c06 : 19.91% | c11 : 22.94% | c06 : 15.00%
28     c07 : 14.27% | c08 : 15.65% | c11 : 13.08%
29     c03 : 10.02% | c04 : 15.07% | c08 : 09.01%
30     c13 : 08.39% | c06 : 13.40% | c07 : 08.79%

```

31	c15 : 08.39%	c03 : 06.49%	c03 : 07.44%	72
32	c11 : 06.70%	c09 : 05.63%	c04 : 07.11%	73
33	c01 : 06.17%	c07 : 05.10%	c01 : 05.06%	74
34	c12 : 04.81%	c01 : 05.09%	c13 : 05.04%	75
35	c08 : 04.75%	c12 : 03.43%	c15 : 04.23%	76
36	c10 : 04.66%	c13 : 02.71%	c12 : 03.71%	77
37	c14 : 04.42%	c14 : 02.70%	c14 : 03.21%	78
38	c05 : 04.01%	c15 : 00.86%	c09 : 03.10%	79
39	c09 : 01.40%	c10 : 00.44%	c10 : 02.34%	80
40	c04 : 01.18%	c05 : 00.29%	c05 : 01.97%	81
41	c02 : 00.90%	c02 : 00.21%	c02 : 00.51%	82

In this example (see Listing 20.1 on the preceding page, Lines 19–41), we observe 460 Party-1 supporters (46%), 436 Party-2 supporters (43.6%), and 104 other voters (10.4%). Favorite candidates of Party-1 supporters, with more than 10%, are c06 (19.91%), c07 (14.27%), and c03 (10.02%). Whereas for Party-2 supporters, favorite candidates are c11 (22.94%), followed by c08 (15.65%), c04 (15.07%), and c06 (13.4%).

We can convert this linear voting profile into a `PerformanceTableau` object where each political party matches a decision objective.

Listing 20.2 Converting a voting profile into a performance tableau

```

1  >>> lvp.save2PerfTab('votingPerfTab')                                91
2  >>> from perfTabs import PerformanceTableau                         92
3  >>> vpt = PerformanceTableau('votingPerfTab')                      93
4  >>> vpt
5  *----- PerformanceTableau instance description ---*               95
6  Instance class      : PerformanceTableau                         96
7  Instance name       : votingPerfTab                                97
8  Actions             : 15                                         98
9  Objectives          : 3                                           99
10 Criteria            : 1000                                       100
11 Attributes          : ['name', 'actions', 'objectives',          101
12           'criteria', 'weightPreorder', 'evaluation']                102
13 >>> vpt.objectives                                         103
14 OrderedDict([
15     ('party0', {'name': 'other', 'weight': Decimal('104'),        104
16       'criteria': ['v0003', 'v0008', 'v0011', ... ]}),           105
17     ('party1', {'name': 'party 1', 'weight': Decimal('460'),        106
18       'criteria': ['v0002', 'v0006', 'v0007', ... ]}),           107
19     ('party2', {'name': 'party 2', 'weight': Decimal('436'),        108
20       'criteria': ['v0001', 'v0004', 'v0005', ... ]})            109
21 ])

```

In Listing 20.2, the linear voting profile `lvp` is first stored in `PerformanceTableau` format (see Line 1). In Line 3, the `PerformanceTableau` class reloads this stored performance tableau data. The three parties of the linear voting profile represent three decision objectives and the 1000 voters are distributed as 1000 performance criteria according to the party they support.

In order to operate now a *primary multipartisan selection* of potential election winners, we compute the corresponding unopposed multiobjective outranking digraph (see Sect. 19.5). 117
118
119

Listing 20.3 Computing unopposed multiobjective outranking situations

```

1 >>> from outrankingDigraphs import \
2     ...     UnOpposedBipolarOutrankingDigraph 120
3 >>> uog = UnOpposedBipolarOutrankingDigraph(vpt) 121
4 >>> uog 122
5 *----- Object instance description -----* 123
6 Instance class      : UnOpposedBipolarOutrankingDigraph 124
7 Instance name       : unopposed_outrankings 125
8 Actions             : 15 126
9 Criteria            : 1000 127
10 Size               : 34 128
11 Oppositeness (%)  : 67.31 129
12 Determinateness (%) : 57.61 130
13 Valuation domain   : [-1.00;1.00] 131
14 Attributes          : ['name', 'actions', 'valuationdomain', 132
15                 'objectives', 'criteria', 133
16                 'methodData', 134
17                 'evaluation', 'order', 'runTimes', 135
18                 'relation', 136
19                 'marginalRelationsRelations', 137
20                 'gamma', 'notGamma'] 138
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139

```

From the potential 105 pairwise outranking situations, we keep 34 positively validated outranking situations, leading to a degree of *oppositeness* between political parties of 67.31% (see Listing 20.3 Line 11). 140
141
142

The corresponding bipolar-valued relation table can be shown by orienting the list of candidates with the help of the initial and terminal prekernels. 143
144

Listing 20.4 Computing unopposed multiobjective outranking situations

```

1 >>> uog.showPreKernels() 145
2 *--- Computing preKernels ---* 146
3 Dominant preKernels : 147
4     ['c11', 'c06', 'c13', 'c15'] 148
5     independence : 0.0 149
6     dominance : 0.18 150
7     absorbency : -0.66 151
8     covering : 0.43 152
9 Absorbent preKernels : 153
10    ['c02', 'c04', 'c14', 'c03'] 154
11    independence : 0.0 155
12    dominance : 0.0 156
13    absorbency : 0.37 157
14    covered : 0.46 158
15 >>> orientedCandidatesList = ['c06','c11','c13','c15',\ 159
16 ...     'c01','c05','c07','c08','c09','c10','c12',\ 160
17 ...     'c02','c03','c04','c14'] 161

```

Multipartisan outranking situations

r(x S y)	a06	a11	a13	a15	a01	a05	a07	a08	a09	a10	a12	a02	a03	a04	a14
a06	-	0.00	0.00	0.00	0.44	0.00	0.25	0.00	0.00	0.00	0.56	0.86	0.29	0.00	0.57
a11	0.00	-	0.00	0.00	0.00	0.55	0.00	0.18	0.59	0.51	0.39	0.80	0.00	0.42	0.47
a13	0.00	0.00	-	0.00	0.00	0.52	-0.27	0.00	0.00	0.00	0.00	0.77	0.00	0.00	0.16
a15	0.00	0.00	0.00	-	0.00	0.39	0.00	0.00	0.00	0.00	0.00	0.66	0.00	0.00	0.00
a01	-0.44	0.00	0.00	0.00	-	0.00	0.00	0.00	0.00	0.00	0.00	0.77	0.00	0.00	0.20
a05	0.00	-0.55	-0.52	-0.39	0.00	-	0.00	-0.47	0.00	-0.12	0.00	0.37	0.00	0.00	0.00
a07	-0.25	0.00	0.27	0.00	0.00	0.00	-	0.00	0.00	0.00	0.30	0.83	0.00	0.00	0.38
a08	0.00	-0.18	0.00	0.00	0.00	0.47	0.00	-	0.00	0.00	0.00	0.77	0.00	0.29	0.00
a09	0.00	-0.59	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-	0.00	0.00	0.55	0.00	0.00
a10	0.00	-0.51	0.00	0.00	0.00	0.12	0.00	0.00	0.00	-	0.00	0.50	0.00	0.00	0.00
a12	-0.56	-0.39	0.00	0.00	0.00	0.00	-0.30	0.00	0.00	0.00	-	0.72	0.00	0.00	0.10
a02	-0.86	-0.80	0.77	-0.66	-0.77	-0.37	-0.83	-0.77	-0.55	-0.50	-0.72	-	0.00	0.00	0.00
a03	-0.29	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-	0.00	0.00
a04	0.00	-0.42	0.00	0.00	0.00	0.00	0.00	-0.29	0.00	0.00	0.00	0.00	0.00	-	0.00
a14	-0.57	-0.47	-0.16	0.00	-0.20	0.00	-0.38	0.00	0.00	0.00	-0.10	0.00	0.00	0.00	-

Valuation domain: [-1.00; +1.00]

Fig. 20.1 Relation table of multipartisan outranking digraph

```

18  >>> uog.showHTMLRelationTable (
19  ...     actionsList=orientedCandidatesList, \
20  ...     tableTitle='Unopposed three-partisan outrankings')

```

In Fig. 20.1, we may notice that the dominating outranking prekernel $\{c06, c11, c13, c15\}$ gathers in fact a multipartisan selection of potential election winners. It is worthwhile noticing that the majority margins obtained from a linear voting profile do verify the zero-sum rule: $(r(x \succ y) + r(y \succ x)) = 0.0$. To each positive outranking situation corresponds an equivalent negative converse situation and the resulting outranking and strict outranking digraphs are the same.

When restricting now, in a secondary election stage, the set of eligible candidates to this dominating prekernel, we may compute the actual best social choice.

Listing 20.5 Recommending the secondary election winner

```

1  >>> from outrankingDigraphs import BipolarOutrankingDigraph
2  >>> g2 = BipolarOutrankingDigraph(vpt, \
3  ...     actionsSubset=['c06','c11','c13','c15'])
4  >>> g2.showRelationTable(ReflexiveTerms=False)
5  * ----- Relation Table -----
6  r   | 'c06'  'c11'  'c13'  'c15'
7  -----|-----
8  'c06' |   -    +0.10  +0.48  +0.52
9  'c11' | -0.10   -    +0.27  +0.29
10 'c13' | -0.48  -0.27   -    +0.19
11 'c15' | -0.52  -0.29  -0.19   -
12 Valuation domain: [-1.0; 1.0]
13 >>> g2.computeCondorcetWinners()

```

```

14   ['c06'] 186
15 >>> g2.computeCopelandRanking () 187
16   ['c06', 'c11', 'c13', 'c15'] 188

```

Candidate c06 appears clearly to be the winner of this election. Notice by the way that the restricted pairwise outranking relation, shown in Listing 20.5 Lines 8–11, models a linear ordering of the preselected candidates. 189
190
191

We can eventually check the quality of this best choice by noticing that Candidate c06 represents indeed the *simple majority*, the *instant runoff*, the BORDA, as well as the CONDORCET winner of the initially given linear voting profile lvp. 192
193
194

```

1 >>> lvp.computeSimpleMajorityWinner () 195
2   ['c06'] 196
3 >>> lvp.computeInstantRunoffWinner () 197
4   ['c06'] 198
5 >>> lvp.computeBordaWinners () 199
6   ['c06'] 200
7 >>> from votingProfiles import MajorityMarginsDigraph 201
8 >>> cd = MajorityMarginsDigraph(lvp) 202
9 >>> cd.computeCondorcetWinners () 203
10  ['c06'] 204

```

In our example voting profile here, the multipartisan primary selection stage appears quite effective in reducing the number of eligible candidates to four out of a set of 15 candidates without by the way rejecting the actual winning candidate. 205
206
207

However, in a very *divisive* two major parties system, like in the USA, where preferences of the supporters of one major party are opposite to the preferences of the supporters of the other major party, the multipartisan outranking digraph will become nearly indeterminate. 208
209
210
211

In Listing 20.6 below, we generate such a divisive kind of linear voting profile with the help of the `DivisivePolitics` flag (see Lines 4 and 14–20). When now converting the voting profile into a performance tableau (Lines 20–21), we can compute the corresponding unopposed outranking digraph. 212
213
214
215

Listing 20.6 A divisive two-party example of a random linear voting profile

```

1 >>> from votingProfiles import RandomLinearVotingProfile 216
2 >>> lvp = RandomLinearVotingProfile (\ 217
3 ...     numberOfCandidates=7,numberOfVoters=500, \ 218
4 ...     WithPolls=True, partyRepartition=0.4,other=0.2,\ 219
5 ...     DivisivePolitics=True, seed=1) 220
6 >>> lvp.showRandomPolls () 221
7 Random repartition of voters 222
8   Party-1 supporters : 240 (48.00%) 223
9   Party-2 supporters : 160 (32.00%) 224
10  Other voters : 100 (20.00%) 225
11 *----- random polls ----- 226
12   Party_1(48.0%) | Party_2(32.0%) | expected 227
13   ----- 228
14   c2 : 30.84% | c1 : 30.84% | c2 : 15.56% 229
15   c3 : 23.67% | c4 : 23.67% | c3 : 12.91% 230
16   c7 : 17.29% | c6 : 17.29% | c7 : 11.43% 231

```

```

17  c5 : 11.22%    | c5 : 11.22%    | c1 : 11.00%    232
18  c6 : 09.79%    | c7 : 09.79%    | c6 : 10.23%    233
19  c4 : 04.83%    | c3 : 04.83%    | c4 : 09.89%    234
20  c1 : 02.37%    | c2 : 02.37%    | c5 : 08.98%    235
21 >>> lvp.save2PerfTab('divisiveExample')          236
22 >>> dvp = PerformanceTableau('divisiveExample') 237
23 >>> from outrankingDigraphs import \           238
24 ...          UnOpposedBipolarOutrankingDigraph 239
25 >>> uodg = UnOpposedBipolarOutrankingDigraph(dvp) 240
26 >>> uodg          241
27 *----- Object instance description -----* 242
28 Instance class : UnOpposedBipolarOutrankingDigraph 243
29 Instance name  : divisiveExample 244
30 Actions       : 7 245
31 Criteria      : 500 246
32 Size          : 0 247
33 Oppositeness (%) : 100.00 248
34 Determinateness (%) : 50.00 249
35 Valuation domain : [-1.00;1.00] 250

```

With an oppositeness degree of 100.0% (see Listing 20.6, Lines 32–33), the preferential disagreement between the political parties is complete, and the unopposed outranking digraph `uodg` becomes completely indeterminate as shown in the relation table below.

```

1 >>> uodg.showRelationTable(ReflexiveTerms=False) 255
2 * ---- Relation Table ---- 256
3 r | 'c1' 'c2' 'c3' 'c4' 'c5' 'c6' 'c7' 257
4 ----- 258
5 'c1' | - +0.00 +0.00 +0.00 +0.00 +0.00 +0.00 259
6 'c2' | +0.00 - +0.00 +0.00 +0.00 +0.00 +0.00 260
7 'c3' | +0.00 +0.00 - +0.00 +0.00 +0.00 +0.00 261
8 'c4' | +0.00 +0.00 +0.00 - +0.00 +0.00 +0.00 262
9 'c5' | +0.00 +0.00 +0.00 +0.00 - +0.00 +0.00 263
10 'c6' | +0.00 +0.00 +0.00 +0.00 +0.00 - +0.00 264
11 'c7' | +0.00 +0.00 +0.00 +0.00 +0.00 +0.00 - 265
12 Valuation domain: [-1.0; 1.0] 266

```

As a consequence, a multipartisan primary selection, computed with a `showBestChoiceRecommendation()` method, will keep the complete initial set of eligible candidates and, hence, become *ineffective* (see Listing 20.7, Line 6).

Listing 20.7 Example of ineffective primary multipartisan selection

```

1 >>> uodg.showBestChoiceRecommendation() 270
2 Rubis best choice recommendation(s) (BCR) 271
3 (in decreasing order of determinateness) 272
4 Credibility domain: [-1.00,1.00] 273
5 === >> ambiguous choice(s) 274
6 choice : ['c1','c2','c3','c4','c5','c6','c7'] 275
7 independence : 0.00 276
8 dominance : 1.00 277
9 absorbency : 1.00 278

```

```

10    covered (%) : 100.00
11    determinateness (%) : 50.00
12    - most credible action(s) = { }

```

With such kind of divisive voting profile, there may indeed not always exist an obvious winner. In Listing 20.8, we see, for instance, that the simple majority winner is $c2$ (Line 2), whereas the instant-run-off winner is $c6$ (Line 4). 282
283
284

Listing 20.8 Example of non-obvious secondary selection

```

1 >>> lvp.computeSimpleMajorityWinner()
2 ['c2']
3 >>> lvp.computeInstantRunoffWinner()
4 ['c6']
5 >>> from votingProfiles import MajorityMarginsDigraph
6 >>> cg = MajorityMarginsDigraph(lvp)
7 >>> cg.showRelationTable(ReflexiveTerms=False)
8 * ---- Relation Table ----
9   r() | 'c1' 'c2' 'c3' 'c4' 'c5' 'c6' 'c7'
10  -----|-----
11  'c1' |   -  -68  -90  -46  -68  -88  -84
12  'c2' |  +68   -  -32  +80  +46   -6  -24
13  'c3' |  +90  +32   -  +58  +46   +4   +8
14  'c4' |   +4  -80  -58   -  -16  -68  -72
15  'c5' |  +68  -46  -46  +16   -  -26  -64
16  'c6' |  +88   +6  -4   +68  +26   -  -2
17  'c7' |  +84  +24  -8  +72  +64   +2   -
18 Valuation domain: [-500;+500]
19 >>> cg.computeCondorcetWinners()
20 ['c3']
21 >>> lvp.computeBordaWinners()
22 ['c3', 'c7']
23 >>> cg.computeCopelandRanking()
24 ['c3', 'c7', 'c6', 'c2', 'c5', 'c4', 'c1']

```

But in our example here, we are lucky. When constructing with the pairwise majority margins digraph (Line 6), a CONDORCET winner, namely $a3$, becomes apparent (Lines 13 and 20), which is also one of the two BORDA winners (Line 22). More interesting even is to notice that the apparent majority margins digraph models in fact a linear ranking $[a3, a7, a6, a2, a5, a4, a1]$ of all the eligible candidates, as shown with a COPELAND ranking rule (Line 24). 309
310
311
312
313
314

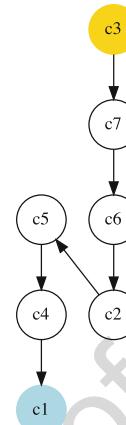
In Fig. 20.2, this linear ranking is shown with a graphviz drawing where all transitive arcs are dropped and the drawing is oriented with the CONDORCET winner $a3$ and loser $a1$ (Line 3 below). 315
316
317

```

1 >>> cg.closeTransitive(Reverse=True)
2 >>> cg.exportGraphViz('divGraph', \
3 ...           firstChoice=['c3'], lastChoice=['c1'])
4 *---- exporting a dot file for GraphViz tools -----
5 Exporting to divGraph.dot
6 dot -Grankdir=BT -Tpng divGraph.dot -o divGraph.png

```

Fig. 20.2 The linear ranking modelled by the majority margins digraph



Digraph3 (graphviz), R. Bisdorff, 2020

20.3 Bipolar Approval–Disapproval Voting Systems

324

Traditionally, most of the voting systems in use in the world do only collect approval 325 votes and abstentions, dismissing potentially strong disapproval opinions, not to 326 be assimilated to voting abstentions. Collecting both explicit approvals (+1) and 327 explicit disapprovals (−1) essentially enriches the expression of voters’ preferences. 328

In the `votingProfiles` module, we provide a `BipolarApprovalVotingProfile` class for handling voting results where, for each eligible Candidate 329 `x`, the voters are invited to *approve* (+1), *disapprove* (−1), or *ignore* (0) the 330 statement that Candidate `c` should win the election (Baujard et al. 2013). 331

File `bpApVotingProfile.py` contains an example of such an approval– 332 disapproval voting profile concerning 100 voters and 15 eligible candidates.¹ We 333 can inspect its content with the `BipolarApprovalVotingProfile` class: 334

Listing 20.9 Bipolar approval voting profiles

```

1  >>> from votingProfiles import \
2      ...                               BipolarApprovalVotingProfile
3  >>> bavp = BipolarApprovalVotingProfile('bpApVotingProfile')
4  >>> bavp
5  *----- VotingProfile instance description -----*
6  Instance class    : BipolarApprovalVotingProfile
7  Instance name     : bpApVotingProfile
8  Candidates        : 15
9  Voters            : 100
10 Attributes       : ['name', 'candidates', 'voters',
11                      'approvalBallot', 'netApprovalScores',
12                      'ballot']
```

¹ The file `bpApVotingProfile.py` may be found in the `examples` directory of the DIGRAPH3 resources.

Beside the candidates and voters attributes, we discover in Listing 20.10 348 the approvalBallot attribute which gathers approval–disapproval votes. Its 349 content is the following. 350

Listing 20.10 Inspecting an approval–disapproval ballot

```

1  >>> bavp.approvalBallot
2      {'v001':
3          {'c01': Decimal('0'),
4          ...
5          'c04': Decimal('1'),
6          ...
7          'c15': Decimal('0')
8          },
9      'v002':
10     {'c01': Decimal('-1'),
11     'c02': Decimal('0'),
12     ...
13     'c15': Decimal('1')
14     },
15     ...
16  'v100':
17     {'c01': Decimal('0'),
18     'c02': Decimal('1'),
19     ...
20     'c15': Decimal('1')
21     }
22 }
```

Let us denote by A_v the set of candidates approved by voter v . In the 373 approvalBallot attribute, we hence record in fact the bipolar-valued truth 374 characteristic values $r(c \in A_v)$ of the statements that Candidate c is *approved* by 375 voter v . In Listing 20.10 Line 5, we observe, for instance, that voter $v001$ positively 376 approves Candidate $c04$. And, in Line 10, we see that voter $v002$ negatively 377 approves, i.e. positively disapproves, Candidate $c01$. 378

The showApprovalResults() method and the showDisapprovalResults() 379 method show how many approvals (respectively, disapprovals) each 380 candidate receives. 381

```

1  >>> bavp.showApprovalResults()
2      Approval results
3      Candidate: 'c12' obtains 34 votes
4      Candidate: 'c05' obtains 30 votes
5      Candidate: 'c03' obtains 28 votes
6      Candidate: 'c14' obtains 27 votes
7      Candidate: 'c11' obtains 27 votes
8      Candidate: 'c04' obtains 27 votes
9      Candidate: 'c01' obtains 27 votes
10     Candidate: 'c13' obtains 24 votes
11     Candidate: 'c07' obtains 24 votes
12     Candidate: 'c15' obtains 23 votes
13     Candidate: 'c02' obtains 23 votes
14     Candidate: 'c09' obtains 22 votes
```

```

15     Candidate: 'c08' obtains 22 votes          396
16     Candidate: 'c10' obtains 21 votes          397
17     Candidate: 'c06' obtains 21 votes          398
18 Total approval votes: 380                  399
19 Approval proportion: 380/1500 = 0.25        400
20 >>> bavp.showDisapprovalResults()          401
21 Disapproval results                      402
22     Candidate: 'c12' obtains 16 votes          403
23     Candidate: 'c03' obtains 22 votes          404
24     Candidate: 'c09' obtains 23 votes          405
25     Candidate: 'c04' obtains 24 votes          406
26     Candidate: 'c06' obtains 24 votes          407
27     Candidate: 'c13' obtains 24 votes          408
28     Candidate: 'c11' obtains 25 votes          409
29     Candidate: 'c02' obtains 26 votes          410
30     Candidate: 'c07' obtains 26 votes          411
31     Candidate: 'c08' obtains 26 votes          412
32     Candidate: 'c05' obtains 27 votes          413
33     Candidate: 'c10' obtains 27 votes          414
34     Candidate: 'c14' obtains 27 votes          415
35     Candidate: 'c15' obtains 27 votes          416
36     Candidate: 'c01' obtains 32 votes          417
37 Total disapproval votes: 376                418
38 Disapproval proportion: 376/1500 = 0.25      419

```

In Lines 3 and 22 above, we notice that, of all eligible candidates, it is Candidate 420
 c12 who receives the highest number of approval votes (34) and the lowest number 421
 of disapproval votes (16). The total number of approval (respectively, disapproval) 422
 votes approaches more or less a proportion of 25% of the $100 \times 15 = 1500$ potential 423
 approval votes. About 50% of the latter remain hence ignored. 424

When operating now, for each Candidate x , the difference between the number of 425
 approval and the number of disapproval votes they receive, we obtain per candidate 426
 a corresponding *net approval* score, in fact, the bipolar truth characteristic value of 427
 the statement ‘Candidate x should win the election’. 428

$$r(\text{Candidate } x \text{ should win the election}) = \sum_v (r(x \in A_v)). \quad (20.1)$$

These bipolar characteristic values are stored in the *netApprovalScores* 429
 attribute and may be printed out with the *showNetApprovalScores()* 430
 method: 431

```

1 >>> bavp.showNetApprovalScores()          432
2 Net Approval Scores                      433
3     Candidate: 'c12' obtains 18 net approvals 434
4     Candidate: 'c03' obtains 6 net approvals   435
5     Candidate: 'c05' obtains 3 net approvals   436
6     Candidate: 'c04' obtains 3 net approvals   437
7     Candidate: 'c11' obtains 2 net approvals   438
8     Candidate: 'c14' obtains 0 net approvals   439
9     Candidate: 'c13' obtains 0 net approvals   440
10    Candidate: 'c09' obtains -1 net approvals 441

```

11	Candidate: 'c07' obtains -2 net approvals	442
12	Candidate: 'c06' obtains -3 net approvals	443
13	Candidate: 'c02' obtains -3 net approvals	444
14	Candidate: 'c15' obtains -4 net approvals	445
15	Candidate: 'c08' obtains -4 net approvals	446
16	Candidate: 'c01' obtains -5 net approvals	447
17	Candidate: 'c10' obtains -6 net approvals	448

We observe in Line 3 above that Candidate c_{12} , with a net approval score of $34 - 16 = 18$, represents indeed the *best approved* candidate for winning the election. With a net approval score of $28 - 22 = 6$, Candidate c_03 appears second best approved. The net approval scores define hence a potentially weak ranking on the set of eligible election candidates, and the winner(s) of the election is(are) determined by the first-ranked candidate(s).

20.4 Pairwise Comparison of Approval–Disapproval Votes

455

The approval votes of each voter define now on the set of eligible candidates three ordered categories: his approved (+1), his ignored (0), and his disapproved (-1) ones. Within each of these three categories, we consider the voter's actual preferences as *not communicated*, i.e. as missing data. This gives for each voter a partially determined strict order which we find in the `ballot` attribute.

460

1	>>> bavp.ballot['v001']['c12']	461
2	{'c02': Decimal('1'), 'c11': Decimal('1'),	462
3	'c14': Decimal('1'), 'c04': Decimal('0'),	463
4	'c06': Decimal('1'), 'c05': Decimal('1'),	464
5	'c12': Decimal('0'), 'c13': Decimal('0'),	465
6	'c15': Decimal('1'), 'c01': Decimal('1'),	466
7	'c08': Decimal('1'), 'c07': Decimal('1'),	467
8	'c09': Decimal('0'), 'c03': Decimal('1'),	468
9	'c10': Decimal('0') }	469

For voter v_{001} , for instance, the best approved Candidate c_{12} is strictly preferred to Candidates: $c_01, c_02, c_03, c_05, c_06, c_07, c_{11}, c_{14}$, and c_{15} . No candidate is preferred to c_{12} , and the comparison with c_04, c_09, c_{10} , and c_{13} is not communicated, hence indeterminate. Mind by the way that the reflexive comparison of c_{12} with itself is, as usual, ignored, i.e. indeterminate. Each voter v defines thus a partially determined transitive strict preference relation denoted \succ_v on the eligible candidates.

476

For each pair of eligible candidates, we aggregate the previous individual voter's preferences into a truth characteristic of the statement: 'Candidate x is *better approved than Candidate y* ', denoted $r(x \succ y)$:

477

478

479

$$r(x \succ y) = \sum_v (r(x \succ_v y)). \quad (20.2)$$

AQ1

We say that Candidate x is *better approved than* Candidate y when $r(x > y) > 0$, i.e. there is a majority of voters who approve *more* and disapprove *less* x than y . Vice versa, we say that Candidate x is *not* better approved than Candidate y when $r(x > y) < 0$, i.e. there is a majority of voters who disapprove *more* and approve *less* x than y . This computation is achieved with the MajorityMarginsDigraph constructor.

```

1 >>> from votingProfiles import MajorityMarginsDigraph 486
2 >>> m = MajorityMarginsDigraph(bavp) 487
3 >>> m 488
4     *----- Digraph instance description -----* 489
5     Instance class      : MajorityMarginsDigraph 490
6     Instance name       : rel_bpApVotingProfile 491
7     Digraph Order       : 15 492
8     Digraph Size        : 97 493
9     Valuation domain    : [-100.00;100.00] 494
10    Determinateness (%) : 52.55 495
11    Attributes          : ['name', 'actions', 496
12          'criteria', 'ballot', 497
13          'valuationdomain', 'relation', 498
14          'order', 'gamma', 'notGamma'] 499

```

The resulting digraph m contains 97 positively validated relations (see Line 8 above), and for all pairs (x, y) of eligible candidates, $r(x > y)$ takes value in a valuation range from -100.00 (unanimously rejected) to $+100.00$ (unanimously approved).

These pairwise $r(x > y)$ values can be inspected in a browser view with the showHTMLRelationTable() method:

```

1 >>> m.showHTMLRelationTable(relationName='r(x > y)') 506

```

In Fig. 20.3, it gets apparent that Candidate $c12$ is a CONDORCET winner, i.e. the candidate who beats all the other candidates and, with the given voting profile gavp, should without doubt win the election. This strongly confirms the first-ranked result obtained with the previous net approval scoring.

Let us eventually compute, with the help of the NETFLOWS ranking rule,² a linear ranking of the 15 eligible candidates and compare the result with the net approval scores ranking.

Listing 20.11 Comparing the net approval and the NETFLOWS rankings

```

1 >>> from linearOrders import NetFlowsOrder 514
2 >>> nf = NetFlowsOrder(m, Comments=True) 515
3 >>> print('NetFlows versus Net Approval Ranking') 516
4 >>> print('Candidate\tNetFlows score\tNet Approval score') 517
5 >>> for item in nf.netFlows: 518
6 ...     print( '%9s\t %+.3f\t %+.1f' %\ 519
7 ...             (item[1], item[0],\ 520
8 ...             bavp.netApprovalScores[item[1]])) 521

```

² See Sect. 8.3.

r(x > y)	c01	c02	c03	c04	c05	c06	c07	c08	c09	c10	c11	c12	c13	c14	c15
c01	-	1	-10	-10	-8	-2	-2	-3	-3	2	-5	-19	-4	-2	2
c02	-1	-	-5	-4	-9	-2	5	1	-7	3	-2	-20	-1	-4	-2
c03	10	5	-	1	4	6	5	10	7	11	0	-5	2	6	9
c04	10	4	-1	-	2	5	7	8	-1	4	0	-7	8	2	8
c05	8	9	-4	-2	-	4	6	7	4	2	1	-17	6	0	3
c06	2	2	-6	-5	-4	-	-1	2	-3	5	-4	-13	-1	-1	2
c07	2	-5	-5	-7	-6	1	-	7	-4	2	-5	-17	-2	-2	4
c08	3	-1	-10	-8	-7	-2	-7	-	-2	2	-2	-16	0	0	-1
c09	3	7	-7	1	-4	3	4	2	-	3	0	-18	-4	0	2
c10	-2	-3	-11	-4	-2	-5	-2	-2	-3	-	-6	-15	-4	-4	2
c11	5	2	0	0	-1	4	5	2	0	6	-	-15	4	0	5
c12	19	20	5	7	17	13	17	16	18	15	15	-	12	13	18
c13	4	1	-2	-8	-6	1	2	0	4	4	-4	-12	-	-1	4
c14	2	4	-6	-2	0	1	2	0	0	4	0	-13	-1	-	-1
c15	-2	2	-9	-8	-3	-2	-4	1	-2	-2	-5	-18	-4	1	-

Valuation domain: [-100; +100]

Fig. 20.3 The bipolar-valued pairwise majority margins

NetFlows versus Net Approval Ranking			522
Candidate	NetFlows score	Net Approval score	523
'c12'	+410.000	+18.0	524
'c03'	+142.000	+6.0	525
'c04'	+98.000	+3.0	526
'c05'	+54.000	+3.0	527
'c11'	+34.000	+2.0	528
'c09'	-16.000	-1.0	529
'c14'	-20.000	+0.0	530
'c13'	-22.000	+0.0	531
'c06'	-50.000	-3.0	532
'c07'	-74.000	-2.0	533
'c02'	-96.000	-3.0	534
'c08'	-102.000	-4.0	535
'c15'	-110.000	-4.0	536
'c10'	-122.000	-6.0	537
'c01'	-126.000	-5.0	538

In Listing 20.11 on the preceding page, we may notice that the NETFLOWS rule delivers a ranking that is very similar to the one previously obtained with the corresponding *Net Approval* scores. Only minor inversions do appear, like in the midfield, where Candidate c09 advances before Candidates c13 and c14 (see Line 16), and Candidates c06 and c07 swap their positions 9 and 10 (see Line 19). The two last ranked Candidates c10 and c01 also swap their positions.

This confirms again the pertinence of the net approval scoring approach for 545
 finding the winner in an approval–disapproval voting system. Yet, voting by 546
 approving (+1), disapproving (−1), or ignoring (0) eligible candidates may also be 547
 seen as a performance evaluation of the eligible candidates on a {−1, 0, 1}-valued 548
 ordinal scale. 549

20.5 Three-Valued Evaluative Voting Systems

550

Following such an epistemic perspective, we may effectively convert the given 551
 BipolarApprovalVotingProfile instance into a Performance- 552
 Tableau instance, so as to get access to a corresponding outranking decision 553
 aiding approach. 554

Mind that, contrary to the majority margins of the ‘*better approved than*’ relation, 555
 all voters consider now the approved candidates to be equivalent (+1). The same 556
 is true for the disapproved (−1) and, respectively, the ignored (0) candidates. 557
 The voter’s marginal preferences model this time a complete preorder with three 558
 equivalence classes. 559

From the saved file AVPerfTab.py (see Line 1 below), we may 560
 construct an outranking relation on the eligible candidates with our standard 561
 BipolarOutrankingDigraph class constructor. The semantics of this 562
 outranking relation are the following: 563

- We say that Candidate x *outranks* Candidate y when there is a majority of voters 564
 who consider x *at least as well evaluated as* y . 565
- We say that Candidate x is *outranked by* Candidate y when there is a majority of 566
 voters who consider x *not at least as well evaluated as* y . 567
- Otherwise, the outranking situation is indeterminate. 568

Listing 20.12 Computing the outranking digraph

```

1  >>> bavp.save2PerfTab(fileName='AVPerfTab',valueDigits=0)      569
2  *--- Saving as performance tableau in AVPerfTab.py ---*      570
3  >>> from outrankingDigraphs import\      571
4  ...                           BipolarOutrankingDigraph      572
5  >>> odg = BipolarOutrankingDigraph('AVPerfTab')      573
6  >>> odg      574
7  *----- Object instance description -----*      575
8  Instance class      : BipolarOutrankingDigraph      576
9  Instance name       : rel_AVPerfTab      577
10 Actions            : 15      578
11 Criteria           : 100      579
12 Size               : 210      580
13 Determinateness (%) : 69.29      581
14 Valuation domain   : [-1.00;1.00]      582
15 Attributes          : ['name', 'actions', 'order',      583
16                      'criteria', 'evaluation', 'NA',      584
17                      'valuationdomain', 'relation',      585
18                      'gamma', 'notGamma', ...]      586

```

AQ2

The size ($210 = 15 \times 14$) of the resulting outranking digraph odg , shown in Listing 20.12 on the previous page Line 12 above, reveals that the corresponding ‘at least as good evaluated as’ relation models actually a trivial *complete* digraph. All candidates appear to be *equally* at least as well evaluated and the strict ‘better evaluated than’ (codual) outranking digraph becomes in fact empty. The converted performance tableau does apparently not contain sufficiently discriminatory performance evaluations for supporting any strict preference situations.

Yet, we may nevertheless try to apply again the NETFLOWS ranking rule to this complete outranking digraph odg and print side by side the corresponding NETFLOWS scores and the previous Net Approval scores.

Listing 20.13 Comparing the NETFLOWS and the Net Approval rankings

```

1 >>> from linearOrders import NetFlowsOrder
2 >>> nf = NetFlowsOrder (odg)
3 >>> print ('NetFlows versus Net Approval Ranking')
4 >>> print ('Candidate\tNetFlows Score\tNet Approval Score')
5 >>> for item in nf.netFlows:
6 ...     print ('%9s\t %+.3f\t %+.0f' %
7 ...           (item[1],item[0],\
8 ...             bavp.netApprovalScores [item[1]] ) )
9 NetFlows versus Net Approval Ranking
10 Candidate    NetFlows score Net Approval score
11   c12        +4.100        +18
12   c03        +1.420         +6
13   c04        +0.980         +3
14   c05        +0.540         +3
15   c11        +0.340         +2
16   c09        -0.160        -1
17   c14        -0.200         0
18   c13        -0.220         0
19   c06        -0.500        -3
20   c07        -0.740        -2
21   c02        -0.960        -3
22   c08        -1.020        -4
23   c15        -1.100        -4
24   c10        -1.220        -6
25   c01        -1.260        -5

```

Despite its apparent poor strict preference discriminating power, we obtain in Listing 20.13 here NETFLOWS scores that are directly proportional (divided by 100) to the scores obtained with the *better approved than* majority margins digraph m (see Listing 20.11 on page 291).

Encouraged by this positive result, we may furthermore compute a RUBIS best choice recommendation (see Chap. 4).

Listing 20.14 Computing a best social choice recommendation

```

1 >>> odg.showBestChoiceRecommendation()
2   Rubis best choice recommendation(s) (BCR)
3   (in decreasing order of determinateness)

```

```

4   Credibility domain: [-1.00,1.00]                                631
5   === >> ambiguous first choice(s)                                632
6   * choice      : ['c01','c02','c03','c04','c05',                  633
7           'c06','c07','c08','c09','c10',                            634
8           'c11','c12','c13','c14','c15']                            635
9   independence    : 0.06                                         636
10  dominance      : 1.00                                         637
11  absorbency     : 1.00                                         638
12  covering (%)   : 100.00                                       639
13  determinateness (%) : 61.13                                       640
14  - most credible action(s) = {                                641
15      'c12': 0.44, 'c03': 0.34, 'c04': 0.30,                      642
16      'c14': 0.28, 'c13': 0.24, 'c06': 0.24,                      643
17      'c11': 0.20, 'c10': 0.20, 'c07': 0.20,                      644
18      'c01': 0.20, 'c08': 0.18, 'c05': 0.18,                      645
19      'c15': 0.14, 'c09': 0.14, 'c02': 0.06, }                    646
20  === >> ambiguous last choice(s)                                647
21  * choice      : ['c01','c02','c03','c04','c05',                  648
22           'c06','c07','c08','c09','c10',                            649
23           'c11','c12','c13','c14','c15']                            650
24  independence    : 0.06                                         651
25  dominance      : 1.00                                         652
26  absorbency     : 1.00                                         653
27  covered (%)   : 100.00                                       654
28  determinateness (%) : 63.73                                       655
29  - most credible action(s) = {                                656
30      'c13': 0.36, 'c06': 0.36, 'c15': 0.34,                      657
31      'c01': 0.34, 'c08': 0.32, 'c07': 0.30,                      658
32      'c02': 0.30, 'c14': 0.28, 'c11': 0.28,                      659
33      'c09': 0.28, 'c04': 0.26, 'c10': 0.24,                      660
34      'c05': 0.20, 'c03': 0.20, 'c12': 0.06, }                    661

```

The strict outranking digraph (\sim ($-odg$)) being actually *empty*, we obtain a 662 unique *ambiguous* – first as well as last – choice recommendation which trivially 663 retains all fifteen candidates (see Listing 20.14 on the facing page, Lines 6–8 above). 664 Yet, the bipolar-valued best choice membership characteristic vector reveals that, 665 among all the fifteen potential winners, it is indeed Candidate c_{12} the most credible 666 one with a 72% majority of voters' support (see Line 15, $(0.44 + 1.0)/2 = 0.72$), 667 followed by Candidate c_{03} (67%) and Candidate c_{04} (65%). Similarly, Candidates 668 c_{13} and c_{06} represent the most credible losers with a 68% majority voters' support 669 (Line 30). 670

We observe here empirically that *evaluative* voting systems, using three-valued 671 ordinal performance scales, match closely corresponding approval–disapproval vot- 672 ing systems. The latter systems model, however, more faithfully the very preferential 673 information that is expressed with *approved*, *disapproved*, and *ignored* statements. 674 The corresponding evaluation on a three-level scale, being value (numbers) based, 675 cannot express the fact that in an approval–disapproval voting system there is no 676 preferential information given concerning the pairwise comparison of all approved 677 and, respectively, disapproved or ignored candidates. 678

Let us finally illustrate how approval–disapproval voting systems may favour 679
 multipartisan supported candidates. We shall therefore compare *approval–* 680
disapproval versus *uninominal plurality* election results when considering a highly 681
 divisive and partisan political context. 682

20.6 Favouring Multipartisan Candidates

683

In modern democracy, politics are largely structured by political parties and 684
 activists. Let us so consider an approval–disapproval voting profile *dvp* where 685
 the random voter behaviour is simulated from two pre-electoral polls concerning 686
 a political scene with essentially two major highly competing parties, like the one 687
 existing in the USA. 688

Listing 20.15 A random approval–disapproval voting profile in a divisive political context

```

1  >>> dvp = RandomBipolarApprovalVotingProfile(\ 689
2  ...           numberofCandidates=15,\ 690
3  ...           numberofVoters=100,\ 691
4  ...           approvalProbability=0.25,\ 692
5  ...           disapprovalProbability=0.25,\ 693
6  ...           WithPolls=True,\ 694
7  ...           partyRepartition=0.5,\ 695
8  ...           other=0.05,\ 696
9  ...           DivisivePolitics=True,\ 697
10 ...           seed=200) 698
11 >>> dvp.showRandomPolls() 699
12 Random repartition of voters 700
13 Party-1 supporters : 45 (45.00%) 701
14 Party-2 supporters : 49 (49.00%) 702
15 Other voters : 6 (06.00%) 703
16 *----- random polls ----- 704
17 Party-1(45.0%) | Party-2(49.0%) | expected 705
18 -----
19 'c05' : 24.10% | 'c07' : 24.10% | 'c07' : 11.87% 707
20 'c14' : 23.48% | 'c10' : 23.48% | 'c10' : 11.60% 708
21 'c03' : 15.13% | 'c01' : 15.13% | 'c05' : 10.91% 709
22 'c12' : 07.55% | 'c04' : 07.55% | 'c14' : 10.67% 710
23 'c08' : 07.11% | 'c09' : 07.11% | 'c01' : 07.67% 711
24 'c15' : 04.37% | 'c13' : 04.37% | 'c03' : 07.09% 712
25 'c11' : 03.99% | 'c02' : 03.99% | 'c04' : 04.55% 713
26 'c06' : 03.80% | 'c06' : 03.80% | 'c09' : 04.49% 714
27 'c02' : 02.79% | 'c11' : 02.79% | 'c12' : 04.32% 715
28 'c13' : 02.63% | 'c15' : 02.63% | 'c08' : 04.30% 716
29 'c09' : 02.24% | 'c08' : 02.24% | 'c06' : 03.57% 717
30 'c04' : 01.89% | 'c12' : 01.89% | 'c13' : 03.32% 718
31 'c01' : 00.57% | 'c03' : 00.57% | 'c15' : 03.25% 719
32 'c10' : 00.20% | 'c14' : 00.20% | 'c02' : 03.21% 720
33 'c07' : 00.14% | 'c05' : 00.14% | 'c11' : 03.16% 721

```

In Listing 20.15 on the preceding page, the divisive political situation is reflected 722
 by the fact that Party-1 and Party-2 supporters show strict opposite preferences. 723
 The leading candidates of Party-1 (c05 and c14) are last choices for Party-2 724
 supporters, and Candidates c07 and c10, leading candidates for Party-2 supporters, 725
 are similarly the least choices for Party-1 supporters. 726

No clear winner may be guessed from these pre-election polls. As Party-2 shows 727
 however slightly more supporters than Party-1, the expected winner in a uninominal 728
 plurality or instant-run-off voting system will be Candidate c07, i.e. the leading 729
 candidate of majority Party-2 (see below). 730

```
1 >>> dvp.computeSimpleMajorityWinner () 731
2     ['c07'] 732
3 >>> dvp.computeInstantRunoffWinner () 733
4     ['c07'] 734
```

Now, in a corresponding approval-disapproval voting system, Party-1 supporters 735
 will usually approve their leading candidates and disapprove the leading candidates 736
 of Party-2. Vice versa, Party-2 supporters will usually approve their leading 737
 candidates and disapprove the leading candidates of Party-1. Let us consult the 738
 resulting approval votes per candidate. 739

```
1 >>> dvp.showApprovalResults () 740
2     Candidate: 'c07' obtains 30 votes 741
3     Candidate: 'c10' obtains 28 votes 742
4     Candidate: 'c05' obtains 28 votes 743
5     Candidate: 'c01' obtains 28 votes 744
6     Candidate: 'c03' obtains 26 votes 745
7     Candidate: 'c02' obtains 26 votes 746
8     Candidate: 'c12' obtains 25 votes 747
9     Candidate: 'c14' obtains 24 votes 748
10    Candidate: 'c13' obtains 24 votes 749
11    Candidate: 'c09' obtains 21 votes 750
12    Candidate: 'c04' obtains 21 votes 751
13    Candidate: 'c08' obtains 19 votes 752
14    Candidate: 'c06' obtains 17 votes 753
15    Candidate: 'c15' obtains 15 votes 754
16    Candidate: 'c11' obtains 12 votes 755
17    Total approval votes: 344 756
18    Approval proportion: 344/1500 = 0.23 757
```

When considering only the approval votes, we find confirmed above that the 758
 leading candidate of Party-2 obtains in this simulation a plurality of approval 759
 votes. In uninominal plurality or instant-run-off voting systems, this candidate wins 760
 hence the election, quite to the despair of Party-1 supporters. As a foreseeable 761
 consequence, this election result will be more or less aggressively contested which 762
 leads to a loss of popular trust in democratic elections and institutions. 763

If we look however on the corresponding disapprovals, we discover that, not 764
 surprisingly, the leading candidates of both parties collect by far the highest number 765
 of disapproval votes. 766

```

1 >>> dvp.showDisapprovalResults() 767
2     Candidate: 'c02' obtains 14 votes 768
3     Candidate: 'c04' obtains 14 votes 769
4     Candidate: 'c13' obtains 14 votes 770
5     Candidate: 'c06' obtains 15 votes 771
6     Candidate: 'c09' obtains 15 votes 772
7     Candidate: 'c08' obtains 16 votes 773
8     Candidate: 'c11' obtains 16 votes 774
9     Candidate: 'c15' obtains 18 votes 775
10    Candidate: 'c12' obtains 20 votes 776
11    Candidate: 'c01' obtains 29 votes 777
12    Candidate: 'c03' obtains 30 votes 778
13    Candidate: 'c10' obtains 37 votes 779
14    Candidate: 'c07' obtains 44 votes 780
15    Candidate: 'c14' obtains 45 votes 781
16    Candidate: 'c05' obtains 49 votes 782
17    Total disapproval votes: 376 783
18    Disapproval proportion: 376/1500 = 0.25 784

```

Balancing now approval against disapproval votes will favour the moderate, bipartisan supported, candidates. 785
786

```

1 >>> dvp.showNetApprovalScores() 787
2     Net Approval Scores 788
3     Candidate: 'c02' obtains 12 net approvals 789
4     Candidate: 'c13' obtains 10 net approvals 790
5     Candidate: 'c04' obtains 7 net approvals 791
6     Candidate: 'c09' obtains 6 net approvals 792
7     Candidate: 'c12' obtains 5 net approvals 793
8     Candidate: 'c08' obtains 3 net approvals 794
9     Candidate: 'c06' obtains 2 net approvals 795
10    Candidate: 'c01' obtains -1 net approvals 796
11    Candidate: 'c15' obtains -3 net approvals 797
12    Candidate: 'c11' obtains -4 net approvals 798
13    Candidate: 'c03' obtains -4 net approvals 799
14    Candidate: 'c10' obtains -9 net approvals 800
15    Candidate: 'c07' obtains -14 net approvals 801
16    Candidate: 'c14' obtains -21 net approvals 802
17    Candidate: 'c05' obtains -21 net approvals 803

```

Candidate c02, appearing in the pre-electoral polls in the midfield (in position 7 for Party-2 and in position 9 for Party-1 supporters, see Listing 20.15 on page 296), shows indeed the highest net approval score. The second highest net approval score obtains Candidate c13, in position 6 for Party-2 and in position 10 for Party-1 supporters. 804
805
806
807
808

Figure 20.4 on the next page, showing the NETFLOWS ranked relation table of the ‘better approved than’ majority margins digraph, confirms below this net approval scoring result. 809
810
811

```

1 >>> m = MajorityMarginsDigraph(dvp) 812
2 >>> m.showHTMLRelationTable(\ 813
3 ...     actionsList=m.computeNetFlowsRanking(), 814
4 ...     relationName='r(x > y)') 815

```

$r(x > y)$	c02	c13	c04	c09	c12	c08	c06	c01	c11	c15	c03	c10	c07	c14	c05
c02	-	6	5	6	9	5	10	12	12	14	11	15	22	22	23
c13	-6	-	2	5	2	5	8	10	14	10	9	13	18	23	20
c04	-5	-2	-	0	2	2	3	7	7	8	11	13	18	21	18
c09	-6	-5	0	-	0	2	5	5	11	9	5	13	16	21	16
c12	-9	-2	-2	0	-	4	6	2	9	6	10	5	10	23	25
c08	-5	-5	-2	-2	-4	-	4	0	8	9	7	5	11	21	20
c06	-10	-8	-3	-5	-6	-4	-	-2	5	5	5	6	13	17	18
c01	-12	-10	-7	-5	-2	0	2	-	1	-1	3	8	11	9	13
c11	-12	-14	-7	-11	-9	-8	-5	-1	-	1	-2	7	14	13	16
c15	-14	-10	-8	-9	-6	-9	-5	1	-1	-	0	3	10	14	14
c03	-11	-9	-11	-5	-10	-7	-5	-3	2	0	-	-3	7	16	16
c10	-15	-13	-13	-13	-5	-5	-6	-8	-7	-3	3	-	3	6	7
c07	-22	-18	-18	-16	-10	-11	-13	-11	-14	-10	-7	-3	-	3	5
c14	-22	-23	-21	-21	-23	-21	-17	-9	-13	-14	-16	-6	-3	-	0
c05	-23	-20	-18	-16	-25	-20	-18	-13	-16	-14	-16	-7	-5	0	-

Valuation domain: [-100; +100]

Fig. 20.4 The pairwise *better approved than* majority margins

Candidate c02 appears indeed *better approved than* any other candidate (CONDORCET winner), and the leading candidates of Party-1, c05 and c14, are *less approved than* any other candidates (weak CONDORCET losers).

```

1 >>> m.computeCondorcetWinners()
2   ['c02']
3 >>> m.computeWeakCondorcetLosers()
4   ['c05', 'c14']

```

We see this result furthermore confirmed when computing the corresponding first and, respectively, last choice recommendations.

```

1 >>> m.showBestChoiceRecommendation()
2 Rubis best choice recommendation(s) (BCR)
3   (in decreasing order of determinateness)
4 Credibility domain: [-100.00,100.00]
5 === >> potential first choice(s)
6   * choice          : ['c02']
7   independence      : 100.00
8   dominance         : 5.00
9   absorbency        : -23.00
10  covering (%)     : 100.00
11  determinateness (%) : 52.50
12  - most credible action(s) = { 'c02': 5.00, }
13 === >> potential last choice(s)
14   * choice          : ['c05', 'c14']
15   independence      : 0.00

```

AQ3

16	dominance	:	-23.00	840
17	absorbency	:	5.00	841
18	covered (%)	:	100.00	842
19	determinateness (%)	:	50.00	843
20	- most credible action(s) = { }			844

Candidate c02, being actually a CONDORCET winner, gives an initial prekernel of digraph m , whereas Party-1 leading Candidates c05 and c14, both being weak CONDORCET losers, give together a terminal prekernel. They hence represent our *first choice* and, respectively, *last choice* recommendations for winning this simulated election.

It is worthwhile noticing again the essential structural and computational role, the ignored value is playing in approval-disapproval voting systems. This epistemic and logical *neutral* term is needed indeed for handling in a consistent and efficient manner *not communicated votes* and/or *indeterminate* preferential statements.

Let us conclude by predicting that, for leading political candidates in an aggressively divisive political context, the perspective to easily fail election with an approval-disapproval voting system might or will induce a change in the usual way of running electoral campaigns. Political parties and politicians, who avoid aggressive competitive propaganda and instead propose multipartisan collaborative social choices, will be rewarded with better election results than any kind of extremism. It could mean the end of sterile political obstructions and war like electoral battles. *Let us do it!*

The last part, Part V, of this monograph illustrates in three chapters computational resources for working with simple undirected graphs.

References

864

Baujard A, Gavrel F, Iggersheim H, Laslier JF, Lebon I (2013) Approval voting, evaluation voting: an experiment during the 2012 French presidential election. <i>Revue Économique</i> (Presses de Sciences Po) 64(2):345–356	865
Benayoun R, Roy B, Sussmann B (1966) Electre: une méthode pour guider le choix en présence de points de vue multiples. Tech. Rep. 49, Société d'Economie et de Mathématique Appliquée, Direction Scientifique	866
	867
	868
	869
	870

AUTHOR QUERIES

- AQ1.** Please check the construct “his approved (+1), his ignored (0), and his disapproved ...” for gender neutrality and amend if required.
- AQ2.** Please check if the sentence “The same is true ...” is fine as edited and amend if required.
- AQ3.** Please check if the sentence “We see this result furthermore ...” is fine as edited and amend if required.

Uncorrected Proof

Part V ₁ Working with Undirected Graphs ₂

The last part introduces Python resources for working with undirected graphs. Its ₃ aim is to eventually show the operational benefits one may get when implementing ₄ vertex adjacency with bipolar-valued characteristics. Several special graph models ₅ and algorithms are illustrated: Q-coloring, MIS and clique enumeration, line graphs ₆ and computing maximal matchings, grid graphs and computing the Ising model, ₇ n -cycle graphs and computing their non-isomorphic MISs, spanning tree graphs ₈ and graph forests, and generating and recognising split, interval, and permutation ₉ graphs. ₁₀

Chapter 21

Bipolar-Valued Undirected Graphs

1

2

Contents

21.1	Implementing Simple Graphs	303	4
21.2	Q-Coloring of a Graph	306	5
21.3	MIS and Clique Enumeration	307	6
21.4	Line Graphs and Maximal Matchings	309	7
21.5	Grids and the ISING Model	311	8
21.6	Simulating METROPOLIS Random Walks	311	9
21.7	Computing the Non-isomorphic MISs of the n -Cycle Graph	313	10

Abstract The chapter introduces bipolar-valued undirected graphs and illustrates 11
several special graph models and algorithms like Q-coloring, MIS and clique 12
enumeration, line graphs and maximal matchings, grid graphs, and n -cycle graphs 13
with their non-isomorphic maximal independent sets of vertices. 14

21.1 Implementing Simple Graphs

15

In the DIGRAPH3 graphs module, the root `Graph` class provides a generic simple 16
graph model, without loops and multiple links. A given object of this `Graph` type 17
contains at least the following attributes: 18

1. `name`: usually the name of the stored instance 19
2. `vertices`: a dictionary with `name` and `shortName` attributes 20
3. `order`: the number of vertices 21
4. `valuationDomain`: a dictionary with three entries: the minimum (-1 means 22
certainly no link), the median (0 means missing information), and the maximum 23
characteristic value ($+1$ means certainly a link) 24
5. `edges`: a dictionary with `frozensets`¹ of pairs of vertices as keys carrying a 25
characteristic value in the range of the previous valuation domain 26

¹ [Python documentation](#).

6. size: the number of positive edges	27
7. gamma: a dictionary containing the direct neighbours of each vertex, automatically added by the object constructor	28
	29

Example Python Terminal Session

A random crisp graph instance can be generated with the RandomGraph class from the graphs module (see below).

Listing 21.1 Generating a random graph instance

```

1 >>> from graphs import RandomGraph
2 >>> g = RandomGraph(order=7, edgeProbability=0.5)
3 >>> g.save(fileName='tutorialGraph')

```

The saved Graph instance, named tutorialGraph.py, is encoded as shown in Listing 21.2.

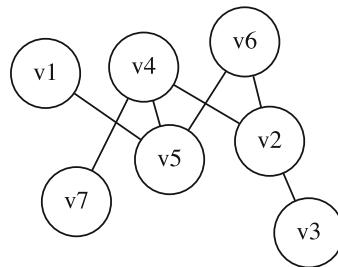
Listing 21.2 Stored instance of a random graph

```

1 # Graph instance saved in Python format
2 from decimal import Decimal
3 vertices = {
4     'v1': {'shortName': 'v1', 'name': 'random vertex'},
5     'v2': {'shortName': 'v2', 'name': 'random vertex'},
6     'v3': {'shortName': 'v3', 'name': 'random vertex'},
7     'v4': {'shortName': 'v4', 'name': 'random vertex'},
8     'v5': {'shortName': 'v5', 'name': 'random vertex'},
9     'v6': {'shortName': 'v6', 'name': 'random vertex'},
10    'v7': {'shortName': 'v7', 'name': 'random vertex'},
11 }
12 valuationDomain = {
13     'min':Decimal('-1'), 'med':Decimal('0'),
14     'max':Decimal('1')}
15 edges = {
16     frozenset(['v1','v2']) : Decimal('-1'),
17     frozenset(['v1','v3']) : Decimal('-1'),
18     frozenset(['v1','v4']) : Decimal('-1'),
19     frozenset(['v1','v5']) : Decimal('1'),
20     frozenset(['v1','v6']) : Decimal('-1'),
21     frozenset(['v1','v7']) : Decimal('-1'),
22     frozenset(['v2','v3']) : Decimal('1'),
23     frozenset(['v2','v4']) : Decimal('1'),
24     frozenset(['v2','v5']) : Decimal('-1'),
25     frozenset(['v2','v6']) : Decimal('1'),
26     frozenset(['v2','v7']) : Decimal('-1'),
27     frozenset(['v3','v4']) : Decimal('-1'),
28     frozenset(['v3','v5']) : Decimal('-1'),
29     frozenset(['v3','v6']) : Decimal('-1'),
30     frozenset(['v3','v7']) : Decimal('-1'),
31     frozenset(['v4','v5']) : Decimal('1'),
32     frozenset(['v4','v6']) : Decimal('-1'),
33     frozenset(['v4','v7']) : Decimal('1'),
34     frozenset(['v5','v6']) : Decimal('1'),
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

```

Fig. 21.1 Example simple graph instance



Digraph3 (graphviz), R. Bisdorff, 2019

```

35     frozenset(['v5','v7']) : Decimal('-1'),
36     frozenset(['v6','v7']) : Decimal('-1'),
37 }
```

The stored graph instance may be reloaded and plotted with the `exportGraphViz()` method (see Fig. 21.1):

```

1 >>> g = Graph('tutorialGraph') 77
2 >>> g.exportGraphViz() 78
3     *---- exporting a dot file for GraphViz tools -----* 79
4     Exporting to tutorialGraph.dot 80
5     fdp -Tpng tutorialGraph.dot -o tutorialGraph.png 81
  
```

Properties like the gamma function—the cover relation—, vertex degrees, and neighbourhood depths are shown with a `showShort()` method.

Listing 21.3 Inspecting a graph instance

```

1 >>> g.showShort() 84
2     *---- short description of the graph -----* 85
3     Name : 'tutorialGraph' 86
4     Vertices : ['v1','v2','v3','v4','v5','v6','v7'] 87
5     Valuation domain : {'min': -1, 'med': 0, 'max': 1} 88
6     Gamma function : 89
7         v1 -> ['v5'] 90
8         v2 -> ['v6', 'v4', 'v3'] 91
9         v3 -> ['v2'] 92
10        v4 -> ['v5', 'v2', 'v7'] 93
11        v5 -> ['v1', 'v6', 'v4'] 94
12        v6 -> ['v2', 'v5'] 95
13        v7 -> ['v4'] 96
14     degrees : [0, 1, 2, 3, 4, 5, 6] 97
15     distribution : [0, 3, 1, 3, 0, 0, 0] 98
16     nbh depths : [0, 1, 2, 3, 4, 5, 6, 'inf.'] 99
17     distribution : [0, 0, 1, 4, 2, 0, 0, 0] 100
  
```

A `Graph` instance corresponds bijectively to a symmetric `Digraph` instance, and we can convert from one to the other with the `graph2Digraph()`, and

vice versa, with the `digraph2Graph()` method (see Listing 21.4). Thus, all computing resources of the `Digraph` class suitable for symmetric digraphs become readily available, and vice versa. 103
104
105

Listing 21.4 Conversion between graphs and digraphs

```

1 >>> dg = g.graph2Digraph() 106
2 >>> dg.showRelationTable(ndigits=0,ReflexiveTerms=False) 107
3 * ---- Relation Table ---- 108
4   S | 'v1' 'v2' 'v3' 'v4' 'v5' 'v6' 'v7' 109
5   ---|----- 110
6   'v1' | -1 -1 -1 1 -1 -1 111
7   'v2' | -1 1 -1 1 -1 1 -1 112
8   'v3' | -1 1 -1 -1 -1 1 -1 113
9   'v4' | -1 1 -1 -1 1 -1 1 114
10  'v5' | 1 -1 -1 1 -1 1 -1 115
11  'v6' | -1 1 -1 -1 1 -1 -1 116
12  'v7' | -1 -1 -1 1 -1 -1 - 117
13 >>> g1 = dg.digraph2Graph() 118
14 >>> g1 119
15 *----- Digraph instance description -----* 120
16 Instance class : Digraph 121
17 Instance name : tutorialGraph 122
18 Digraph Order : 7 123
19 Digraph Size : 14 124
20 Valuation domain : [-1.00;1.00] 125
21 Determinateness (%) : 100.00 126
22 Attributes : ['name', 'order', 'actions', 127
23 : 'valuationdomain', 'relation', 128
24 : 'gamma', 'notGamma'] 129

```

21.2 Q-Coloring of a Graph

130

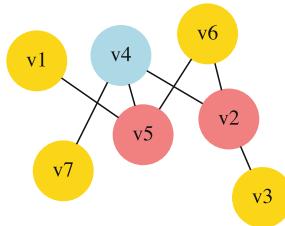
With the `Q_Coloring` class, a 3-coloring of the tutorial graph `g` may be computed with a Gibbs sampler (Geman and Geman 1984) and plotted as shown in Listing 21.5. 131
132
133

Listing 21.5 Computing a 3-coloring of the random graph `g`

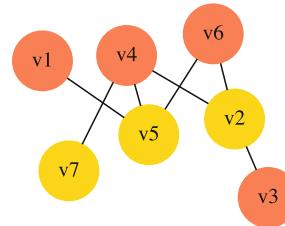
```

1 >>> from graphs import Q_Coloring 134
2 >>> qc = Q_Coloring(g) 135
3   Running a Gibbs Sampler for 42 step ! 136
4   The q-coloring with 3 colors is feasible !! 137
5 >>> qc.showConfiguration() 138
6   v5 lightblue 139
7   v3 gold 140
8   v7 gold 141
9   v2 lightblue 142
10  v4 lightcoral 143

```



Digraph3 (graphviz), R. Bisdorff, 2019



Digraph3 (graphviz), R. Bisdorff, 2019

Fig. 21.2 3-Coloring and 2-coloring of the tutorial graph

```

11  v1 gold
12  v6 lightcoral
13  >>> qc.exportGraphViz('tutorial-3-coloring')
14  *---- exporting a dot file for GraphViz tools
15  Exporting to tutorial-3-coloring.dot
16  fdp -Tpng tutorial-3-coloring.dot\
17          -o tutorial-3-coloring.png

```

Actually, with the given tutorial graph instance, a 2-coloring is already feasible (see Fig. 21.2).

```

1  >>> qc = Q_Coloring(g,colors=['gold','coral'])
2  Running a Gibbs Sampler for 42 step !
3  The q-coloring with 2 colors is feasible !!
4  >>> qc.showConfiguration()
5  v5 gold
6  v3 coral
7  v7 gold
8  v2 gold
9  v4 coral
10 v1 coral
11 v6 coral
12 >>> qc.exportGraphViz('tutorial-2-coloring')
13 Exporting to tutorial-2-coloring.dot
14 fdp -Tpng tutorial-2-coloring.dot\
15          -o tutorial-2-coloring.png

```

21.3 MIS and Clique Enumeration

168

2-colorings define sets of independent vertices that are maximal in cardinality—the MISs. The `showMIS()` method computes and prints out such sets. The result is stored in a `misset` attribute (see Listing 21.6)

169

170

171

Listing 21.6 Computing and printing the maximal independent sets of graph `g`

```

1 >>> g = Graph('tutorialGraph')                                172
2 >>> g.showMIS()                                         173
3     *--- Maximal Independent Sets ---*                      174
4     ['v2', 'v5', 'v7']                                     175
5     ['v3', 'v5', 'v7']                                     176
6     ['v1', 'v2', 'v7']                                     177
7     ['v1', 'v3', 'v6', 'v7']                                178
8     ['v1', 'v3', 'v4', 'v6']                                179
9     number of solutions: 5                                180
10    cardinality distribution                            181
11    card.: [0, 1, 2, 3, 4, 5, 6, 7]                      182
12    freq.: [0, 0, 0, 3, 2, 0, 0, 0]                      183
13    execution time: 0.00032 sec.                         184
14    Results in self.misset                                185
15 >>> g.misset                                         186
16     [frozenset({'v7', 'v2', 'v5'}),                      187
17     frozenset({'v3', 'v7', 'v5'}),                      188
18     frozenset({'v1', 'v2', 'v7'}),                      189
19     frozenset({'v1', 'v6', 'v7', 'v3'}),                190
20     frozenset({'v1', 'v6', 'v4', 'v3'})]                  191

```

A MIS in the dual `-g` of a graph instance `g` corresponds to a maximal *clique*, i.e. 192
 a maximal complete subgraph in `g`. Maximal cliques may be computed and printed 193
 with the `showCliques()` method. The result is stored in a `cliques` attribute as 194
 shown in Listing 21.7. 195

Listing 21.7 Computing and printing the maximal independent sets of graph `g`

```

1 >>> g.showCliques()                                         196
2     *--- Maximal Cliques ---*                           197
3     ['v2', 'v3']                                         198
4     ['v4', 'v7']                                         199
5     ['v2', 'v4']                                         200
6     ['v4', 'v5']                                         201
7     ['v1', 'v5']                                         202
8     ['v2', 'v6']                                         203
9     ['v5', 'v6']                                         204
10    number of solutions: 7                                205
11    cardinality distribution                            206
12    card.: [0, 1, 2, 3, 4, 5, 6, 7]                      207
13    freq.: [0, 0, 7, 0, 0, 0, 0, 0]                      208
14    execution time: 0.00049 sec.                         209
15    Results in self.cliques                            210
16 >>> g.cliques                                         211
17     [frozenset({'v2', 'v3'}), frozenset({'v4', 'v7'}), 212
18     frozenset({'v2', 'v4'}), frozenset({'v4', 'v5'}), 213
19     frozenset({'v1', 'v5'}), frozenset({'v6', 'v2'}), 214
20     frozenset({'v6', 'v5'})]                            215

```

21.4 Line Graphs and Maximal Matchings

216

The graphs module also provides a LineGraph constructor. A *line graph* represents the adjacencies between edges of the given graph instance. In Listing 21.8, we compute, for instance, the line graph of the 5-cycle graph.

Listing 21.8 Computing the line graph of the 5-cycle graph

```

1  >>> from graphs import CycleGraph, LineGraph
2  >>> g = CycleGraph(order=5)
3  >>> g
4  *----- Graph instance description -----*
5  Instance class    : CycleGraph
6  Instance name     : cycleGraph
7  Graph Order       : 5
8  Graph Size        : 5
9  Valuation domain  : [-1.00, 1.00]
10 Attributes        : ['name', 'order',
11                  'vertices', 'valuationDomain',
12                  'edges', 'size', 'gamma']
13 >>> lg = LineGraph(g)
14 >>> lg
15 *----- Graph instance description -----*
16 Instance class    : LineGraph
17 Instance name     : line-cycleGraph
18 Graph Order       : 5
19 Graph Size        : 5
20 Valuation domain  : [-1.00, 1.00]
21 Attributes        : ['name', 'graph',
22                  'valuationDomain', 'vertices',
23                  'order', 'edges', 'size', 'gamma']
24 >>> lg.showShort()
25 *---- short description of the graph ----*
26 Name              : 'line-cycleGraph'
27 Vertices          : [frozenset({'v1', 'v2'}),
28                  frozenset({'v1', 'v5'}), frozenset({'v2', 'v3'}),
29                  frozenset({'v3', 'v4'}), frozenset({'v4', 'v5'})]
30 Valuation domain  : {'min': Decimal('-1'),
31                  'med': Decimal('0'), 'max': Decimal('1')}
32 Gamma function    :
33 frozenset({'v1', 'v2'}) ->
34      [frozenset({'v2', 'v3'}), frozenset({'v1', 'v5'})]
35 frozenset({'v1', 'v5'}) ->
36      [frozenset({'v1', 'v2'}), frozenset({'v4', 'v5'})]
37 frozenset({'v2', 'v3'}) ->
38      [frozenset({'v1', 'v2'}), frozenset({'v3', 'v4'})]
39 frozenset({'v3', 'v4'}) ->
40      [frozenset({'v2', 'v3'}), frozenset({'v4', 'v5'})]
41 frozenset({'v4', 'v5'}) ->
42      [frozenset({'v4', 'v3'}), frozenset({'v1', 'v5'})]
43 degrees           : [0, 1, 2, 3, 4]
44 distribution       : [0, 0, 5, 0, 0]
45 nbh depths        : [0, 1, 2, 3, 4, 'inf.']
46 distribution       : [0, 0, 5, 0, 0, 0]

```

Iterated line graph constructions are usually expanding, except for chordless cycles, where the same cycle is repeated, and for non-closed paths, where iterated line graphs progressively reduce one by one the number of vertices and edges and eventually become an empty graph.

Notice in Listing 21.9 that the MISs in a line graph provide *maximal matchings*—maximal sets of independent edges—of the original graph.

Listing 21.9 Computing the MISs of the line graph of the 8-cycle graph

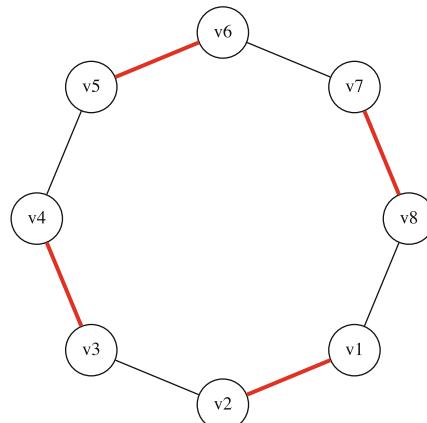
```

1 >>> c8 = CycleGraph(order=8)          272
2 >>> lc8 = LineGraph(c8)            273
3 >>> lc8.showMIS()                 274
4     *--- Maximal Independent Sets ---* 275
5     [frozenset({'v3', 'v4'}), frozenset({'v5', 'v6'}), frozenset({'v1', 'v8'})] 276
6     [frozenset({'v2', 'v3'}), frozenset({'v5', 'v6'}), frozenset({'v1', 'v8'})] 277
7     [frozenset({'v8', 'v7'}), frozenset({'v2', 'v3'}), frozenset({'v4', 'v5'})] 278
8     [frozenset({'v8', 'v7'}), frozenset({'v2', 'v3'}), frozenset({'v1', 'v8'})] 279
9     [frozenset({'v7', 'v6'}), frozenset({'v3', 'v4'}), frozenset({'v1', 'v8'})] 280
10    [frozenset({'v2', 'v1'}), frozenset({'v8', 'v7'}), frozenset({'v4', 'v5'})] 281
11    [frozenset({'v2', 'v1'}), frozenset({'v7', 'v6'}), frozenset({'v4', 'v5'})] 282
12    [frozenset({'v2', 'v1'}), frozenset({'v7', 'v6'}), frozenset({'v3', 'v4'})] 283
13    [frozenset({'v7', 'v6'}), frozenset({'v2', 'v3'}), frozenset({'v1', 'v8'})] 284
14        'v8'),
15        frozenset({'v4', 'v5'}))          285
16    [frozenset({'v2', 'v1'}), frozenset({'v8', 'v7'}), frozenset({'v3', 'v4'})], 286
17        frozenset({'v5', 'v6'}))          287
18    number of solutions: 10            288
19    cardinality distribution        289
20    card.: [0, 1, 2, 3, 4, 5, 6, 7, 8] 290
21    freq.: [0, 0, 0, 8, 2, 0, 0, 0, 0] 291
22    execution time: 0.00029 sec.      292
23

```

The two last MISs of cardinality 4 (see Lines 13–16 13-16 in Listing 21.9) give perfect maximum matchings of the 8-cycle graph as shown in Fig. 21.3.

Fig. 21.3 A perfect maximum matching of the 8-cycle graph
Every vertex of the 8-cycle is adjacent to a matching edge.
Odd cycle graphs do not admit any perfect matching



Digraph3 (graphviz), R. Bisdorff, 2019

Listing 21.10 Computing maximum matchings in the 8-cycle graph

```
1 >>> maxMatching = c8.computeMaximumMatching() 296
2 >>> c8.exportGraphViz(fileName='maxMatchingcycleGraph', \ 297
3 ... 298
4     matching=maxMatching) 299
5
6 *---- exporting a dot file for GraphViz tools ----* 300
7 Exporting to maxMatchingcycleGraph.dot 301
8 Matching: {frozenset({'v1', 'v2'}), 302
9     frozenset({'v5', 'v6'}), 303
10    frozenset({'v3', 'v4'}), 304
11    frozenset({'v7', 'v8'}) } 305
12
13 circo -Tpng maxMatchingcycleGraph.dot\ 306
14     -o maxMatchingcycleGraph.png
```

21.5 Grids and the ISING Model

307

Special classes of graphs, like the `GridGraph` class, provide $n \times m$ rectangular or triangular grids. With a *Gibbs* sampler², the `IsingModel` class can simulate in Fig. 21.4 on the next page an ISING model on, for instance, a 15×15 grid (Ising 1925).

Listing 21.11 Simulating an Ising model on a 15×15 rectangular grid

```
1 >>> from graphs import GridGraph, IsingModel 312
2 >>> g = GridGraph(n=15,m=15) 313
3 >>> g.showShort() 314
4 *----- show short -----* 315
5 Grid graph : grid-6-6 316
6 n : 6 317
7 m : 6 318
8 order : 36 319
9 >>> im = IsingModel(g,beta=0.3,nSim=100000) 320
10 Running a Gibbs Sampler for 100000 step ! 321
11 >>> im.exportGraphViz(colors=['lightblue','lightcoral']) 322
12 *----- exporting a dot file for GraphViz tools -----* 323
13 Exporting to grid-15-15-isinq.dot 324
14 fdp -Tpng qgrid-15-15-isinq.dot -o qgrid-15-15-isinq.png 325
```

21.6 Simulating METROPOLIS Random Walks

326

Finally, we provide the `MetropolisChain` class, a specialisation of the `Graph` class, implementing a generic Monte Carlo Markov Chain (MCMC) sampler for simulating random walks on a graph following given transition probabilities `probs`

² Geman and Geman (1984).

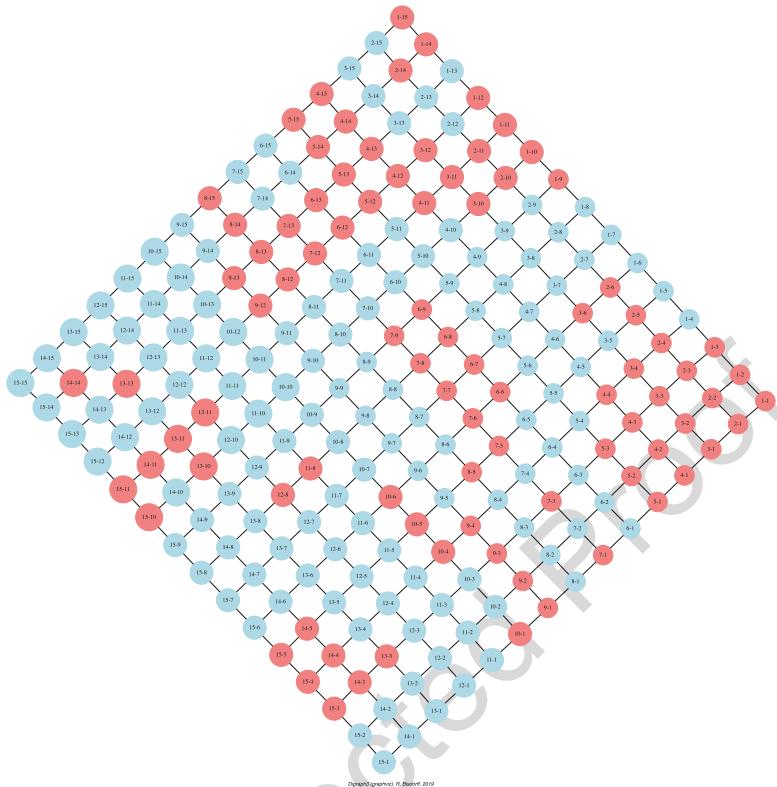


Fig. 21.4 Ising model of the 15×15 grid graph

`= {'v1': x, 'v2': y, ...}` for visiting each vertex (see Listing 21.12) (Metropolis et al. 1953). 330
331

Listing 21.12 Simulating random walks on a graph

```

1 >>> from graphs import Graph 332
2 >>> g = Graph(numberOfVertices=5, edgeProbability=0.5) 333
3 >>> g.showShort() 334
4     *---- short description of the graph ----* 335
5     Name          : 'randomGraph' 336
6     Vertices      : ['v1', 'v2', 'v3', 'v4', 'v5'] 337
7     Valuation domain : {'max': 1, 'med': 0, 'min': -1} 338
8     Gamma function : 339
9         v1 -> ['v2', 'v3', 'v4'] 340
10        v2 -> ['v1', 'v4'] 341
11        v3 -> ['v5', 'v1'] 342
12        v4 -> ['v2', 'v5', 'v1'] 343
13        v5 -> ['v3', 'v4'] 344
14 >>> # initialize a potential stationary probability vector 345
15 >>> n = g.order 346

```

```

16 >>> i = 0
17 >>> for v in g.vertices:
18 ...     probs[v] = (n - i) / (n * (n+1) / 2)
19 ...     i += 1

```

347
348
349
350

The `checkSampling()` method of the `MetropolisChain` class (see 351
below) generates a random walk of $nSim = 30,000$ steps on the given graph and 352
records by the way the observed relative frequency with which each vertex is passed 353
by. In Listing 21.13, we notice how well the random walk is matching the given 354
stationary probability vector. 355

Listing 21.13 Checking the quality of the MCMC sampler

```

1 >>> from graphs import MetropolisChain
2 >>> met = MetropolisChain(g, probs)
3 >>> frequency = met.checkSampling(verticesList[0], nSim=30000)
4 >>> for v in verticesList:
5 ...     print(v, probs[v], frequency[v])
6     v1 0.3333 0.3343
7     v2 0.2666 0.2680
8     v3 0.2 0.2030
9     v4 0.1333 0.1311
10    v5 0.0666 0.0635

```

356
357
358
359
360
361
362
363
364
365

In this example, the stationary transition probability distribution (see Listing 366
21.13 above), shown by the `showTransitionMatrix()` method in Listing 367
21.14, is quite adequately simulated. 368

Listing 21.14 Printing the transition probability distribution

```

1 >>> met.showTransitionMatrix()
2 * ----- Transition Matrix -----
3   Pij | 'v1'   'v2'   'v3'   'v4'   'v5'
4   -----|-----
5   'v1' | 0.23   0.33   0.30   0.13   0.00
6   'v2' | 0.42   0.42   0.00   0.17   0.00
7   'v3' | 0.50   0.00   0.33   0.00   0.17
8   'v4' | 0.33   0.33   0.00   0.08   0.25
9   'v5' | 0.00   0.00   0.50   0.50   0.00

```

369
370
371
372
373
374
375
376
377

For the reader interested in algorithmic applications of Markov Chains, we 378
recommend consulting (Häggström 2002). 379

21.7 Computing the Non-isomorphic MISs of the n -Cycle Graph

380
381

Due to the public success of our common work with Jean-Luc Marichal (Bisdorff 382
and Marichal 2008), we present in this chapter an example Python session for 383
computing the non-isomorphic maximal independent sets (MISs) of the 12-cycle 384

graph, i.e. a `CirculantDigraph` class instance of order 12 and symmetric circulants 1 and -1 . 385
386

```

1 >>> from digraphs import CirculantDigraph 387
2 >>> c12 = CirculantDigraph(order=12,circulants=[1,-1]) 388
3 >>> c12 # 12-cycle digraph instance 389
4 *----- Digraph instance description -----* 390
5 Instance class : CirculantDigraph 391
6 Instance name : c12 392
7 Digraph Order : 12 393
8 Digraph Size : 24 394
9 Valuation domain : [-1.0, 1.0] 395
10 Determinateness : 100.000 396
11 Attributes : ['name', 'order', 'circulants', 397
12 'actions', 'valuationdomain', 398
13 'relation', 'gamma', 'notGamma'] 399

```

Such n -cycle graphs—see the 12-cycle graph in Fig. 21.5 on page 317—are also 400
provided as undirected graph instances by the `CycleGraph` class. 401

```

1 >>> from graphs import CycleGraph 402
2 >>> cg12 = CycleGraph(order=12) 403
3 >>> cg12 404
4 *----- Graph instance description -----* 405
5 Instance class : CycleGraph 406
6 Instance name : cycleGraph 407
7 Graph Order : 12 408
8 Graph Size : 12 409
9 Valuation domain : [-1.0, 1.0] 410
10 Attributes : ['name', 'order', 'vertices', 411
11 'valuationDomain', 'edges', 412
12 'size', 'gamma'] 413

```

A non-isomorphic MIS of the 12-cycle graph corresponds in fact to a set of 414
isomorphic MISs, i.e. an orbit of MISs under the automorphism group of the 12- 415
cycle graph. In Listing 21.15, we are now first computing all maximal independent 416
sets that are detectable in the 12-cycle digraph with the `showMIS()` method. 417

Listing 21.15 Computing the MISs of the 12-cycle graph

```

1 >>> c12.showMIS(withListing=False) 418
2 *-- Maximal independent choices ---* 419
3 number of solutions: 29 420
4 cardinality distribution 421
5 card.: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] 422
6 freq.: [0, 0, 0, 0, 3, 24, 2, 0, 0, 0, 0, 0] 423
7 Results in c12.misset 424

```

In the 12-cycle graph, we observe 29 labelled MISs: 3 of cardinality 4, 24 of 425
cardinality 5, and 2 of cardinality 6. In case of n -cycle graphs with $n > 20$, 426
as the cardinality of the MISs becomes big, it is preferable to use, as shown in 427

Listing 21.16, the `perrinMIS` shell command³ compiled from C and installed 428 along with all the `DIGRAPH3` python modules for computing the set of MISs 429 observed in the graph. 430

Listing 21.16 Computing the MISs with the `perrinMIS` shell command

```

1 ...$ echo 12 | /usr/local/bin/perrinMIS          431
2 # ----- #                                         432
3 # Generating MIS set of Cn with the             # 433
4 # Perrin sequence algorithm.                   # 434
5 # Temporary files used.                      # 435
6 # even versus odd order optimised.          # 436
7 # RB December 2006                            # 437
8 # Current revision Dec 2018                 # 438
9 # ----- #                                         439
10 Input cycle order ? <-- 12                   440
11 mis 1 : 100100100100                         441
12 mis 2 : 010010010010                         442
13 mis 3 : 001001001001                         443
14 ...
15 ...
16 ...
17 mis 27 : 001001010101                         447
18 mis 28 : 101010101010                         448
19 mis 29 : 010101010101                         449
20 Cardinalities:                                450
21 0 : 0                                         451
22 1 : 0                                         452
23 2 : 0                                         453
24 3 : 0                                         454
25 4 : 3                                         455
26 5 : 24                                         456
27 6 : 2                                         457
28 7 : 0                                         458
29 8 : 0                                         459
30 9 : 0                                         460
31 10 : 0                                         461
32 11 : 0                                         462
33 12 : 0                                         463
34 Total: 29                                     464
35 execution time: 0 sec. and 2 millisec.        465

```

³The `perrinMIS` shell command may be installed system-wide with the command `make installPerrin` from the main `DIGRAPH3` directory. It is stored by default into `/usr/local/bin/`. This may be changed with the `INSTALLDIR` flag. The command `make installPerrinUser` installs it instead without sudo into the user's private `.bin` directory.

Reading in the result of the `perrinMIS` shell command, stored in a file called by 466
default `curd.dat`, may be operated with the `readPerrinMisset()` method. 467

```

1 >>> c12.readPerrinMisset(file='curd.dat') 468
2 >>> c12.misset 469
3     {frozenset({'5', '7', '10', '1', '3'}), 470
4     frozenset({'9', '11', '5', '2', '7'}), 471
5     frozenset({'7', '2', '4', '10', '12'}), 472
6     ...
7     ...
8     ...
9     frozenset({'8', '4', '10', '1', '6'}), 476
10    frozenset({'11', '4', '1', '9', '6'}), 477
11    frozenset({'8', '2', '4', '10', '12', '6'}) 478
12 }
```

For computing the corresponding non-isomorphic MISs, we actually need the 480
automorphism group of the 12-cycle graph. The `Digraph` class therefore provides 481
the `automorphismGenerators()` method which adds automorphism group 482
generators to a `Digraph` class instance with the help of the external `dreadnaut` 483
shell command from the *nauty* software package.⁴ 484

Listing 21.17 Computing the automorphism group generators

```

1 >>> c12.automorphismGenerators() 485
2 ...
3     Permutations 487
4     {'1':'1', '2':'12', '3':'11', '4':'10', 488
5     '5':'9', '6':'8', '7':'7', '8':'6', '9':'5', 489
6     '10':'4', '11':'3', '12':'2'} 490
7     {'1':'2', '2':'1', '3':'12', '4':'11', '5':'10', 491
8     '6':'9', '7':'8', '8':'7', '9':'6', '10':'5', 492
9     '11':'4', '12':'3'} 493
10 >>> print('grpsize = ', c12.automorphismGroupSize) 494
11 grpsize = 24 495
```

The 12-cycle graph's automorphism group, of size 24, is generated with both the 496
permutations shown in Listing 21.17. 497

The `showOrbits()` method renders now the labelled representatives of each 498
of the four orbits of isomorphic MISs observed in the 12-cycle graph (see Lines 7– 499
10 below). 500

⁴The `automorphismGenerators` method uses the `dreadnaut` shell command from the *nauty* software package. See [https://www3.cs.stonybrook.edu/~algorith/implement/nauty/](https://www3.cs.stonybrook.edu/~algorith/implement/nauty/implement.shtml) [implement.shtml](https://www3.cs.stonybrook.edu/~algorith/implement/nauty/implement.shtml). On Mac OS, there exist dmg installers, and on Ubuntu Linux or Debian, one may easily install it with `...$ sudo apt-get install nauty`.

Listing 21.18 Computing the MIS orbits of the 12-cycle graph

```

1 >>> c12.showOrbits(c12.misset, withListing=False)      501
2 ...
3 *---- Global result ----      502
4 Number of MIS: 29      503
5 Number of orbits : 4      504
6 Labelled representatives and cardinality:      505
7 1: ['2','4','6','8','10','12'], 2      506
8 2: ['2','5','8','11'], 3      507
9 3: ['2','4','6','9','11'], 12      508
10 4: ['1','4','7','9','11'], 12      509
11 Symmetry vector      510
12 stabilizer size: [1, 2, 3, ..., 8, 9, ..., 12, 13, ...]      511
13 frequency : [0, 2, 0, ..., 1, 0, ..., 1, 0, ...]      512

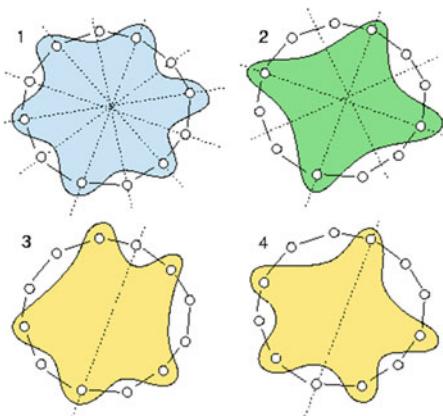
```

The corresponding group stabilisers' sizes and frequencies, orbit 1 with 6 symmetry axes, orbit 2 with 4 symmetry axes, and orbits 3 and 4 both with one symmetry axis (see Lines 12–13 in Listing 21.18), are illustrated in the corresponding unlabelled graphs of Fig. 21.5.

The non-isomorphic MISs in the 12-cycle graph represent in fact all the ways one may write the number 12 as the circular sum of '2's and '3's without distinguishing opposite directions of writing. The first orbit corresponds to writing six times a '2', and the second orbit corresponds to writing four times a '3'. The third and fourth orbits correspond to writing two times a '3' and three times a '2'. There are two non-isomorphic ways to do this latter circular sum, separating the '3's either by one and two '2's or by zero and three '2's (Bisdorff and Marichal 2008).

Chapter 22 is devoted more specifically to tree graphs and graph forests.

Fig. 21.5 Symmetry axes of the four non-isomorphic MISs of the 12-cycle graph



References

526

- Bisdorff R, Marichal J (2008) Counting non-isomorphic maximal independent sets of the n-cycle graph. *J Int Seq* 11(Art. 08.5.7):1–16. <https://cs.uwaterloo.ca/journals/JIS/VOL11/Marichal/marichal.html> 527
528
529
- Geman S, Geman D (1984) Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans Pattern Anal Mach Intell* 6:721–741 530
531
- Häggström O (2002) Finite Markov Chains and algorithmic applications. Cambridge University 532
Press, Cambridge 533
- Ising E (1925) Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift für Physik* 31:253–258 534
- Metropolis N, Rosenbluth A, Rosenbluth M, Teller A (1953) Equation of state calculations by fast 535
computing machines. *J Chem Phys* 21(6):1087 536

Uncorrected Proof

AUTHOR QUERIES

- AQ1.** Please confirm if the term “3-coloring” can be changed to “3-colouring” throughout the book.
- AQ2.** Please check if the sentence “There are two non-isomorphic ...” is fine as edited and amend if required.

Uncorrected Proof

Chapter 22

On Tree Graphs and Graph Forests

1

2

Contents

22.1 Generating Random Tree Graphs	319	4
22.2 Recognising Tree Graphs	322	5
22.3 Spanning Trees and Forests	324	6
22.4 Maximum Determined Spanning Forests	326	7

Abstract The chapter specifically addresses working with tree graphs and graph forests. We illustrate how to generate and recognise random tree graphs and how to compute the centres of a tree and draw a rooted and oriented tree. Finally, algorithms for computing spanning trees and forests are presented.

22.1 Generating Random Tree Graphs

12

Using the `RandomTree` class from the `graphs` module, we may, for instance, generate a random tree graph with 9 vertices.

13

14

Listing 22.1 Generating a random tree graph

```
1 >>> from graphs import RandomTree
2 >>> rt = RandomTree(order=9, seed=100)
3 >>> rt
4     ----- Graph instance description -----
5     Instance class    : RandomTree
6     Instance name     : randomTree
7     Graph Order       : 9
8     Graph Size        : 8
9     Valuation domain : [-1.00; 1.00]
10    Attributes       : ['name', 'order',
11                          'vertices', 'valuationDomain',
12                          'edges', 'prueferCode',
13                          'size', 'gamma']
14    ----- RandomTree specific data -----
15    Pruefer code     : ['v3', 'v8', 'v8', 'v3', 'v7', 'v6', 'v7']
```

```

16 >>> rt.exportGraphViz('tutRandomTree')          30
17     ----- exporting a dot file for GraphViz tools --* 31
18     Exporting to tutRandomTree.dot                32
19     neato -Tpng tutRandomTree.dot -o tutRandomTree.png 33

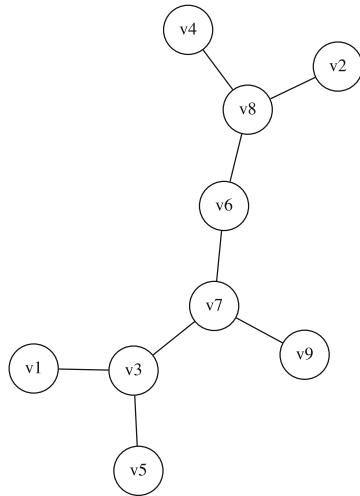
```

In Fig. 22.1, one may notice that a tree graph of order $n > 2$ always contains $n - 1$ edges (see Lines 7 and 8 in Listing 22.1 on the previous page) and its structure is entirely characterised by a corresponding PRÜFER code, i.e. a list of vertices keys of length $n - 2$. See, for instance, in Line 15 the code `['v3', 'v8', 'v8', 'v3', 'v7', 'v6', 'v7']` corresponding to our sample tree graph `rt`. 34
35
36
37
38
39
40
41
42
43
44
45
46

Each position of the code indicates the parent of the remaining leaf with the smallest vertex label. Vertex `v3` is thus the parent of `v1` and we drop leaf `v1`, `v8` is now the parent of leaf `v2` and we drop `v2`, vertex `v8` is again the parent of leaf `v4` and we drop `v4`, vertex `v3` is the parent of leaf `v5` and we drop `v5`, `v7` is now the parent of leaf `v3` and we may drop `v3`, `v6` becomes the parent of leaf `v8` and we drop `v8`, and `v7` becomes the parent of leaf `v6` and we may drop `v6`. The two eventually remaining vertices, `v7` and `v9`, give the last link in the reconstructed tree (Barthélémy and Guenoche 1991). 47
48
49
50

It is as well possible to, first, generate a random PRÜFER code of length $n - 2$ from a set of n vertices and then construct the corresponding tree of order n by reversing the procedure illustrated above (see Listing 22.2 on the facing page). The resulting tree graph is shown in Fig. 22.2 on the next page. 47
48
49
50

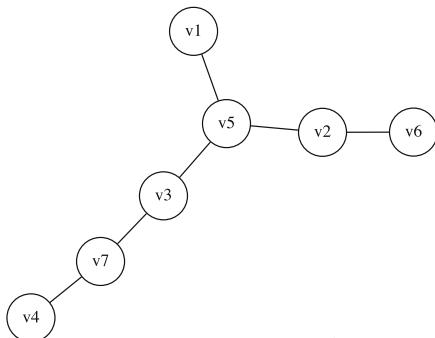
Fig. 22.1 Random tree graph instance of order 9. One may distinguish vertices like `v1`, `v2`, `v4`, `v5`, or `v9` of degree 1, called the *leaves* of the tree, and vertices like `v3`, `v6`, `v7`, or `v8` of degree 2 or more, called the *nodes* of the tree



Diagraph3 (graphviz), R. Bisдорff, 2019

Fig. 22.2 Tree graph instance generated with the random PRÜFER code

```
['v5', 'v7', 'v2',
 'v5', 'v3']
```



DiGraph3 (graphviz), R. Bisdorff, 2019

Listing 22.2 Generating a tree graph with a random PRÜFER code

```

1  >>> verticesList = ['v1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7']      51
2  >>> n = len(verticesList)                                         52
3  >>> from random import seed, choice                                53
4  >>> seed(101)                                                 54
5  >>> code = []                                                 55
6  >>> for k in range(n-2):                                         56
7  ...     code.append( choice(verticesList) )                         57
8  >>> print(code)                                              58
9  ['v5', 'v7', 'v2', 'v5', 'v3']                                     59
10 >>> rt = RandomTree(prueferCode=code)                            60
11 >>> rt
12     *----- Graph instance description -----*                   61
13     Instance class    : RandomTree                            62
14     Instance name     : randomTree                           63
15     Graph Order       : 7                                    64
16     Graph Size        : 6                                    65
17     Valuation domain : [-1.00; 1.00]                         66
18     Attributes : ['name', 'order', 'vertices',             67
19                  'valuationDomain', 'edges',                 68
20                  'prueferCode', 'size', 'gamma']            69
21     *----- RandomTree specific data -----*                  70
22     Pruefer code : ['v5', 'v7', 'v2', 'v5', 'v3']          71
23 >>> rt.exportGraphViz('tutPruefTree')                      72
24     *----- exporting a dot file for GraphViz tools -----* 73
25     Exporting to tutPruefTree.dot                           74
26     neato -Tpng tutPruefTree.dot -o tutPruefTree.png        75
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
```

Following from the bijection between a labelled tree and its PRÜFER code, we actually know that there exist n^{n-2} different tree graphs with the same n vertices.

Given a genuine graph, how can we recognise that it is in fact a tree instance?

22.2 Recognising Tree Graphs

80

Given a graph g of order n and size s , the following 5 assertions A1, A2, A3, A4, 81
and A5 are all equivalent (see Barthélemy and Guenoche 1991): 82

- A1: g is a tree. 83
- A2: g is without (chordless) cycles and $n = s + 1$. 84
- A3: g is connected and $n = s + 1$. 85
- A4: Any two vertices of g are always connected by a unique path. 86
- A5: g is connected and dropping any single edge will always disconnect g . 87

Assertion A3, for instance, gives a simple test for recognising a tree graph. In 88
case of a *lazy evaluation* of the test in Listing 22.3, Line 3 below, it is opportune, 89
from a computational complexity perspective, to first check the order and size of 90
the graph, before checking its potential connectedness. We provide the `isTree()` 91
method for computing both these tests (see Line 8). 92

Listing 22.3 Recognising a tree graph

```

1 >>> from graphs import RandomGraph
2 >>> g = RandomGraph(order=8, edgeProbability=0.3, seed=62)
3 >>> if g.order == (g.size +1) and g.isConnected():
4 ...     print('The graph is a tree ?', True)
5 ... else:
6 ...     print('The graph is a tree ?', False)
7 ...     The graph is a tree ? True
8 >>> g.isTree()
9     True

```

The random graph of order 8 and edge probability 30%, generated with seed 62, 102
is actually a tree graph instance, as confirmed by its graphviz drawing shown in 103
Fig. 22.3 on the next page. 104

```

1 >>> g.exportGraphViz('test62')
2     *---- exporting a dot file for GraphViz tools ---*
3     Exporting to test62.dot
4     fdp -Tpng test62.dot -o test62.png

```

Yet, we still have to recover its corresponding PRÜFER code. Therefore, we can 109
use the `tree2Pruefer()` method from the `TreeGraph` class. But, first, the 110
instance class of graph g is changed to the `TreeGraph` type (see Line 2 below). 111

Listing 22.4 Computing the PRÜFER code of a tree graph instance

```

1 >>> from graphs import TreeGraph
2 >>> g.__class__ = TreeGraph
3 >>> g.tree2Pruefer()
4     ['v6', 'v1', 'v2', 'v1', 'v2', 'v5']

```

Fig. 22.3 Recognising a tree graph. One may notice that vertex v_2 is actually situated in the *centre* of the tree with a neighbourhood depth of 2

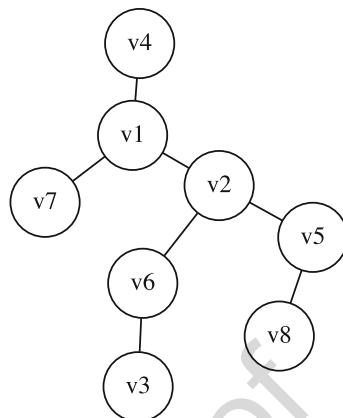


Diagram3 (graphviz), R. Bisdorff, 2019

Fig. 22.4 Drawing an oriented tree rooted at its centre

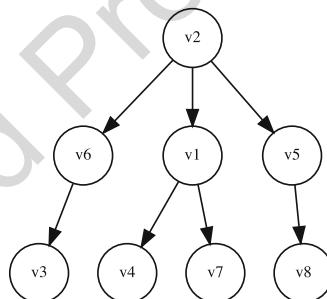


Diagram3 (graphviz)
R. Bisdorff, 2020

In Fig. 22.3, we noticed that vertex v_2 is actually situated in the *centre* of the tree with a neighbourhood depth of 2. Centres of a graph are the vertices with minimal neighbourhood depth. For finding such centre(s), the Graph class provides the `computeGraphCentres()` method (see Line 1 in Listing 22.5). Knowing now the centre of graph g , we may draw a correspondingly rooted and oriented tree with the `exportOrientedTreeGraphViz()` method from the `TreeGraph` class (see Fig. 22.4).

Listing 22.5 Computing the centres of a tree and drawing a rooted and oriented tree

```

1  >>> g.computeGraphCentres()
2  {'v2': 2}
3  >>> g.exportOrientedTreeGraphViz(\ 
4  ...     fileName='rootedTree', root='v2')
5  ----- exporting a dot file for GraphViz tools
6  Exporting to rootedTree.dot
7  dot -Grankdir=TB -Tpng rootedTree.dot -o rootedTree.png

```

Let us now turn our attention towards a major application of tree graphs, namely *spanning trees* and *forests* related to graph traversals.

22.3 Spanning Trees and Forests

132

With the `RandomSpanningTree` class, we may generate, from a given connected graph `g` instance, *uniform* random instances of a spanning tree by using WILSON's algorithm (see Listing 22.6 and Fig. 22.5). 133
134
135

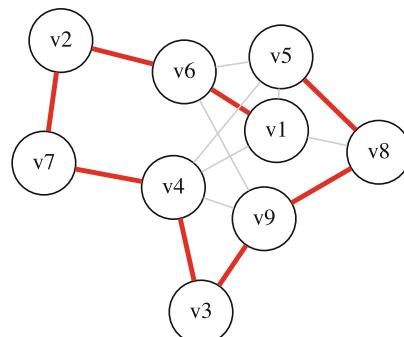
Listing 22.6 Generating uniform random spanning trees

```

1  >>> from graphs import RandomGraph, \
2      ...           RandomSpanningTree
3  >>> g = RandomGraph(order=9, \
4      ...           edgeProbability=0.4, seed=100)
5  >>> spt = RandomSpanningTree(g)
6  >>> spt
7  *----- Graph instance description -----*
8  Instance class   : RandomSpanningTree
9  Instance name    : randomGraph_randomSpanningTree
10 Graph Order      : 9
11 Graph Size       : 8
12 Valuation domain : [-1.00; 1.00]
13 Attributes       : ['name', 'vertices', 'order',
14                      'valuationDomain',
15                      'edges', 'size', 'gamma',
16                      'dfs', 'date', 'dfsx',
17                      'prueferCode']
18 *----- RandomTree specific data -----*
19 Pruefer code : ['v7', 'v9', 'v5', 'v1', 'v8', 'v4', 'v9']
20 >>> spt.exportGraphViz(fileName='randomSpanningTree', \
21 ...                         WithSpanningTree=True)
22 *----- exporting a dot file for GraphViz tools -----*
23 Exporting to randomSpanningTree.dot
24 [[v1', 'v5', 'v6', 'v5', 'v1', 'v8', 'v9', 'v3', 'v9', 'v4',
25   'v7', 'v2', 'v7', 'v4', 'v9', 'v8', 'v1']]
26 neato -Tpng randomSpanningTree.dot \
27   -o randomSpanningTree.png

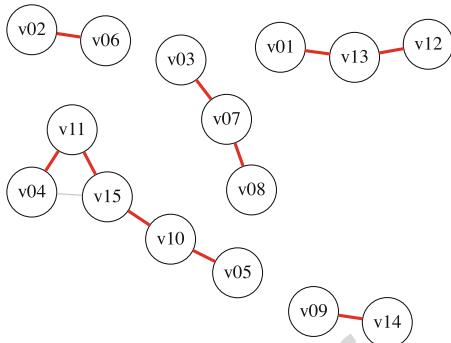
```

Fig. 22.5 Random spanning tree



Digraph3 (graphviz), R. Bischoff, 2019

Fig. 22.6 Random spanning forest



Digraph3 (graphviz), R. Bisdorff, 2019

WILSON's algorithm only works for connected graphs (Wilson 1996). More 163 general, and in case of a not connected graph, the RandomSpanningForest 164 class generates a not necessarily uniform random instance of a *spanning forest*— 165 one or more random tree graphs—generated from a random *depth first search* of the 166 graph components' traversals (see Listing 22.7 and Fig. 22.6). 167

Listing 22.7 Computing spanning forests over disconnected graphs

```

1 >>> g = RandomGraph(order=15, \
2 ...                           edgeProbability=0.1, seed=140)           168
3 >>> g.computeComponents()           169
4   [ {'v12','v01','v13'}, {'v02','v06'},           170
5   {'v08','v03','v07'}, {'v15','v11','v10','v04','v05'}, 171
6   {'v09','v14'}]           172
7 >>> spf = RandomSpanningForest(g, seed=100)           173
8 >>> spf.exportGraphViz(fileName='spanningForest', \
9 ...                           WithSpanningTree=True)           174
10 *---- exporting a dot file for GraphViz tools ----* 175
11   Exporting to spanningForest.dot           176
12   [ ['v03','v07','v08','v07','v03'],           177
13   ['v13','v12','v13','v01','v13'],           178
14   ['v02','v06','v02'],           179
15   ['v15','v11','v04','v11','v15',           180
16   'v10','v05','v10','v15'],           181
17   ['v09','v14','v09']]           182
18   neato -Tpng spanningForest.dot -o spanningForest.png 183

```

22.4 Maximum Determined Spanning Forests

186

In case of valued graphs—supporting weighted edges—we may finally construct a *most determined* spanning tree (or forest if not connected) using KRUSKAL’s greedy minimum spanning tree algorithm on the dual valuation of the graph (Kruskal 1956).¹

In Listing 22.8, we generate, for instance, a randomly valued graph with five vertices and seven edges bipolar-valued in $[-1.0; 1.0]$.

Listing 22.8 Generating randomly bipolar-valued graphs

```

1 >>> from graphs import RandomValuationGraph 193
2 >>> g = RandomValuationGraph(order=5, seed=2) 194
3 >>> g 195
4     *----- Graph instance description -----* 196
5     Instance class      : RandomValuationGraph 197
6     Instance name       : randomGraph 198
7     Graph Order         : 5 199
8     Graph Size          : 7 200
9     Valuation domain   : [-1.00; 1.00] 201
10    Attributes          : ['name', 'order', 202
11          'vertices', 'valuationDomain', 203
12          'edges', 'size', 'gamma'] 204

```

To inspect the edges’ actual weights, we transform in Listing 22.9 the random graph g into a corresponding digraph dg and use the `showRelationTable()` method for printing its symmetric adjacency matrix.

Listing 22.9 Symmetric relation table

```

1 >>> dg = g.graph2Digraph() 208
2 >>> dg.showRelationTable() 209
3     *----- Relation Table -----* 210
4     S | 'v1' 'v2' 'v3' 'v4' 'v5' 211
5     - - - - - 212
6     'v1' | 0.00 0.91 0.90 -0.89 -0.83 213
7     'v2' | 0.91 0.00 0.67 0.47 0.34 214
8     'v3' | 0.90 0.67 0.00 -0.38 0.21 215
9     'v4' | -0.89 0.47 -0.38 0.00 0.21 216
10    'v5' | -0.83 0.34 0.21 0.21 0.00 217
11    Valuation domain: [-1.00;1.00] 218

```

To compute the most determined spanning tree or forest, we can use the `Best-DeterminedSpanningForest` class constructor.

¹ KRUSKAL’s algorithm is a minimum spanning tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest.

Listing 22.10 Computing best determined spanning forests

```

1 >>> from graphs import \
2             BestDeterminedSpanningForest
3 >>> mt = BestDeterminedSpanningForest(g)
4 >>> print(mt)
5     ----- Graph instance description -----
6     Instance class    : BestDeterminedSpanningForest
7     Instance name     : bdSpanningForest
8     Graph Order       : 5
9     Graph Size        : 4
10    Valuation domain : [-1.00; 1.00]
11    Attributes        : ['name', 'vertices', 'order',
12                          'valuationDomain',
13                          'edges', 'size', 'gamma',
14                          'dfs', 'date',
15                          'averageTreeDetermination']
16    ----- best determined spanning tree -----
17    Depth first search path(s) :
18    [['v1', 'v2', 'v4', 'v2', 'v5', 'v2', 'v1', 'v3', 'v1']]
19    Average determination(s) : [Decimal('0.655')]

```

The random graph g is connected and, hence, admits a single spanning tree of maximum mean determination = $(0.47 + 0.91 + 0.90 + 0.34)/4 = 0.655$ (see Lines 9, 6, and 10 in Listing 22.9 on the preceding page) (Fig. 22.7).

AQ1

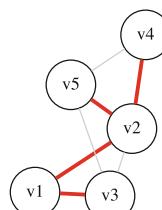
```

1 >>> mt.exportGraphViz(\ \
2 ...     fileName='bestDeterminedspanningTree', \
3 ...     WithSpanningTree=True)
4     ----- exporting a dot file for GraphViz tools -----
5     Exporting to spanningTree.dot
6     [['v4', 'v2', 'v1', 'v3', 'v1', 'v2', 'v5', 'v2', 'v4']]
7     neato -Tpng bestDeterminedSpanningTree.dot \
8             -o bestDeterminedSpanningTree.png

```

One may easily verify that all other potential spanning trees, including instead the edges $\{v3, v5\}$ and/or $\{v4, v5\}$, will show a lower average determination.

Chapter 23, the last on undirected graphs, is devoted to different models of BERGE or perfect graphs, namely split, interval, comparability, and permutation graphs.

Fig. 22.7 Best determined spanning tree

Digraph3 (graphviz), R. Bisдорff, 2019

References

256

- Barthélémy J, Guenoche A (1991) *Trees and Proximities Representations*. Wiley, Hoboken 257
- Kruskal J (1956) On the shortest spanning subtree of a graph and the traveling salesman problem. 258
Proc Am Math Soc 7:48–50 259
- Wilson DB (1996) Generating random spanning trees more quickly than the cover time. In: 260
Proceedings of the twenty-eighth annual ACM symposium on the theory of computing 261
(Philadelphia PA), ACM New York, pp 296–303 262

Uncorrected Proof

AUTHOR QUERY

- AQ1.** Missing citation for Fig. “22.7” was inserted here. Please check if appropriate. Otherwise, please provide citation for Fig. “22.7”. Note that the order of main citations of figures in the text must be sequential.

Uncorrected Proof

Chapter 23

About Split, Comparability, Interval, and Permutation Graphs

1
2
3

Contents

23.1 A ‘multiply’ Perfect Graph	329	5
23.2 Who Is the Liar?	331	6
23.3 Generating Permutation Graphs	333	7
23.4 Recognising Permutation Graphs	336	8

Abstract The last chapter of this book eventually presents some famous classes of perfect graphs, namely comparability, interval, permutation, and split graphs. We first present an example of a graph which is at the same time a triangulated, a comparability, a split, and a permutation graph. The importance to be an interval is illustrated with BERGE’s mystery story. We discuss furthermore the generation of permutation graphs and close with how to recognise that a given graph is in fact a permutation graph.

23.1 A ‘multiply’ Perfect Graph

16

A graph g is called:

- BERGE or *perfect* when g and its dual $-g$ both do not contain any chordless odd cycles of length greater than 3 (Berge 1963; Chudnovsky et al. 2006).
- *Triangulated* graph when g does not contain any *chordless cycle* of length 4 and more.

Definition 23.1 (Some Perfect Graph Classes) Following Martin Golumbic (see Golumbic 2004, p. 149), we call a given graph g :

- *Comparability* graph when g is *transitively orientable*.
- *Interval* graph when g is *triangulated* and its dual— g is a *comparability* graph.
- *Permutation* graph when g and its dual $-g$ both are *comparability* graphs.
- *Split* graph when g and its dual $-g$ both are *triangulated* graphs.

To illustrate these *perfect* graph classes, we will generate from 8 intervals, randomly chosen in the default integer range $[0, 10]$, a `RandomIntervalIntersectionsGraph` instance `g` (see Line 2 below). 28
29
30

```

1  >>> from graphs import\                                31
2  ...          RandomIntervalIntersectionsGraph\        32
3  >>> g = RandomIntervalIntersectionsGraph(\        33
4  ...          order=8, seed=100)\                     34
5  >>> g\                                         35
6  *----- Graph instance description -----*          36
7  Instance class : RandomIntervalIntersections\    37
8  Instance name  : randIntervalIntersections\      38
9  Seed          : 100\                            39
10 Graph Order   : 8\                            40
11 Graph Size    : 23\                            41
12 Valuation domain : [-1.0; 1.0]\            42
13 Attributes    : ['seed', 'name', 'order',\    43
14     'intervals', 'vertices', 'valuationDomain',\ 44
15     'edges', 'size', 'gamma']\                  45
16 >>> print(g.intervals)\                     46
17 [(2,7), (2,7), (5,6), (6,8), (1,8), (1,1), (4,7), (0,10)] 47

```

With `seed = 100`, we obtain here an *interval graph*, in fact a *perfect graph*, which is *conjointly* a *triangulated*, a *comparability*, a *split*, and a *permutation graph* (see Listing 23.1, Lines 2, 6, 10, and 14 and Fig. 23.1 on the facing page). 48
49
50

Listing 23.1 Testing perfect graph categories

```

1 >>> g.isPerfectGraph(Comments=True)\          51
2 Graph 'randIntervalIntersections' is perfect ! 52
3 >>> g.isIntervalGraph(Comments=True)\        53
4 Graph 'randIntervalIntersections' is triangulated. 54
5 Graph 'dual_randIntervalIntersections' is transitively orientable. 55
6 => Graph 'randIntervalIntersections' is an interval graph. 56
7 >>> g.isSplitGraph(Comments=True)\          57
8 Graph 'randIntervalIntersections' is triangulated. 58
9 Graph 'dual_randIntervalIntersections' is triangulated. 59
10 => Graph 'randIntervalIntersections' is a split graph. 60
11 >>> g.isPermutationGraph(Comments=True)\    61
12 Graph 'randIntervalIntersections' is transitively orientable. 62
13 Graph 'dual_randIntervalIntersections' is transitively orientable. 63
14 => Graph 'randIntervalIntersections' is a permutation graph. 64
15 >>> g.exportGraphViz('randomSplitGraph')\    65
16 *--- exporting a dot file for GraphViz tools -----* 66
17 Exporting to randomSplitGraph.dot\                  67
18 fdp -Tpng randomSplitGraph.dot -o randomSplitGraph.png 68

```

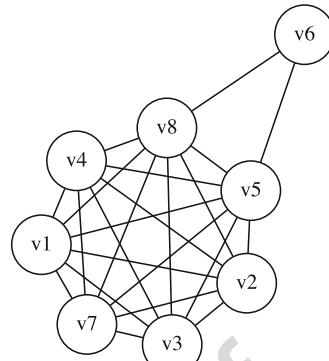
Notice however that the four properties:

1. g is a *comparability* graph. 70
2. g is a *co-comparability* graph, i.e. $-g$ is a comparability graph. 71
3. g is a *triangulated* graph. 72
4. g is a *co-triangulated* graph, i.e. $-g$ is a comparability graph. 73

are all independent one of the other (Golumbic 2004, p. 275). 74

Fig. 23.1 A conjointly triangulated, comparability, interval, permutation and split graph

In the figure here, one may readily recognise the essential characteristic of split graphs, namely being always splittable into two disjoint sub-graphs: an *independent choice* $\{v6\}$ and a *clique*— $\{v1, v2, v3, v4, v5, v7, v8\}$ —which explains their name



Digraph3 (graphviz), R. Bisdorff, 2019

23.2 Who Is the Liar?

75

Claude Berge's famous mystery story related by Golumbic (2004, p. 20) may well illustrate the importance of being an *interval* graph.

76
77

Suppose the file `berge.py`¹ contains the following Graph instance data:

78

```

1  vertices = {
2      'A': {'name': 'Abe', 'shortName': 'A'},
3      'B': {'name': 'Burt', 'shortName': 'B'},
4      'C': {'name': 'Charlotte', 'shortName': 'C'},
5      'D': {'name': 'Desmond', 'shortName': 'D'},
6      'E': {'name': 'Eddie', 'shortName': 'E'},
7      'I': {'name': 'Ida', 'shortName': 'I'},
8  }
9  valuationDomain = {'min':-1, 'med':0, 'max':1}
10 edges = {
11     frozenset([('A', 'B')) : 1,
12     frozenset([('A', 'C')) : -1,
13     frozenset([('A', 'D')) : 1,
14     frozenset([('A', 'E')) : 1,
15     frozenset([('A', 'I')) : -1,
16     frozenset([('B', 'C')) : -1,
17     frozenset([('B', 'D')) : -1,
18     frozenset([('B', 'E')) : 1,
19     frozenset([('B', 'I')) : 1,
20     frozenset([('C', 'D')) : 1,
21     frozenset([('C', 'E')) : 1,
22     frozenset([('C', 'I')) : 1,
23     frozenset([('D', 'E')) : -1,
24     frozenset([('D', 'I')) : 1,
25     frozenset([('E', 'I')) : 1,
26 }
```

79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103

¹ The file `berge.py` is provided in the `examples` directory of the DIGRAPH3 resources.

Fig. 23.2 Graph representation of the testimonies of the professors

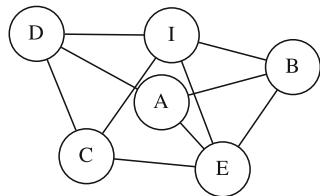


Diagram3 (graphviz), R. Bisdorff, 2019

Six professors (labelled A, B, C, D, E, and I) had been to the library on the 106 day that a precious document was stolen. Each entered once, stayed for some time, 106 and then left. If two professors were in the library at the same time, then at least 107 one of them saw the other. Detectives questioned the professors and gathered the 108 testimonies that *Abe* saw *Burt* and *Eddie*; *Burt* saw *Abe* and *Ida*; *Charlotte* saw 109 *Desmond* and *Ida*; *Desmond* saw *Abe* and *Ida*; *Eddie* saw *Burt* and *Ida*; and *Ida* saw 110 *Charlotte* and *Eddie*. This data is gathered in the previous file, where each positive 111 edge $\{x, y\}$ models the testimony that, either x saw y or y saw x . 112

```

1  >>> from graphs import Graph           113
2  >>> g = Graph('berge')                 114
3  >>> g.showShort()                     115
4  *---- short description of the graph ----* 116
5  Name           : 'berge'                117
6  Vertices       : ['A', 'B', 'C', 'D', 'E', 'I'] 118
7  Valuation domain : {'min': -1, 'med': 0, 'max': 1} 119
8  Gamma function :                      120
9   A -> ['D', 'B', 'E']                121
10  B -> ['E', 'I', 'A']                122
11  C -> ['E', 'D', 'I']                123
12  D -> ['C', 'I', 'A']                124
13  E -> ['C', 'B', 'I', 'A']            125
14  I -> ['C', 'E', 'B', 'D']            126
15 >>> g.exportGraphViz('berge1')        127
16 *---- exporting a dot file for GraphViz tools -----* 128
17 Exporting to berge1.dot               129
18 fdp -Tpng berge1.dot -o berge1.png 130

```

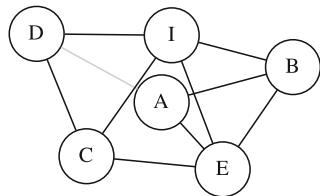
From graph theory, we know that time interval intersections graphs must in 131 fact be interval graph instances, i.e. *triangulated* and *co-comparative* graphs. The 132 testimonies graph shown in Fig. 23.2 should therefore not contain any chordless 133 cycle of four and more vertices. Now, the presence or not of such chordless cycles in 134 the testimonies graph can be checked with the `computeChordlessCycles()` 135 method (Bisdorff 2010). 136

```

1  >>> g.computeChordlessCycles()          137
2  Chordless cycle certificate: ['D', 'C', 'E', 'A', 'D'] 138
3  Chordless cycle certificate: ['D', 'I', 'E', 'A', 'D'] 139
4  Chordless cycle certificate: ['D', 'I', 'B', 'A', 'D'] 140
5  [(['D', 'C', 'E', 'A', 'D'], frozenset({'C', 'D', 'E', 'A'})), 141
6  ([('D', 'I', 'E', 'A', 'D'), frozenset({'D', 'E', 'I', 'A'}))], 142
7  ([('D', 'I', 'B', 'A', 'D'), frozenset({'D', 'B', 'I', 'A'})))] 143

```

Fig. 23.3 The triangulated testimonies graph



Digraph3 (graphviz), R. Bisdorff, 2019

We see in the listing above three intersection cycles of length 4, which is impossible to occur on the linear timeline. Obviously, one professor lied!

And it is *Desmond*. If we doubt his testimony that he saw *Abe* (see Line 1 below), we obtain indeed in Fig. 23.3 a triangulated graph instance whose dual is a comparability graph.

```

1 >>> g.setEdgeValue( ('D', 'A'), 0) 149
2 >>> g.showShort() 150
3     *---- short description of the graph ----* 151
4     Name          : 'berge' 152
5     Vertices      : ['A', 'B', 'C', 'D', 'E', 'I'] 153
6     Valuation domain : {'med': 0, 'min': -1, 'max': 1} 154
7     Gamma function : 155
8         A -> ['B', 'E'] 156
9         B -> ['A', 'I', 'E'] 157
10        C -> ['I', 'E', 'D'] 158
11        D -> ['I', 'C'] 159
12        E -> ['A', 'I', 'B', 'C'] 160
13        I -> ['B', 'E', 'D', 'C'] 161
14 >>> g.isIntervalGraph(Comments=True) 162
15 Graph 'berge' is triangulated. 163
16 Graph 'dual_berge' is transitively orientable. 164
17 => Graph 'berge' is an interval graph. 165
18 >>> g.exportGraphViz('berge2') 166
19     *---- exporting a dot file for GraphViz tools -----* 167
20     Exporting to berge2.dot 168
21     fdp -Tpng berge2.dot -o berge2.png 169
  
```

23.3 Generating Permutation Graphs

170

A graph is called a *permutation* or *inversion* graph if there exists a permutation of its list of vertices, like `[4, 3, 6, 1, 5, 2]`, such that the graph is isomorphic to the inversions operated by the permutation in this list (see Golumbic 2004, Chap. 7, pp 171
172
173 157-170). This kind of graphs is also part of the class of BERGE graphs (Fig. 23.4). 174

```

1 >>> from graphs import PermutationGraph 175
2 >>> g = PermutationGraph(permulation = [4, 3, 6, 1, 5, 2]) 176
3 >>> g 177
  
```

Fig. 23.4 The [4,3,6,1,5,2] permutation graph

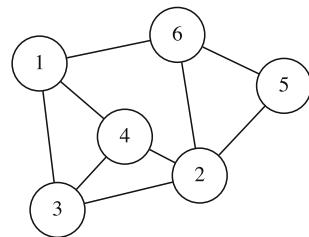


Diagram3 (graphviz), R. Bisdorff, 2019

```

4 *----- Graph instance description -----* 178
5 Instance class      : PermutationGraph 179
6 Instance name       : permutationGraph 180
7 Graph Order         : 6 181
8 Permutation         : [4, 3, 6, 1, 5, 2] 182
9 Graph Size          : 9 183
10 Valuation domain   : [-1.00; 1.00] 184
11 Attributes          : ['name', 'vertices', 'order', 185
12                 'permutation', 'valuationDomain', 186
13                 'edges', 'size', 'gamma'] 187
14 >>> g.isPerfectGraph() 188
15 True 189
16 >>> g.exportGraphViz(fileName='permutationGraph') 190
17 *---- exporting a dot file for GraphViz tools -----* 191
18 Exporting to permutationGraph.dot 192
19 fdp -Tpng permutationGraph.dot\ 193
20           -o permutationGraph.png 194

```

By using colour sorting queues, the minimal vertex colouring shown in Fig. 23.5 on the next page is computable in $O(n \log(n))$ (Golumbic 2004).

```

1 >>> g.computeMinimalVertexColoring(Comments=True) 197
2     vertex 1: lightcoral 198
3     vertex 2: lightcoral 199
4     vertex 3: lightblue 200
5     vertex 4: gold 201
6     vertex 5: lightblue 202
7     vertex 6: gold 203
8 >>> g.exportGraphViz(fileName='coloredPermutationGraph', \ 204
9     ...           WithVertexColoring=True) 205
10 *---- exporting a dot file for GraphViz tools -----* 206
11 Exporting to coloredPermutationGraph.dot 207
12 fdp -Tpng coloredPermutationGraph.dot\ 208
13           -o coloredPermutationGraph.png 209

```

The correspondingly coloured *matching diagram* of the nine inversions—the actual edges of the permutation graph—, which are induced by the given [4,3,6,1,5,2] permutation, may as well be drawn with the graphviz neato layout and explicitly positioned horizontal lists of vertices as shown in Fig. 23.6 on the facing page.

Fig. 23.5 Minimal vertex colouring of the permutation graph

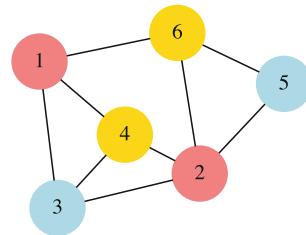


Diagram3 (graphviz), R. Bisdorff, 2019

Fig. 23.6 Coloured matching diagram of the permutation [4,3,6,1,5,2]

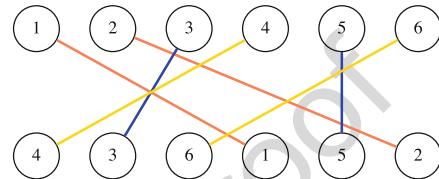


Diagram3 (graphviz), R. Bisdorff, 2019

```

1 >>> g.exportPermutationGraphViz('matchingDiagram', \
2 ...                               WithEdgeColoring=True)
3 *----- exporting a dot file for GraphViz tools -----
4 Exporting to matchingDiagram.dot
5 neato -n -Tpng matchingDiagram.dot\
6   -o matchingDiagram.png

```

215
216
217
218
219
220

As mentioned before, a permutation graph and its dual are *transitively orientable*.
The `transitiveOrientation()` method constructs from a given permutation graph a digraph where each edge of the permutation graph is converted into an arc oriented in an increasing alphabetic order of the adjacent vertices' keys (see Golumbic 2004). This orientation of the edges of a permutation graph is always transitive and delivers a *transitive ordering* of the vertices as shown in Fig. 23.7 on the next page.

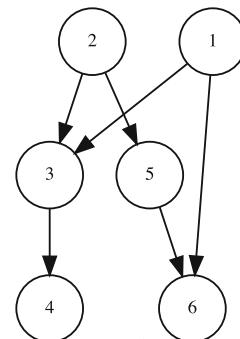
```

1 >>> dg = g.transitiveOrientation()
2
3 *----- Digraph instance description -----
4 Instance class   : TransitiveDigraph
5 Instance name    : oriented_permutationGraph
6 Digraph Order    : 6
7 Digraph Size     : 9
8 Valuation domain : [-1.00; 1.00]
9 Determinateness  : 100.000
10 Attributes      : ['name', 'order', 'actions',
11                           'valuationdomain', 'relation',
12                           'gamma', 'notGamma', 'size']
13 >>> print('Transitivity degree: %.3f' %\
14 ...           dg.computeTransitivityDegree())
15 Transitivity degree: 1.000
16 >>> dg.exportGraphViz(fileName='orientedPermGraph')

```

228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243

Fig. 23.7 The transitive orientation of the permutation graph



DiGraph3 (graphviz)
R. Bisdorff, 2020

```

17  *---- exporting a dot file for GraphViz tools ----*
18  Exporting to orientedPermGraph.dot
19  dot -Grankdir=TB -Tpng orientedPermGraph.dot \
20      -o orientedPermGraph.png

```

The dual of a permutation graph is again a permutation graph and as such also
transitively orientable.

```

1  >>> dgd = (-g).transitiveOrientation()
2  >>> print('Dual transitivity degree: %.3f' %\
3  ...           dgd.computeTransitivityDegree() )
4  Dual transitivity degree: 1.00

```

23.4 Recognising Permutation Graphs

Now, a given graph g is a permutation graph if and only if both g and $-g$ are
transitively orientable. This property gives a polynomial test procedure (in $O(n^3)$)
due to the transitivity check) for recognising permutation graphs.

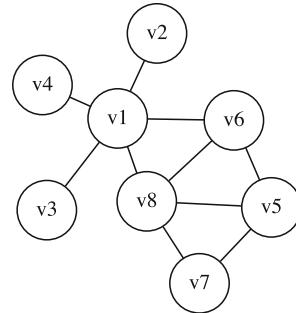
Let us consider, for instance, the following random graph of order 8 generated
with an edge probability of 40% and a random seed equal to 4335.

```

1  >>> from graphs import RandomGraph
2  >>> g = RandomGraph(order=8, \
3  ...           edgeProbability=0.4, seed=4335)
4  >>> g
5  *----- Graph instance description -----
6  Instance class   : RandomGraph
7  Instance name    : randomGraph
8  Seed             : 4335
9  Edge probability : 0.4
10 Graph Order     : 8
11 Graph Size      : 10
12 Valuation domain : [-1.00; 1.00]

```

Fig. 23.8 Random graph of order 8 generated with edge probability 0.4



Digraph3 (graphviz), R. Bisdorff, 2019

```

13     Attributes      : ['name', 'order', 'vertices',
14                           'valuationDomain', 'seed',
15                           'edges', 'size', 'gamma',
16                           'edgeProbability']
17 >>> g.isPerfectGraph()          272
18     True                         273
19 >>> g.exportGraphViz(fileName='randomGraph4335') 274
20     *---- exporting a dot file for GraphViz tools ----* 275
21     Exporting to randomGraph4335.dot                         276
22     fdp -Tpdf randomGraph4335.dot -o randomGraph4335.pdf 277
23

```

If the random perfect graph instance g , shown in Fig. 23.8, is indeed a permutation graph, g and its dual $-g$ are both *transitively orientable*, i.e. comparability graphs (Golumbic 2004). With the `isComparabilityGraph()` test, we may easily check this fact. This method proceeds indeed by trying to construct a transitive neighbourhood decomposition of a given graph instance and, if successful, stores the resulting edge orientations into an `edgeOrientations` attribute (Golumbic 2004, p.129-132).

```

1 >>> if g.isComparabilityGraph():
2     ...     print(g.edgeOrientations)
3     {('v1', 'v1'): 0, ('v1', 'v2'): 1, ('v2', 'v1'): -1, ('v1', 'v3'): 1,
4     ('v3', 'v1'): -1, ('v1', 'v4'): 1, ('v4', 'v1'): -1, ('v1', 'v5'): 0,
5     ('v5', 'v1'): 0, ('v1', 'v6'): 1, ('v6', 'v1'): -1, ('v1', 'v7'): 0,
6     ('v7', 'v1'): 0, ('v1', 'v8'): 1, ('v8', 'v1'): -1, ('v2', 'v2'): 0,
7     ('v2', 'v3'): 0, ('v3', 'v2'): 0, ('v2', 'v4'): 0, ('v4', 'v2'): 0,
8     ('v2', 'v5'): 0, ('v5', 'v2'): 0, ('v2', 'v6'): 0, ('v6', 'v2'): 0,
9     ('v2', 'v7'): 0, ('v7', 'v2'): 0, ('v2', 'v8'): 0, ('v8', 'v2'): 0,
10    ('v3', 'v3'): 0, ('v3', 'v4'): 0, ('v4', 'v3'): 0, ('v3', 'v5'): 0,
11    ('v5', 'v3'): 0, ('v3', 'v6'): 0, ('v6', 'v3'): 0, ('v3', 'v7'): 0,
12    ('v7', 'v3'): 0, ('v3', 'v8'): 0, ('v8', 'v3'): 0, ('v4', 'v4'): 0,
13    ('v4', 'v5'): 0, ('v5', 'v4'): 0, ('v4', 'v6'): 0, ('v6', 'v4'): 0,
14    ('v4', 'v7'): 0, ('v7', 'v4'): 0, ('v4', 'v8'): 0, ('v8', 'v4'): 0,
15    ('v5', 'v5'): 0, ('v5', 'v6'): 1, ('v6', 'v5'): -1, ('v5', 'v7'): 1,
16    ('v7', 'v5'): -1, ('v5', 'v8'): 1, ('v8', 'v5'): -1, ('v6', 'v6'): 0,
17    ('v6', 'v7'): 0, ('v7', 'v6'): 0, ('v6', 'v8'): 1, ('v8', 'v6'): -1,
18    ('v7', 'v7'): 0, ('v7', 'v8'): 1, ('v8', 'v7'): -1, ('v8', 'v8'): 0}
19 >>> g.exportEdgeOrientationsGraphViz('transOrientGraph')          307
20     *---- exporting a dot file for GraphViz tools ----* 308
21     Exporting to transOrientGraph.dot                         309
22     fdp -Tpng transOrientGraph.dot \ 310
23             -o transOrientGraph.png

```

Fig. 23.9 Transitive neighbourhoods of the graph g

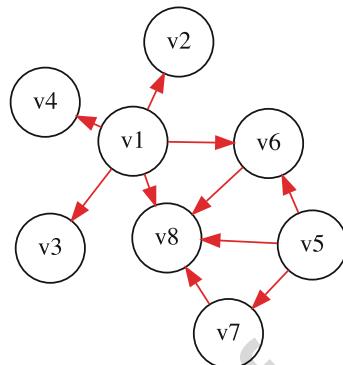


Diagram3 (graphviz), R. Bisдорff, 2019

The resulting orientation of the edges of g , shown in Fig. 23.9, is indeed transitive. The same procedure applied to the dual graph $-g$ gives as well a transitive orientation of its edges shown in Fig. 23.10 on the facing page.

```

1  >>> gd = -g
2  >>> if gd.isComparabilityGraph():
3  ...     print(gd.edgeOrientations)
4  {('v1','v1'): 0, ('v1','v2'): 0, ('v2','v1'): 0, ('v1','v3'): 0,
5  ('v3','v1'): 0, ('v1','v4'): 0, ('v4','v1'): 0, ('v1','v5'): 1,
6  ('v5','v1'): -1, ('v1','v6'): 0, ('v6','v1'): 0, ('v1','v7'): 1,
7  ('v7','v1'): -1, ('v1','v8'): 0, ('v8','v1'): 0, ('v2','v2'): 0,
8  ('v2','v3'): -2, ('v3','v2'): 2, ('v2','v4'): -3, ('v4','v2'): 3,
9  ('v2','v5'): 1, ('v5','v2'): -1, ('v2','v6'): 1, ('v6','v2'): -1,
10 ('v2','v7'): 1, ('v7','v2'): -1, ('v2','v8'): 1, ('v8','v2'): -1,
11 ('v3','v3'): 0, ('v3','v4'): -3, ('v4','v3'): 3, ('v3','v5'): 1,
12 ('v5','v3'): -1, ('v3','v6'): 1, ('v6','v3'): -1, ('v3','v7'): 1,
13 ('v7','v3'): -1, ('v3','v8'): 1, ('v8','v3'): -1, ('v4','v4'): 0,
14 ('v4','v5'): 1, ('v5','v4'): -1, ('v4','v6'): 1, ('v6','v4'): -1,
15 ('v4','v7'): 1, ('v7','v4'): -1, ('v4','v8'): 1, ('v8','v4'): -1,
16 ('v5','v5'): 0, ('v5','v6'): 0, ('v6','v5'): 0, ('v5','v7'): 0,
17 ('v7','v5'): 0, ('v5','v8'): 0, ('v8','v5'): 0, ('v6','v6'): 0,
18 ('v6','v7'): 1, ('v7','v6'): -1, ('v6','v8'): 0, ('v8','v6'): 0,
19 ('v7','v7'): 0, ('v7','v8'): 0, ('v8','v7'): 0, ('v8','v8'): 0}
20 >>> gd.exportEdgeOrientationsGraphViz('transOrientDualGraph')
21 *----- exporting a dot file for GraphViz tools -----*
22 Exporting to transOrientDualGraph.dot
23 fdp -Tpng transOrientDualGraph.dot \
24 -o transOrientDualGraph.png

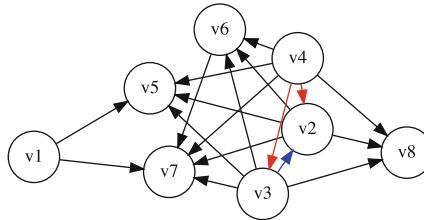
```

Let us recheck these facts by explicitly constructing transitively oriented digraph instances with the `computeTransitivelyOrientedDigraph()` method.

```

1  >>> og = g.computeTransitivelyOrientedDigraph(\_
2  ...                                         PartiallyDetermined = True)
3  >>> print('Transitivity degree: %.3f' %\
4  ...                                         (og.transitivityDegree))
5  Transitivity degree: 1.000
6  >>> ogd = (-g).computeTransitivelyOrientedDigraph(\_
7  ...                                         PartiallyDetermined = True)
8  >>> print('Transitivity degree: %.3f' %\
9  ...                                         (ogd.transitivityDegree))
10 Transitivity degree: 1.000

```



Digraph3 (graphviz), R. Bisdorff, 2019

Fig. 23.10 Transitive neighbourhoods of the dual graph $-g$

It is worthwhile noticing that the orientation of g is achieved with a single neighbourhood decomposition, covering all the vertices, whereas the orientation of the dual graph $-g$ here needs a decomposition into three subsequent neighbourhoods marked in black, red, and blue

The `PartiallyDetermined = True` flag in Lines 2 and 7 above is required 351 here in order to orient only the actual edges of the graphs. Relations between vertices 352 not linked by an edge are put to the indeterminate characteristic value 0. This allows 353 us to compute, later on, convenient disjunctive digraph fusions. 354

As both graphs are indeed transitively orientable, we conclude that the given 355 random graph g is actually a permutation graph instance. Yet, we still need to 356 find now its corresponding permutation. We therefore implement a recipe given by 357 (Golumbic 2004, p. 159). 358

We will first *fuse* both og and ogd orientations above with an *epistemic* 359 *disjunction* operated with the symmetric o-max operator (see Sect. 2.5). 360

```

1 >>> from digraphs import FusionDigraph 361
2 >>> f1 = FusionDigraph(og,ogd,operator='o-max') 362
3 >>> s1 = f1.computeCopelandRanking() 363
4 >>> print(s1) 364
5   ['v5','v7','v1','v6','v8','v4','v3','v2'] 365

```

With the `computeCopelandRanking()` method, we obtain the linear ordering 366 $[v5, v7, v1, v6, v8, v4, v3, v2]$ of the vertices (see Line 5 above). 367

We reverse now the orientation of the edges in og in order to generate, again 368 by disjunctive fusion, the *inversions* that are produced by the permutation we 369 are looking for (see $-og$ in Line 1 below). Computing again a ranking with the 370 COPELAND rule reveals the permuted list of vertices we are looking for (see Line 4 371 below). 372

```

1 >>> f2 = FusionDigraph((-og),ogd,operator='o-max') 373
2 >>> s2 = f2.computeCopelandRanking() 374
3 >>> print(s2) 375
4   ['v8','v7','v6','v5','v4','v3','v2','v1'] 376

```

Vertex $v8$ is put from position 5 to position 1, vertex $v7$ is put from position 2 to 377 position 1, vertex $v6$ from position 4 to position 3, vertex $v5$ from position 1 to 378 position 4, etc. We generate these position swaps for all vertices and obtain thus the 379 required permutation (see Line 5 below). 380

```

1 >>> permutation = [0 for j in range(g.order)]           381
2 >>> for j in range(g.order):                           382
3 ...     permutation[s2.index(s1[j])] = j+1           383
4 >>> print(permutation)                                384
5     [5, 2, 4, 1, 6, 7, 8, 3]                         385

```

It is worthwhile noticing by the way that transitive orientations of a given graph and its dual are usually *not unique* and so may also be the resulting permutations. However, they all correspond to isomorphic graphs (Golumbic 2004). In our case here, we observe two different permutations and their reverses:

```

1 s1: ['v1', 'v4', 'v3', 'v2', 'v5', 'v6', 'v7', 'v8'] 390
2 s2: ['v4', 'v3', 'v2', 'v8', 'v6', 'v1', 'v7', 'v5'] 391
3 (s1 -> s2): [2, 3, 4, 8, 6, 1, 7, 5]                392
4 (s2 -> s1): [6, 1, 2, 3, 8, 5, 7, 4]                393

```

and

```

1 s3: ['v5', 'v7', 'v1', 'v6', 'v8', 'v4', 'v3', 'v2'] 395
2 s4: ['v8', 'v7', 'v6', 'v5', 'v4', 'v3', 'v2', 'v1'] 396
3 (s3 -> s4): [5, 2, 4, 1, 6, 7, 8, 3]                397
4 (s4 -> s3) = [4, 2, 8, 3, 1, 5, 6, 7]                398

```

The `computePermutation()` method does directly operate all these steps: computing transitive orientations, ranking their epistemic fusion, and delivering a corresponding permutation.

```

1 >>> g.computePermutation(Comments=True)           402
2     ['v1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8'] 403
3     ['v2', 'v3', 'v4', 'v8', 'v6', 'v1', 'v7', 'v5'] 404
4     [2, 3, 4, 8, 6, 1, 7, 5]                      405

```

Let us finally check in Fig. 23.11 that the two permutations [2, 3, 4, 8, 6, 1, 7, 5] and [4, 2, 8, 3, 1, 5, 6, 7], observed above, will generate corresponding *isomorphic* permutation graphs.

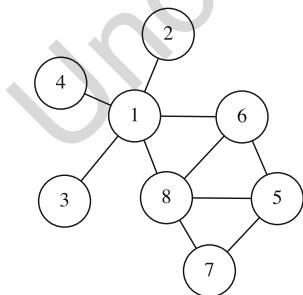


Diagram3 (graphviz), R. Bisdorff, 2019

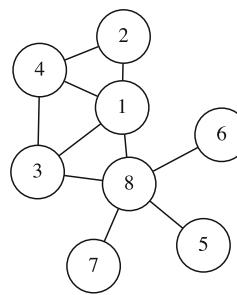


Diagram3 (graphviz), R. Bisdorff, 2019

Fig. 23.11 Isomorphic permutation graphs

```
1 >>> gtesta = PermutationGraph(\ 409
2 ...     permutation=[2, 3, 4, 8, 6, 1, 7, 5]) 410
3 >>> gtestb = PermutationGraph(\ 411
4 ...     permutation=[4, 2, 8, 3, 1, 5, 6, 7]) 412
5 >>> gtesta.exportGraphViz('gtesta') 413
6 >>> gtestb.exportGraphViz('gtestb') 414
```

And, we indeed recover two isomorphic copies of the original random graph 415
(compare with Fig. 23.8 on page 337). 416

References

417

- Berge C (1963) Perfect graphs. Six papers on graph theory. Indian Statistical Institute, Calcutta, pp 418
1–21 419
- Bisdorff R (2010) Enumerating chordless circuits in directed graphs. In: ORBEL24-2010, 24th 420
annual conference of the Belgian operational research society (ORBEL aka Sogesci-B.V.W.B.), 421
January 28–29, Liège (BE), Université de Liège (BE), pp 1–12. <http://hdl.handle.net/10993/23926> 422
423
- Chudnovsky M, Robertson N, Seymour P, Robin T (2006) The strong perfect graph theorem. Ann. 424
Math. 164(1):51–229 425
- Golumbic M (2004) Algorithmic graph theory and perfect graphs. Annals of discrete mathematics, 426
vol 57, 2nd edn. Elsevier, Amsterdam 427

AUTHOR QUERIES

- AQ1.** The sentence “Notice however that the ...” is not clear. Please rephrase the construct for clarity.
- AQ2.** Missing citation for Fig. 23.4 was inserted here. Please check if appropriate. Otherwise, please provide citation for Fig. 23.4. Note that the order of main citations of figures in the text must be sequential.

Uncorrected Proof

Index

A

Anti-holes, 231
AsymmetricPartialDigraph class, 16,
 220
automorphismGenerators(), 316

B

Barbut, M, 36
Berge, Cl, xv
BERGE graph, 329
BestDeterminedSpanningForest
 class, 326
BipolarApprovalVotingProfile
 class, 287
BipolarOutrankingDigraph, 32, 33
BipolarOutrankingDigraph class, 29,
 46
Borda score, 85
Bouyssou, D., 26, 52
BrokenCocsDigraph class, 48

C

checkSampling(), 313
Chordless circuits, 230
cIntegerOutrankingDigraphs
 module, 139
CirculantDigraph class, 11, 230, 314
closeSymmetric(), 21
closeTransitive(), 21, 159
Codual transform, 47
CoDualDigraph class, 20
Coduality principle, 35, 46

Comparability graph, 329
CompleteDigraph class, 24, 228
computeBordaWinners(), 85
computeChordlessCircuits(), 48, 89,
 100
computeChordlessCycles(), 332
computeCondorcetWinners(), 46, 87
computeCopelandRanking(), 339
computeCoSize(), 218
computeDeterminateness(), 11
computeGraphCentres(), 323
computeInstantRunoffWinner(), 85
computeKernelVector(), 244, 247
computeMinimalVertexColoring(),
 334
computeNetFlowsRanking(), 175
computeOrdinalCorrelation(), 217
computePermutation(), 340
computeRankAnalysis(), 85
computeRankingByChoosing(), 94
computeRankingCorrelation(), 101,
 124, 179
computeSimpleMajorityWinner(), 85
computeSize(), 11
computeTransitivelyOriented-
 Digraph(), 338
computeTransitivityDegree(), 11,
 99
computeUninominalVotes(), 85
computeWeakCondorcetWinners(), 46
CONDORCET
 digraph, 98
 winner, 49, 87

C
ConfidentBipolarOutranking-Digraph class, 161
ConverseDigraph class, 20
convertEvaluation2Decimal(), 139
convert2Standard(), 139
convertWeight2Decimal(), 139
Copeland, A.H., 100
CopelandRanking class, 101
cPerformanceTableau class, 138
cQuantilesRankingDigraph class, 144
cRandomPerformanceTableau class, 138
cRandPerfTabs module, 138
cSparseIntegerOutranking-Digraphs module, 141
CycleGraph class, 309, 314

D

Dias, L.C., 112
Digraph class, 8
digraph2Graph(), 306
digraphsTools module, 140
domkernelrestrict(), 244
dreadnaut shell command, 316
DualDigraph class, 20, 231

E

EmptyDigraph class, 24, 229
EquivalenceDigraph class, 215
export3DplotOfCriteria-Correlation(), 182, 220
exportGraphViz(), 9
exportOrientedTreeGraphViz(), 323
exportPermutationGraphViz(), 334
exportRatingByRankingGraphViz(), 193
ExtendedTriangularRandom-Variable class, 69
External stability, 225

F

Filled circuits, 231
FusionDigraph class, 18, 339

G

Gibbs sampler, 306
Golumbic, M.Ch., xvi, 329
GraphBorder class, 17
Graph class, 303

graph2Digraph(), 305
GraphInner class, 17
graphs module, 303
Graphviz, 15, 35
GridDigraph class, 11
GridGraph class, 311

H

Hertz, A., 240
Holes, 226

I

IncrementalQuantilesEstimator class, 126
independentChoices(), 240
IndeterminateDigraph class, 24, 229
IndeterminateInnerPart parameter, 232
IntegerBipolarOutrankingDigraph class, 139
Internal stability, 225
isComparabilityGraph(), 337
IsingModel class, 311
isIntervalGraph(), 330
isPerfectGraph(), 330
isPermutationGraph(), 330
isSplitGraph(), 330
isTree(), 322
IteratedNetFlowsRanking class, 110

K

KemenyRanking class, 105
Kendall, M.G., 36, 212
Kernel, 225
 initial, 228
 prekernel, 228
 terminal, 228
Kohler, G., 109

L

Lamboray, Cl., 112
LearnedQuantilesRatingDigraph class, 129, 191
linearOrders module, 101, 218
LinearVotingProfile class, 83
LineGraph class, 309

M

Majority margin, 86

- Majority margins digraph, 86
MajorityMarginsDigraph class, 86
Marichal, J.-L., 313
Maximal independent choice, 226
MetropolisChain class, 311, 313
MIS, 225
MISModel class, 226
- N**
NetFlowsOrder class, 218
NetFlowsRanking class, 103
Neumann, J. von, 243
numberOfBins parameter, 127
- O**
o-max(), 339
Oppositeness degree, 163
Outranking
 situation, 45
 strict situation, 47
outrankingDigraphs module, 29, 46
- P**
parallel shell tool, 4
Performance criteria, 59
PerformanceQuantiles class, 126, 189
Performance tableau
 coherent, 60
PerformanceTableau class, 43
perfTabs module, 43
Permutation graph, 329
PermutationGraph class, 333
perrinMIS shell command, 315
Pirlot, M., 242
PreRankedOutrankingDigraph class, 120
- Q**
Q_Coloring class, 306
QuantilesSortingDigraph class, 117
- R**
Random3ObjectivesPerformance-Tableau class, 74, 261
RandomAcademicPerformance-Tableau class, 79, 205
RandomCBPerformanceTableau class, 71
RandomDigraph class, 7
- randomDigraphs module, 7, 13
RandomGraph class, 304
RandomIntervalIntersections
 Graph class, 330
RandomLinearVotingProfile class, 83
randomNumbers module, 126
RandomPerformanceGenerator(), 128
RandomPerformanceTableau class, 68
randomPerfTabs module, 67
RandomRegularGraph class, 226
RandomSpanningForest class, 325
RandomSpanningTree class, 324
RandomTree class, 319
RandomValuationDigraph class, 13
RandomValuationGraph class, 326
RankedPairsRanking class, 112
RankingByChoosingDigraph class, 50, 159, 222
RankingsFusionDigraph class, 105, 194
readPerrinMisset(), 316
recodeValuation(), 35
RobustOutrankingDigraph class, 173, 269
Roubens, M., 248
Roy, B., 41
Roy, B., 22, 36, 279
- S**
save(), 14
Sen, A., 97
setEdgeValue(), 333
showActionsSortingResult(), 197
showApprovalResults(), 288
showBestChoiceRecommendation(), 48
showChordlessCircuits(), 48, 100, 173
showCliques(), 308
showComponents(), 14
showCorrelation(), 179, 217
showCourseStatistics(), 80
showCriteria(), 44, 171
showCriteriaCorrelationTable(), 181, 219
showCriteriaQuantileLimits(), 118
showDecomposition(), 121
showDisapprovalResults(), 288
showHTMCriteria(), 154
showHTMLLimitingQuantiles(), 129
showHTMLPairwiseOutrankings(), 159
showHTMLPerformanceHeatmap(), 44, 212

- showHTMLPerformanceTableau()**, 210
showHTMLRatingHeatmap(), 132, 192
showHTMLRelationMap(), 121
showHTMLRelationTable(), 24, 46, 93
showHTMLVotingHeatmap(), 90
showHTMPerformanceTableau(), 76
showLimitingQuantiles(), 127, 191
showLinearBallots(), 84
showMIS(), 307
showMIS_AH(), 240
showNeighborhoods(), 14
showNetApprovalScores(), 289
showObjectives(), 75
showPairwiseComparison(), 33, 49, 64
showPairwiseOutrankings(), 34, 178
showPerformanceTableau(), 31, 62, 130
showPolarisations(), 156, 270
showPreKernels(), 235, 282
showQuantilesRating(), 132, 192
showRankAnalysisTable(), 86
showRankingByChoosing(), 94
showRankingConsensusQuality(), 81, 106, 180, 195
showRelationMap(), 142
showRelationTable(), 14, 32, 63
showShort(), 8, 305
showSorting(), 118
showSortingCharacteristics(), 118, 197
showStatistics(), 10, 72
showTransitionMatrix(), 313
Slater, P., 107
SlaterRanking class, 107
sortingDigraphs module, 117, 191
SparseIntegerOutrankingDigraph class, 141
sparseOutrankingDigraphs module, 120
Split graph, 329
Stability denotation, 263
StabilityDenotation flag, 263
StrongComponentsCollapsed Digraph class, 22
symmetricAverage(), 273
SymmetricPartialDigraph class, 16, 220
- T**
- THE University rankings 2016 by CS subject**, 166
Tideman, N., 112
total_size(), 140
TransitiveDigraph class, 22, 163
transitiveDigraphs module, 50, 102, 103, 159, 222
transitiveOrientation(), 335
tree2Pruefer(), 322
TreeGraph class, 322
Triangulated graph, 329
- U**
- UnOpposedBipolarOutranking Digraph class**, 163, 274
updateQuantiles(), 128
- V**
- votingProfiles module**, 83, 287
- W**
- Warshall, S.*, 22
WeakCopelandOrder class, 102
weakly complete, 46
WeakNetFlowsOrder class, 103
Weak tournament, 87